ADVISOR: An Adjustable Framework for Test Oracle Automation of Visual Output Systems

A. Esat Genç, Hasan Sözer, M. Furkan Kıraç, Barış Aktemur

***Abstract*—Test oracles differentiate between the correct and incorrect system behavior. Automation of test oracles for visual output systems mainly involves image comparison, where a snapshot of the output is compared with respect to a reference image. Hereby, the captured snapshot can be subject to variations such as scaling and shifting. These variations lead to incorrect evaluations. Existing approaches employ computer vision techniques to address a specific set of variations. We introduce ADVISOR, an adjustable framework for test oracle automation of visual output systems. It allows the use of a flexible combination and configuration of computer vision techniques. We evaluated a set of valid configurations with a benchmark dataset collected during the tests of commercial Digital TV systems. Some of these configurations achieved up to 3% better overall accuracy with respect to state-of-the-art tools. Further, we observed that there is no configuration that reaches the best accuracy for all types of image variations. We also empirically investigated the impact of significant parameters. One of them is a threshold regarding image matching score that determines the final verdict. This parameter is automatically tuned by offline training. We evaluated runtime performance as well. Results showed that differences among the ADVISOR configurations and state-of-the-art tools are in the order of seconds per image comparison.**

***Index Terms*—test automation, test oracle, black-box testing, computer vision, adjustable framework**

I. Introduction

Automation of testing activities is a commonly preferred approach for reducing the costs of testing [1], [2], which can account for at least half of the overall development costs [3]. One of these activities is performed by a test oracle [4], which differentiates between the correct and incorrect system behavior. An analysis of the literature [5] reveals that test oracle automation has received significantly less attention compared to the automation of other testing activities. However, it is a necessity for achieving overall test automation. Otherwise, one has to check the system behavior for all test cases manually even if these test cases are generated and executed automatically.

Test oracle automation becomes a straightforward comparison task if formal specifications regarding the intended system behavior exist [6]. However, such specifications are not always available and a trivial comparison is not usually effective, especially when the expected output takes complex forms such as an image [7]. Accordingly, test oracle implementations for visual output systems tend to be fragile and they lead to many *false positives* [8], where an error is reported although an error does not exist, as well as *false negatives*, where an error is not reported even though exists.

A common implementation of test oracles for visual output systems involves image comparison, where a snapshot of the observed output during test is compared with respect to a previously taken reference image. Hereby, the captured snapshot can be subject to several variations due to for instance, scaling, shifting, rotation or color saturation, depending on the method used for capturing the image. These variations lead to false positive evaluations, necessitating technical staff to manually examine the cases and thus incurring labor costs. There are many techniques available in the computer vision domain to address such issues and perform an effective comparison between images. Existing approaches in the literature [9], [10] employ a combination of these techniques to address a specific set of variations for a dedicated test oracle implementation. However, some of these techniques might not be the most effective one for addressing a particular variation, while some other techniques might not be necessary in the absence of a particular variation and as such, introduce an unnecessary performance overhead. They are also liable to increase false negatives if not tuned properly while attempting to decrease false positives, causing important bugs to be missed. Hence, a generically applicable, efficient and effective test oracle implementation for visual output systems must be configurable to employ and tune the most effective techniques from the computer vision domain based on the application context and the test setup.

In this paper, we introduce ADVISOR, an adjustable framework for test oracle automation of visual output systems. ADVISOR allows the use of a flexible combination and configuration of alternative techniques from the computer vision domain. Hereby, we did not develop a novel image comparison algorithm from scratch. Hence, we do not introduce theoretical contributions for the image processing and computer vision domains. Our main contribution is the analysis of these domains and compiling the state-of-the-art techniques for leveraging test oracle automation. We reviewed these techniques in terms of their pros and cons for applicability and composability in the context of software/system testing. Then, we implemented them as part of our framework. To the best of our knowledge, there does not exist such a generic framework for test oracle automation of visual output systems.

