A Scalable Automated System to Measure User Experience on Smart Devices

Zongyi (Joe) Liu*, Bruce Ferry, Simon Lacasse, Spencer Fonte, Ray Matthieu and Glen Larsen
Amazon.com
Seattle, WA 98104
Email: joeliu*, bferry, lacasse, spencerf, ramatthi, glenlars@amazon.com

***Abstract*—In this paper, we present an automated scalable system that measures user experience on smart devices such as TVs, tablets and smart phones. The system consists of three parts: (i) a robot with a mobile arm to perform touches and clicks on a tested device such as a tablet or a phone, and sensors to capture the video signals, (ii) a signal capturing process records the input video in real time, controlled by algorithms that use a deep detection model and a text matching model to estimate app state, and a deep classification model to navigate, and (iii) a quality metrics computation process that uses spatial and temporal computer vision algorithms. We show that this system can continuously evaluate major streaming apps such as the Prime Video app, popular shopping apps such as Amazon retail app, and other mobile apps. We also do manual validation for a subset of the metrics to evaluate the reliability of our system.**

I. Introduction

Online streaming services such as Prime Video are growing rapidly. It is important to quantify the performance in order to make improvements. Since humans are the customers, one straightforward measuring method is to have a controlled group of people use the services and provide feedback. However, this kind of human rating is not only subjective, but also non-scalable. On the other hand, using a machine to compute the quality of service has been actively studied in recent years. Generally speaking, these metrics can be categorized into three types: (i) full-reference (FR) metrics that require a complete noise-free signal (master). For example, peak signal noise ratio (PSNR), mean square error (MSE), or more complicated perceptual visual quality metrics such as SSIM [1], VDP [2], PEVQ [3], VMAF [4]. (ii) Reduced-reference (RR) metrics that require some samples from the master. For this type, people usually build a model using the limited reference data. For example, M. Tagliasacchi *et. al* [5] modeled the noise in the testing images, and J. A. Redi *et. al* [6] studied the human visual impact using color distortion. (iii) Non-reference (NR) metrics that do not require a master. For example, the motion vectors, colorfulness and spatial information of an image.

In this paper, we present an automated scalable test system: the Device eXperience Automation (DXA) system developed for measuring the user experience. For mobile devices, we developed the Mobile eXperience Automation (MXA) system. Armed with a capacitive touch arm and linear actuators, MXA is able to recreate the tactile input of a customer and capture the important events with a camera. For Living Room devices (e.g., Sony TV), we developed a system called Living Room eXperience Automation (LXA) which includes a camera and instead of a moving arm, an automated remote control. A third system, Capture eXperience Automation (CXA), is specialized to have a smaller footprint using proprietary capture technology in place of a camera, and Bluetooth in place of a mechanical arm. Our system is able to compute both FR metrics such as lost frames, and VMAF and NR metrics such as app loading/playing latency.

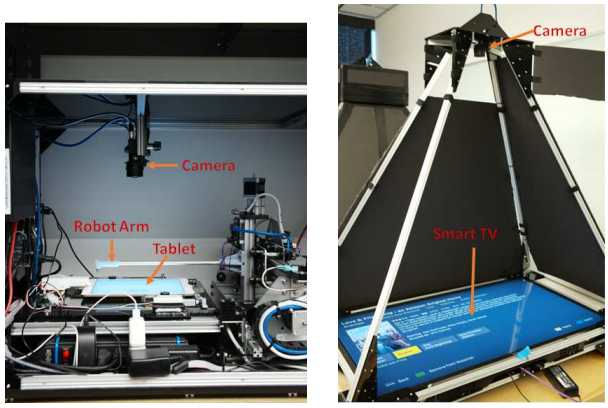
The quality metrics computation consists of two steps: (i) video capturing and (ii) quality analysis. In step (i) we propose two novel algorithms: a matching algorithm that employs the state of art single shot multibox detection (SSD) [7] for logo matching and an alignment distance model for text matching, and a selection tracking algorithm that is based on deep model AlexNet [8] for text region classification. In step (ii) we show that our system can compute metrics including app loading time, streaming services start playing time, retail page rendering time, etc. And we also present an interrupt detection algorithm that consists of temporal motion computation for freeze screen detection, spatial max/min filters for signal cleaning, and Hough transform for circle detection, e.g., a buffering icon spinner.