We evaluated several instances of our framework with respect to state-of-the-art tools. We used a benchmark dataset, which includes 1000 image pairs that are collected during the tests of real Digital TV systems [11]. These image pairs are manually labeled to distinguish those pairs that are associated with failures. Although more than half of the image pairs are not actually associated with failures, they are subject to variations that can result in failing tests (i.e., false positives). These image pairs are further categorized with respect to 3 types of variations they involve: *i)* pixel shifting, *ii)* scaling, and *iii)* color saturation. Each image pair that is not associated with any failure, is subject to either one of these variation types. Hence, we were not only able to evaluate the overall accuracy of alternative test oracle implementations, but also their accuracy for image pairs that are particularly subject to these types of variations.

Results show that the accuracy of a test oracle can be improved if the involved techniques are selected and fine-tuned for the application context. Several instances of ADVISOR achieved up to 3% better overall accuracy compared to the state-of-the-art tools. These instances are obtained by combining various techniques for image transformation and image matching. A set of invalid combinations are not allowed by the tool. We evaluated the remaining, feasible configurations in our experiments. We observed that there is no configuration that reaches the best accuracy for all the 3 types of image variations. This observation proves the need for a configurable framework. One should be able to (de)select techniques based on the test setup and the frequency of image variation types taking place in the captured images. In our experiments, we used the default values for almost all the parameters for each of the utilized techniques. These are the parameters that do not significantly impact the results. However, we empirically investigated the effects of the other parameters in more detail. One of these parameters is the *threshold* of image matching score that determines the pass/fail verdict of the oracle. ADVISOR sets this parameter automatically during a preliminary (offline) training phase.

There might exist a trade-off between accuracy and runtime performance if the overall test process is not fully automated. However, we observed that the most accurate tool or configuration is not necessarily the slowest one. Also, results showed that differences among the evaluated tools and configurations are in the order of seconds per image comparison.

The remainder of this paper is organized as follows. In the following section, we summarize the related studies. In Section III, we present a domain analysis of the relevant techniques borrowed from the computer vision domain and depict the solution space, which provides a common ground for creating alternative test oracle implementations. In Section IV, we explain the implementation of ADVISOR. In Section V, we present an empirical evaluation of our framework, where we compare a set of its instances with respect to the state-of-the-art tools. Finally, in Section VI, we provide our conclusions.

II. Related Work

Systems that provide graphical user interfaces (GUI) can be considered as an important category of visual output systems. Hence, GUI testing techniques are related to our work. These techniques have been investigated for more than two decades [12] at the time of writing this paper. The majority of these techniques focus on the modeling and verification of functional behavior rather than GUI appearance and they are not purely black-box testing techniques. They run on the same machine as the system under test [13], [14] and they assume that GUI components (e.g., buttons, labels) or a document object model (e.g., as in HTML) for Web applications [15], [16] are available. However, this assumption may not hold for all types of systems. For instance, testers do not usually have any access to the internal events during the testing of embedded systems such as those from the consumer electronics domain. They do not have any access to the GUI components either. Such components and a static structure regarding their organization are not externally visible for some systems like Digital TVs. Hence, the visual output that is observed on the screen has to be validated in a black-box fashion.

Automated test oracles that employ image comparisons have been proposed for pure black-box testing [17]. This approach has become popular among researchers in the last few years in particular [8], [10], [18]–[21]. Many of these recent studies focus on Web applications [18]–[21]. For example, Selay et al. proposed the use of image comparisons to detect layout failures in these applications [18], [19]. The proposed technique utilizes previously observed failure patterns and compares a selected set of regions in the compared images. However, it assumes that these images are not subject to any variations due to, for instance, scaling or color saturation. Therefore, any difference detected among the selected regions is deemed as a failure. Our framework can be configured to take such variations into account, if there are any expected. Mahajan and Halfond also aimed at detection of presentation failures in Web applications [20], [21]. Their first study [20] employs pixel-to-pixel comparison and ignores variations among images as well. Later, they propose the use of *perceptual image differencing* [22] to compare images [21]. This technique takes a particular set of variations into account to avoid spatial and luminance sensitivity in comparisons. Mahajan and Halfond used an external tool, *pdiff*[1], for implementing their approach. Unlike our generic framework, this tool considers spatial and luminance sensitivity only. It was also shown to be substantially inefficient in terms of running time with respect to other recent test oracle implementations [10], [11]

Sub-image searching was used in a visual testing tool called Sikuli [23], where test scripts and assertions can be specified via a set of keywords and images of GUI elements. These images are searched within a Web page, and assertions can lead to failure based on their (non-)existence. This tool, however, facilitates the implementation of specific assertions only and it also ignores variations among images that do not actually indicate a failure.

[1]http://pdiff.sourceforge.net

Image comparison has been used for automating test oracles in other application domains as well. For instance, such an approach was used in automative industry [24]. Hereby, snapshots of the interactive display that is presented to drivers are taken during tests. These snapshots are compared with respect to a specification that defines the layout of the display as well as the set of icons and textual information expected to be displayed. The comparison involves a set of specialized techniques; *i)* pixel-to-pixel comparison for icons, *ii)* optical character recognition for textual information, and *iii)* custom visual feature extraction for complex display items such as the level of a gauge. The tool was employed in a simulation environment during model-in-the-loop tests. As a result, captured images are not subject to any variations in that context. In this study, we aimed at providing a generic solution rather than a tool dedicated for a particular context and application domain.

Automated test oracles can reach to a verdict based on a similarity measure calculated with respect to the compared images [8], [25]. The similarity measurement is defined based on a set of keypoints extracted from these images. These keypoints may relate to the color, texture, and shape of objects. This approach has been applied mainly for desktop applications and Web applications. Our framework incorporates a variety of keypoint extraction and comparison metrics rather than relying on a particular similarity measure only.

Efficiency and reusability of automated test oracles have been recently evaluated for android devices [9]. In the experimental setup, snapshots of the mobile device screen are taken via an external camera. These snapshots are compared with respect to reference images. 3 different kinds of image comparison techniques are implemented/used: *i)* SURF (Speeded up robust features), *ii)* Histogram matching, and *iii)* Template matching. We compared the accuracy of this tool with respect to several instances of our framework (Section V).

We previously implemented an automated test oracle [10] called VISOR for testing visual output systems. This tool employs an image processing pipeline for comparing images. It includes a series of image filters that align the compared images and remove noise to eliminate differences caused by scaling and translation. Hence, VISOR provides an efficient but a dedicated solution for addressing scaling and translation variations only. In this work, we introduce a configurable framework that can employ a combination of available techniques in the computer vision domain tuned to address any set of variations. We evaluated several instances of this framework by comparing its accuracy with respect to VISOR as well as other tools previously employed/implemented for test oracle automation based on image comparison.

There exist a recent survey [5] conducted for analyzing and categorizing test oracles proposed so far. Our framework supports the development of so-called *specified test oracles* according to the proposed classification. Hereby, the evaluated image is compared with respect to a specification, which is also provided in the form of an image, called the reference image. Hence, our approach adopts so-called *visual assertions* according to a previously made classification of test oracles used for GUI testing [26].