The paper is organized as follows: Sec. II describes how the robot is built and the sensors for signal capturing, Sec. III describes the algorithms that control the robot in real-time to record videos, Sec. IV describes the algorithms that analyze the playback and compute the number of interrupts occurred, Sec. V evaluates the overall performance of our system including the accuracy and throughput, and Sec. VI concludes the paper and lists future works.

II. Robot System Description

MXA is a self-contained system about $0.5m \times 0.5m \times 1m$ in size with external cables for Ethernet, USB, and AC power. An internal, small form-factor computer running Ubuntu Linux controls the operation of the system and provides a point of access for an operator. The capacitive touch arm is mounted on a 3D motion control platform driven by stepper motors; a CNC motion controller board (TinyG) drives the motors. Also connected to the PC are a high-speed digital video camera and a wifi access point that provides a network dedicated to the device under test (DUT). The DUT itself sits in a dedicated tray that is equipped with linear actuators capable of operating physical buttons on the DUT. The LXA system is simpler: a structural frame which positions the camera over the TV upon

which content and interaction is displayed and recorded. Both MXA and LXA have a micro-controller providing an interface for controlling the actuators via the PC. Fig 1 shows a sample setup for MXA and LXA, respectively.



MXA LXA

Fig. 1. The sample setup of a Mobile eXperience Automation (MXA) and a Living Room eXperience Automation (LXA). Figure is better viewed in pdf or printed in color.

We leverage two distinct video sensors: a camera and an HDMI capture device. The physical camera is the FLIR Flea3 by Point Grey. We're currently using this camera to capture 24-bit color video up to 20 fps at resolutions typically around 1030×1056. And the HDMI capture device allows full HD 1080p capture.

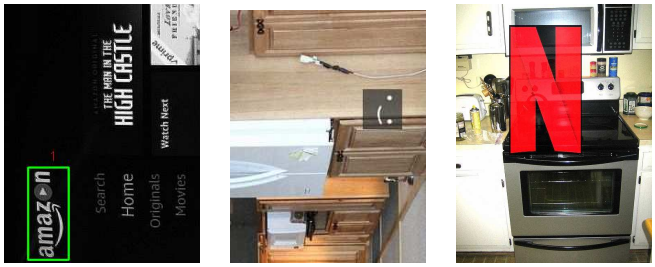
III. Signal Capturing

Our capturing process involves the execution of what we call an "experiment". This experiment executes a series of actions that we feel are representative of normal customer interaction with the applications we're measuring. For example, one experiment could be watching a video from Prime Video. During this experiment, we run machine learning and computer vision algorithms to control our system to take actions such as selecting a movie to watch and pressing play button to begin movie playback. On each action, our system is timestamping each action for later analysis. Once the experiment is complete, the full video, along with all action timestamp information are uploaded to S3 for detailed analysis. In this section, we will describe two key algorithms in the capturing process in details.

A. Logo and Keyword Matching Algorithm for State Estimation

To play the correct content, the robot needs to know the current state of a DUT. For example, have we opened Prime Video app, or are we in the *My List* page of another streaming app? In our system, we estimate the device state by matching logos or keywords from the current captured image.