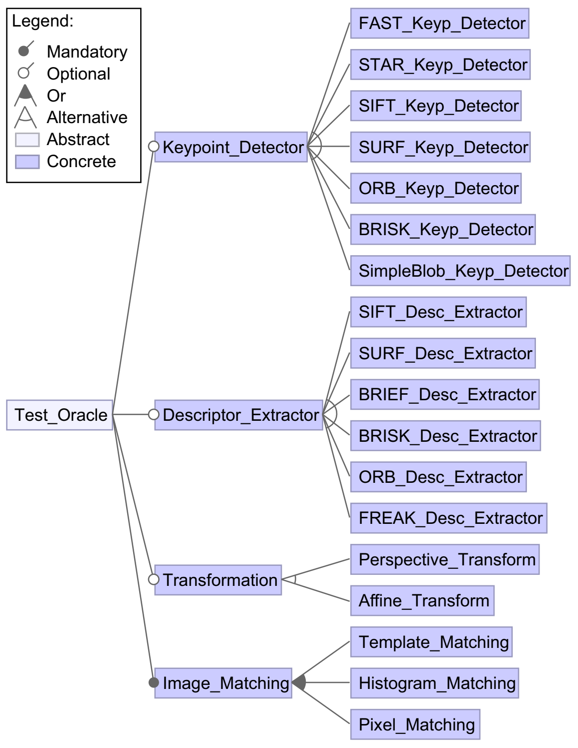
III. Domain Analysis

We performed an analysis summarizing the Computer Vision based methodologies related to image comparison. Domain analysis is an essential step in software product line engineering [27] to identify commonalities and variations among the products of a product family. The commonalities and variations are represented by means of a so-called *feature model* [28]. A *feature diagram* depicts this model, which is a compact, visual representation of all the products in the product family.

We created a feature model for ADVISOR as a result of our domain analysis. ADVISOR represents a family of test oracle implementations based on image comparison. Each implementation employs a set of techniques to perform the comparison. The feature model defines the available techniques that can be employed as well as the constraints among them (e.g., two techniques must be used together or one technique cannot be used with some other technique). Therefore, it also determines the possible set of test oracle implementations that can be instantiated with ADVISOR.

The feature diagram of ADVISOR is depicted in Figure 1. This diagram describes the solution space. Common features among all the test oracle implementations are modeled as mandatory features. Possible variations among these implementations are captured by the set of optional features and the hierarchical structure. Exactly one feature out of the ones that are bound with the *Alternative* connection must be chosen. At least one or more features must be chosen if they are connected to their parent with the *Or* connection. The selection of an optional or alternative feature can require the *inclusion* or *exclusion* of another feature. Hence, a feature diagram is provided together with a set of constraints as listed at the bottom of Figure 1. All the constrains must be satisfied by the overall selection of features. Therefore, they form a predicate altogether with a logical AND operator (i.e., $\wedge$) among them. All the constraints listed here are of type *inclusion*. That is, selecting an optional feature requires the selection of another optional feature that it depends on. This relation is expressed with the $\Rightarrow$ symbol. For instance, employing *transformation* makes it necessary to employ *descriptor extractor* and this in turn, makes it necessary to employ *keypoint detector*. Since these three features are always used together, we defined an inclusion constraint between each pair of these features in both directions.

We can see in Figure 1 that a test oracle involves four main features each of which can be implemented using a set of alternative or complementary algorithms. *Image matching* is a mandatory step for comparing the captured and the reference images with each other. The remaining three features are optional steps that are used for transforming images before the comparison is performed. In principle, only the *image matching* feature is obligatory to implement a test oracle based on image comparison. If one knew that the captured images are by no means subject to any variation at all (even if this would be rarely the case), then it would be unnecessary to employ any transformation process. Images can be directly compared with each other via *pixel matching*, for instance. Features other than



(Keypoint_Detector ⇒ Descriptor_Extractor) ∧ (Keypoint_Detector ⇒ Transformation)
(Descriptor_Extractor ⇒ Keypoint_Detector) ∧ (Descriptor_Extractor ⇒ Transformation)
(Transformation ⇒ Keypoint_Detector) ∧ (Transformation ⇒ Descriptor_Extractor)

Fig. 1: Feature diagram of ADVISOR.

image matching are left optional not to restrict the framework unnecessarily and keep it as generically applicable as possible.

A captured image may be subject to several changes such as translation, rotation, scaling; depending on the capturing method. Images modified by such transformations are needed to be transformed back into the same planar surface with corresponding reference images, making them aligned before comparison. To facilitate such a transformation, keypoints of an image is extracted and utilized. Hence, an image is described as a combination of features[2], namely keypoints. Each keypoint has a location in the image. A keypoint's surroundings is also described with a so-called keypoint descriptor. The descriptor is designed in such a way that it is resilient to different image transformations such as illumination change, translation, rotation, scaling, etc. Locations of important keypoints are detected by using *keypoint detector* which is also called a *feature detector*. Surroundings of the extracted keypoint locations are described by using *descriptor extractor* algorithms. Finally, an alignment transformation is computed based on the keypoint locations and descriptors. The image is then transformed by using *transformation* algorithms for aligning it with the reference image.

We discuss algorithms for keypoint detection, descriptor extraction, image transformation and image matching in further detail in the following subsections.

A. Keypoint Detection and Descriptor Extraction

An image may be represented by local keypoints and global keypoints [29]. Local keypoints include corners, edges and lines that take place in images [30], whereas global keypoints include contour representations, shape descriptors, and texture [31]. ADVISOR involves many keypoint detector alternatives including FAST [32], Star [33], SIFT [34], SURF [35], ORB [36], BRIEF [37], BRISK [38] and SimpleBlob that find local keypoints of an image. Implementations are available as part of the OpenCV library[3]. Note that some of the above algorithms support both keypoint detection and keypoint description whereas others are only capable of detection or description.

A descriptor acts like a fingerprint that differentiates a keypoint's surroundings from others. It is represented as a vector, which contains information on the surrounding (neighborhood) pixels of a specific location in an image. Different descriptors use different window sizes and techniques for representing the neighborhood. Nevertheless, all of them are designed for being robust against different possible transformations that an image can undergo. Some of these transformations include translation, scaling, rotation, illumination changes, compression artifacts. This means that even if an image is under severe transformation, an important keypoint in an image can still be detected and its surroundings can still be described with a relatively similar description vector. Since descriptors constitute invariant representations of an image patch regardless of transformations, they are commonly used for supporting the image alignment and matching process. ADVISOR uses BRIEF [37], BRISK [38], FREAK [39], ORB [36], SIFT [34] and SURF [35] as descriptor extractors.

We briefly review the keypoint detection and descriptor extraction techniques employed in ADVISOR in the following subsection.

1) FAST: Features from Accelerated Segment Test (FAST) [32] is a keypoint detector that is developed for real-time applications. FAST uses machine learning for high speed corner detection. A so-called segment test criterion selects a candidate circular region with sixteen pixels around a candidate corner. Candidate corner is categorized based on its relative intensity with respect to a circular region of adjacent pixels around it by a predefined threshold level. There are three different categories, namely; darker, similar and brighter. For instance, a candidate corner is classified as darker if the difference of its intensity level is greater than all the pixels in the circular region and the amount of this difference is larger than the threshold value. A decision tree [40] with an entropy optimizer is employed to determine if a pixel is a corner or not. The main strength of the FAST

[2]We need to clarify the two different uses of the term "feature" in this paper. Firstly, this term is used for defining a functionality or capability of a product in software product line engineering. Secondly, the same term refers to a measurable property or characteristic of images in computer vision domain. We adopt the term keypoint instead of this second use to prevent confusion in the rest of the paper.

[3]http://opencv.org

algorithm is its speed. However, it is difficult to determine an optimal threshold level, especially when the image is subject to a high level of noise. Moreover, it is also not scale invariant. As a result, it may not be effective for noisy images that are subject to scaling.

2) Star: Star keypoint detector is derived from the Center Surrounded Extrema (CenSurE) detector [33]. The aim of this keypoint detector is to find extrema in different scales and locations. It uses bi-level filters that multiply intensity values of image pixels with 1 or -1 to reduce the computational load. These filters take the form of circles, octagons, hexagons and boxes. Filter performance depends on its form, which is subject to a trade-off. For instance, octagon filter has a better accuracy in detecting keypoints, whereas box filter has better computational speed. Non-maxima suppression method is used for various scales to find candidate local keypoints. Detected keypoints are further filtered by a method such as Harris corner detector. The remaining points are deemed as the local keypoints by Star keypoint detector.