Our algorithm performs three types of matching: non-text logo matching, keyword matching, and template matching. For logo matching, we use the Single Shot Multibox Detection (SSD) proposed by Liu [7] that performs regional proposal and classification in one pass. To increase the speed, we pick the VGG-16 network [9] with 300×300 input. For the training dataset, we collected around three thousand images captured from the DUTs in our lab, and manually labeled the logo areas, as shown in Fig. 2 (a). In addition, we built a synthesized image database. Briefly speaking, we (i) sample 20,000 images (I_{coco}) from coco dataset [10] 2017 version, (ii) render a logo on a random location of I_{coco}, (iii) add Gaussian noise image with $\mu = 0$ and $\Sigma \in (0, 0.05)$, (iv) add Gamma noise in the range of $(0, 0.15)$, and (v) rotate the image randomly in one of the (0, 90, 180, 270) degrees. Fig. 2 (b) and (c) list two examples of the synthesized images.

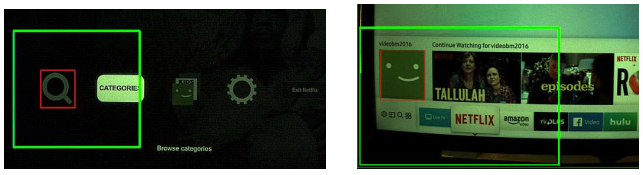


(a) (b) (c)

Fig. 2. Sample images of the training dataset: (a) an image (cropped) captured from our device with *Amazon* logo manually labeled, (b) an image (cropped) with *who's watching* logo rendered and then rotated in 90 degree, and (c) an image with a logo rendered and then rotated in 0 degree.

After the logo detector model is trained, we run forward prediction on the current captured image. For detected logos, we will then check their location in order to increase the accuracy. For example, we have observed that when we are in Prime Video app, the *Prime* logo is more likely to appear on the top left of the page. Today, we pre-define the location for every logo for each state (screen) of interest. Fig. 3 shows two matching examples, where we use the *overlay* logo to determine if we are in the app exit page, and the *who is watching* logo to determine if we have selected the correct icon.

For keyword matching, we first use an OCR service to perform text recognition. Then we group the detected words using the alignment distance (D_{align}) defined in Eq. 1 and 2. Two words are merged into one group if their distance is less than 0.5, and we treat a group as a new word and repeat this process until it converges. Then for each text group, we compute its Levenshtein distance D_{lev} to the pre-defined text token, and if D_{lev} is small, we report a match. Similar to logo



(a) (b)

Fig. 3. Sample output (cropped) of SSD based logo detection for (a) overlay logo, and (b) who is watching logo. The green bounding box is the pre-defined location constraint, and the red bounding box is the detection reported by SSD. Figure is better viewed in pdf or printed in color.

detection, we also pre-defined a matching location for every keyword.

$$D_{align}(W_1, W_2) = \begin{cases} D_{align(V)}(W_1, W_2), & \text{if } D_{align(H)}(W_1, W_2) < T_H \\ \infty, & \text{Otherwise} \end{cases}$$
$$T_H = 2 \times min(Width(W_1), Width(W_2)) \quad (1)$$

$$\begin{aligned} D_{align(V)}(W_1, W_2) &= \frac{Height(W_1) \bigcap Height(W_2)}{min(Height(W_1), Height(W_2))} \\ D_{align(H)}(W_1, W_2) &= max(L_{W_1}, L_{W_2}) \\ &\quad -min(R_{W_1}, R_{W_2}) \end{aligned} \quad (2)$$

where L_{W_i}, R_{W_i} are the left border and right border of word W_i, respectively.

In addition to logo and text matching, we also use the template matching algorithm as implemented in OpenCV. Specifically, we first pre-create a template image I_{tp}, define a region of interest ROI_{tp} and a matching threshold T_{tp}. Generally speaking, the size of I_{tp} is less than ROI_{tp}, which in turn is less than the captured image I_{in}. Then we scroll I_{tp} within the ROI_{tp} area of I_{in}, and compute the normalized Euclidean distance D_{eu} at each location. And we report a match if the minimum of D_{eu} is less than T_{tp}. Though this approach is sensitive to illumination and orientation changes, a template can be created quickly so that we can add a new device or service into our production line first, and then replace it with a more robust SSD model.