3) SIFT: Scale Invariant Feature Transform (SIFT) [41] is used for both keypoint detection and descriptor extraction. Scale space theory [42] is employed for keypoint detection. This approach finds keypoints by utilizing a Gaussian pyramid. Algorithm works in various scales of the same image starting from a lower resolution version and going step by step to a higher resolution representation. Local extrema is detected by comparing a sample point with its neighbors in its below and above resolution scales. After candidate keypoints are found, accurate localization of keypoints are computed. Finally, neighborhood pixels are assigned to a histogram's bin by quantizing the underlying edge's orientation. This method is called Histograms of Oriented Gradients [43] (HOG). The histogram bin with the maximum value is chosen as a normalization point. Histogram is rotated in such a manner that the bin with the maximum value always stays at the leftmost bin. This provides rotational invariance. Histograms are extracted around the keypoint location in a 4x4 grid created in a 16x16 pixels patch around the keypoint. Each histogram is represented by 8 bins. 16 histograms are extracted. Each bin is represented as an 8 bit number. This gives a 128 byte description vector that describes the 16x16 patch around a keypoint. SIFT keypoints are computationally expensive to obtain but they lead to better accuracy compared to other keypoint detection methods.

4) SURF: Speeded Up Robust Features (SURF) [35] is mainly based on SIFT. Keypoint detector part of this algorithm is based on Hessian matrix which credits integral images to decrease computational complexity and as such improve the performance. Hereby, the determinant of the Hessian is called as Fast-Hessian detector and it is employed to find location and scale. The Gaussian distribution is approximated by using second-order Gaussian derivatives (Laplacian of Gaussians), which are evaluated by integral images. This approach leads to a faster computation. In addition, the same integral images are utilized for calculating box filters of any size in parallel without creating different resolution representations of the same image. SURF keypoint detectors are faster that SIFT but the descriptors are slightly less performant representations of the neighborhood when compared to SIFT.

5) BRIEF: Binary Robust Independent Elementary Features (BRIEF) [37] is used for extracting descriptors in the form of bit vectors. The trade-off between the speed and accuracy can be controlled via the length of this vector. Because of its sensitivity to noise, a preliminary Gaussian smoothing with 9x9 window size is applied for erasing high frequency information from the image. A preliminary pixel pair list to be compared is created and hardcoded in the algorithm. This may be achieved by different methods such as uniform sampling, Gaussian sampling, or random sampling. The bit vector is formed by comparing the intensity levels of an interest point and its predefined pair. If the first pixel's intensity is greater than its pair's intensity it is encoded as 1, or zero otherwise. In this study, the dimension of the bit vector is selected as either of 128, 256 and 512.

6) ORB: Oriented FAST and Rotated BRIEF (ORB) [36] is a combination of FAST keypoint detector and BRIEF descriptor extractor together with some enhancements. The idea is to sustain performance of SIFT for low-powered devices by using FAST keypoint detector, while using BRIEF descriptor extractor enhanced for rotation invariance. This is done by using a technique for measuring corner properties [44] which extracts orientation from corner intensity. For descriptor extraction, a more efficient method called steer BRIEF is introduced which creates a lookup table from angles quantized by 12 degrees. Whenever an angle from lookup table has a coherent result, rotation set regarding to the angle is selected as the dominant orientation.

ORB carries characteristics of both FAST keypoint detector and BRIEF descriptor extractor which means that computational load is extremely low.

7) BRISK: Binary Robust Invariant Scalable Keypoints (BRISK) [38] aims to find keypoints repeatedly in every viewpoint. Keypoints are fast computed by finding maxima in scale space with the help of FAST. Descriptor of BRISK is constructed in binary form. This is achieved by employing intensity comparison tests that is proposed in BRIEF [37]. However, as an advancement, the pattern is ensured to be equidistant on concentric circles. Gaussian smoothing is applied to overcome aliasing effects in the pattern. BRISK's detector and descriptor are both fast methods making it available in real-time and low-power platforms.

8) SimpleBlob: A conventional way of segmenting an image is binarizing it using a threshold and finding connected regions in the binarized image. The connected regions are called blobs. In order to binarize an image SimpleBlob assumes that the image is grayscale. Therefore, color information is erased from the image as the first step of this method. SimpleBlob applies several different binarization thresholds and corresponding binary images. Blobs are detected, and their centroids are used as keypoint locations. Blob's various properties such as average brightness, bounding box size, pixel area, eccentricity are combined and used as a descriptor [45].

We used the SimpleBlob blob detection algorithm as it is implemented as part of the OpenCV library.