B. Selection Tracking Algorithm

For mobile DUTs, the robot can search the logos and keywords using the algorithm described above, and then click the detected regions. However, for living room DUTs that don't have a touch screen, the robot will need to send remote command such as *move left* or *ok* via the blue-tooth or IR channel. So it is important for the system to know which menu is currently selected and which menu we should go next.

For Prime Video app, the selected menu is usually rendered with orange foreground or background as shown in Fig. 5. Based on the cuda-convnet [8], we build a classifier with three convolution layers, one fully connected layer, and three output classes: color foreground texts, color background texts, and other texts. We also pre-process an input image to enhance the contrast for text regions. Specifically, given a text region i (R_i), we first compute the color histogram in HSV channel within R_i. Then we scale up the V value to [0, 255], and then convert it back to RGB channel. Fig. 4 shows the sample images of the enhancement. We can see that it improves the images from two DUTs, particularly the one collected from a Samsung TV that is very dark. Fig. 5 lists the classification outputs of frames captured at different states from LXA DUTs. It shows that our model has correctly classified the selected menus.

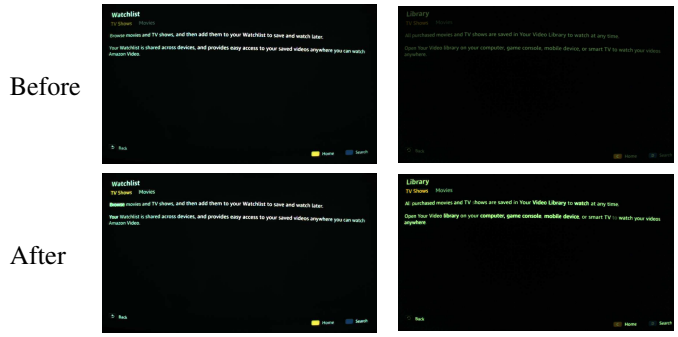


Fig. 4. Sample of images before and after color enhancement. Figure is better viewed in pdf or printed in color.

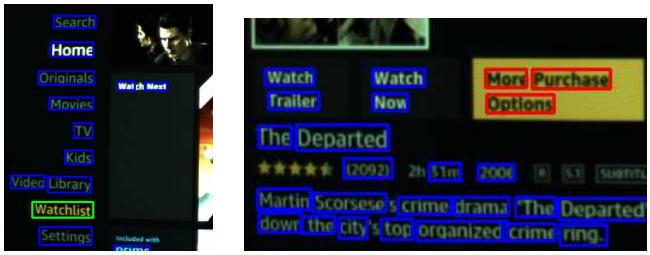


Fig. 5. Samples (cropped) of word classification for captured images from Prime video app. Here texts classified as type 0 are boxed with blue line, type 1 are boxed with green line, and type 2 are boxed with red line. Figure is better viewed in pdf or printed in color.

For some apps, a selected menu is boxed with a white rectangle. So we first apply the Canny algorithm [11] to detect edges, then search contours from the edges. Here we need to be careful because images with low intensity value may have broken edges. To address this problem, we also do color enhancement first before the edge detection step. And we filter a contour (CT) if the area ratio between CT and its min area rectangle is less than a threshold T_R. In our algorithm, we set T_R to 0.9. For overlap or nested contours, we will pick the outermost one. Then we label the texts inside a detected contour as being selected. Fig 6 shows sample outputs, and we can see that the algorithm correctly identifies the selected menu.

IV. Quality Evaluation

After the video capturing, we start to evaluate the data quality. Today, our system can compute the metrics for different types of applications on smart devices:

- Major online streaming apps such as Prime Video: (i) whether interrupt(s) occurs during a movie play, e.g, frame freezed or dropped frames, (ii) time for the application to launch, (iii) time for the video to start playing, (iv) percentage of sessions that interrupt occurred, and (v) user perceived quality score per frame (VMAF) [4]

Vizio TV PS4

Fig. 6. Samples (cropped) of word classification for captured images from a video streaming app. Here words classified as selected are boxed with green line, and words classified as not-selected are boxed with blue line. Figure is better viewed in pdf or printed in color.

- Popular Retail shopping apps such as Amazon online shopping: (i) time for the app to launch, (ii) time for search results to appear, and (iii) time for detail page to load.
- Kindle app: (i) time for the app to launch to home screen, (ii) time to open a book from library, and (iii) time to open the app when it was closed with book open.

In this section, we will mainly describe the algorithm we used to measure whether interrupt happens. We observe that most of time when this happens, a loading spinner will be displayed in the center area of the screen. So in our algorithm, we detect spinners from the video playback. The algorithm can be described in two phases. In phase one we identify frames that have high motion in the center area, and low motion in the border area of the screen. This is based on our observation that when an interrupt occurs the video is frozen except for the spinner. And our model works as defined in Eq. 3: given an $I_{motion}(t)$, we label it as a high motion frame ($I_{hm}(t)$) if its MO_{in} is greater than $T_{MO}(high)$ and MO_{out} is less than $T_{MO}(low)$, where $T_{MO}(high)$ and $T_{MO}(low)$ are empirically set.

$$
\begin{aligned}
I_{motion}(t) &= min_W(I_{intensity}(t) - I_{intensity}(t-1)) \\
MO_{in} &= \forall_{x,y \in ROI_{sp}}(max(I_{motion}(x,y))) \\
MO_{out} &= \forall_{x,y \notin ROI_{sp}}(max(I_{motion}(x,y)))
\end{aligned} \tag{3}
$$

where min_W is a two-dimensional min-filter with a window size of W, and ROI_{sp} is a pre-defined center region of I_{motion} where a spinner usually appears.

Once we identify a high motion frame at time t, we start our phase two computation. Instead of using the deep model SSD [7] algorithm that takes longer time to compute, we pick a primitive approach that is more scalable on a CPU machine. Specifically, we create an accumulated motion image (I_{accmo}) from the ROI_{sp} of $I_{motion}(t)$ and its temporal neighbors, as shown in Fig. 7. Then we run Hough transform on I_{accmo}, and report a match if a circle is detected. Fig. 8 lists two sample images where we detect a spinner. We can see that the algorithm works decently for images of different intensity levels.

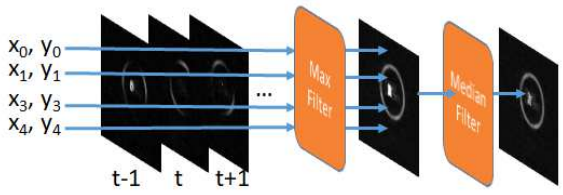


Fig. 7. An example of generating I_{accmo} from the ROI_{sp} of $I_{motion}(t)$ and its temporal neighbors. Note the *max filter* here is in temporal and the *median filter* here is in spatial.

Fig. 8. Sample images (cropped) that we have detected an interrupt spinner. Figure is better viewed in PDF or printed in color.

V. System Performance

Today, we have built our system to measure video/shopping app services 24/7 on 20+ smart devices including phones, tablets, TVs, TV sticks, game consoles and PCs. This is a scalable system that runs 130 - 150 experiments (video captures) each day on one smart device. And for each experiment it produces multiple quality metrics, including time to load an app, time to start streaming a movie, time to render a webpage, etc. To have an understanding how reliable these automatically generated metrics are, we perform manual validation on a subset of them. Specifically, we sort the metrics using a ten-day window, sideline the experiments in the top and bottom 5% to a manual validation channel. For the remaining 90%, we also randomly sample 5% and perform manual inspection. And we mark an experiment as incorrect if any of its metrics are labeled as error during this process. Table I lists manual validation results for the Prime Video app. To ensure our system is not over-engineered for Amazon apps, we also manually inspect the metrics of other popular streaming apps. And we can see that the overall reliability is pretty good: the average error rate is only 1.3% for Prime Video and 2.4% for others, respectively. Of course, there are a few outliers. For example, the Prime Video app on Fire Tablet 7 has an error rate of 7%. This is mainly caused by the very low intensity of the captured images. And we are working on increasing the camera exposure rate to improve the accuracy.

VI. Conclusions and Future Work

In this paper, we presented an automated scalable quality measurement system to evaluate user experience on smart devices. It consists of a robot to perform touches and navigation on a DUT (smart device), a brain (local PC) that runs matching and selection tracking algorithms to capture signals from the

TABLE I

THE PER-DEVICE EXPERIMENT ERROR RATE IN PERCENTAGE FOR PRIME VIDEO (**PVR**) AND OTHER STREAMING SERVICES (**OTR**), AND THE TOTAL NUMBER OF MANUALLY INSPECTED EXPERIMENTS IN THOUSAND (**PVC** AND **OTC**). THESE EXPERIMENTS ARE INSIDE A SIX-MONTH TIME WINDOW

Device	Fire Tablet 7	i-pad	i-phone	Fire TV	LG TV	PS4	Roku	Samsung Phone
PVR	7	1	2.5	1	2	1	0.5	0
PVC	0.7	1	0.6	1.1	0.4	0.1	1.2	0.3
OTR	1	1.3	0.5	4.5	1.5	3.5	2.5	0
OTC	1	1	0.7	0.7	0.9	0.2	0.9	0.2

Device	Samsung TV	Sony Blue Ray	Sony TV	Windows 10	Xbox One	**Average**
PVR	0.5	0	0	1	1	**1.3**
PVC	0.1	0.2	1	1.1	0.8	-
OTR	6.5	1.5	0.5	6.5	1.5	**2.4**
OTC	0.1	0.4	0.8	0.8	0.9	-

DUT, and a set of computer vision algorithm to compute quality metrics. We showed that the system can generate over a thousand metrics for one DUT, and these metrics are highly reliable as proved by the manual validation on experiments inside a six-month window.

In terms of future work, first, we need to improve the quality of input signal from a camera: today, we manually zoom the camera and set exposure rate using human eye perception. This process is highly subjective, and we plan to improve it by using the real-time computed metrics, such as PSNR and colorfulness, as a guidance. Second, we will investigate a more robust navigation algorithm during the data capturing process that is able to deal with various exceptions such as network slowness or content not available. The algorithm should also be able to find texts/logos and their locations using machine learning instead of being pre-defined by human. We plan to build a finite state transition model based on computer vision, to provide more reliable estimation than the current single frame based technique. Third, we will study if we can compute more complicated quality metrics, such as video stuttering w/o a master video or interrupt occurs w/o displaying a spinner on the screen. Finally, we will be adding additional input signals, such as sound, to increase the accuracy of our measures and generate sound quality metrics.

REFERENCES

[1] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

[2] Daly and Scott, "Digital images and human vision," A. B. Watson, Ed. Cambridge, MA, USA: MIT Press, 1993, ch. The Visible Differences Predictor: An Algorithm for the Assessment of Image Fidelity, pp. 179–206.

[3] "Objective perceptual multimedia video quality measurement in the presence of a full reference," p. 108, August 2008.

[4] Z. Li, A. Aaron, I. Katsavounidis, A. Moorthy, and M. Manohara, "Toward a practical perceptual video quality metric," June 2016.

[5] M. Tagliasacchi, G. Valenzise, and M. e. a. Naccari, "A reduced-reference structural similarity approximation for videos corrupted by channel errors," in *Multimed Tools Appl*. Springer US, July 2010, vol. 48, pp. 471–492.

[6] J. A. Redi, P. Gastaldo, I. Heynderickx, and R. Zunino, "Color distribution information for the reduced-reference assessment of perceived image quality," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 12, pp. 1757–1769, Dec 2010.

[7] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *ECCV*, 2016.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[10] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[11] C. J., "A computational approach to edge detection," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 6, pp. 679–698, 1986.