9) FREAK: Fast Retina Keypoint (FREAK) [46] is a method for descriptor extraction inspired from the human visual system. FREAK uses circular sampling regions whose density diminishes through the center of attention. The design approximates the human visual system. FREAK creates a binary descriptor that is more suitable for computational purposes.

Keypoint detectors and descriptors such as BRIEF, BRISK, ORB, Star, SimpleBlob, FREAK and FAST are all suitable to be used in real-time applications because of having low computational load and complexity, whereas SURF and SIFT algorithms are suitable to be used in more demanding applications, employing complex transformations.

B. Transformation

Transformation refers to the geometric transformation of all pixels of an image. It is an essential operation for aligning two images when the frame capturing method is subject to translation or scaling effects. These effects might have been induced by the use of different camera orientations or lens distortions.

ADVISOR provides two alternatives for application of global transformations: *affine* transformation and *perspective* transformation.

Affine transformation covers translation, scaling, skewing, and rotation of an image. It has six degrees of freedom, meaning that we have six unknowns, and therefore need six equations to solve the system. Therefore, at least three pairs of 2D points selected in two different images to be aligned. Affine transformation always keeps parallel lines parallel. This approach is sufficient for aligning scaled, rotated and translated images. An upgrade to affine transformation is perspective transformation. Perspective transformation is an eight-degree-of-freedom system. We must use at least four pairs of 2D points selected in two different images to be aligned. Perspective transformation does not preserve parallelism, length and angle but preserves straight lines. In the end, one can only say that straight lines still stay straight. Perspective transformation is the most general transformation and covers all possible scenarios that an image can undergo. For most of the scenarios in real test environments, using affine transform based alignment is sufficient.

C. Image Matching

Image matching is mainly performed for finding same images under different transformations. It is used as a similarity measurement, which can be utilized for image retrieval, classification, registration, motion tracking, etc. In the following subsections, we discuss the types of image matching techniques we employ as part of ADVISOR.

1) Template Matching: Template matching is a type of shape matching approach that finds a predefined area from one image in another image. Hereby, the predefined area extracted from one image slides over the other image in one pixel strides. Histograms of the template and the corresponding underlying patch are calculated and matched during this process. Various histogram matching methods such as *cross-correlation*, *sum of absolute differences*, *sum of squared difference*, *correlation coefficient* and *coarse-to-fine* [47] [48] are employed. *Cross-correlation* has better performance when pixel intensity levels change in sub-regions of an image but its computational load is dramatically high especially for big window sizes [49]. On the other hand, *sum of absolute differences* method is faster but its performance is worse when pixel intensity levels change in sub-regions of an image. *Correlation coefficient* is more robust to pixel intensity changes [50]. OpenCV library offers three of these methods; *cross-correlation*, *sum of squared differences* and *correlation coefficient* and their normalized versions as a template matching method. We employed *correlation coefficient* method to find template matching score.

In ADVISOR, six different window sizes are used as the template region for histogram extraction.

2) Histogram Matching: A histogram contains the number of pixels of an image that suit to a specific criterion. In this study, we use pixel intensity histograms applied to all color channels separately. Representing images as histograms relaxes the image template representation in such a manner that locations of the pixels are not important anymore. The only important feature turns out to be the intensity of a pixel. Changing a pixel's location wouldn't change the histogram. Hence, we don't directly compare the images pixel by pixel but compare the histograms extracted from them. This provides robustness to matching images under translation transformation. Despite translational robustness, this method might not work well with color saturated images, which are also available in one of our data sets.

Several histogram matching methods are proposed. Some of these methods consider histograms as points in a high dimensional vector space and calculate distances between the points. Other methods apply probabilistic similarity metrics between histograms. *Euclidean distance* and *intersection* are the examples of distance based methods. Probabilistic methods are based on *probability density function* (PDF) and *Bhattacharyya distance* [51], *Kullback and Leibler divergence* [52], *Hellinger distance* [53], *Chi-Square* [54] and *Earth Mover's distance* [55]. OpenCV library offers *Correlation*, *Chi-Square*, *Intersection*, *Bhattacharyya*, *Hellinger* and *K-L* methods as a parameter of histogram matching function. In our case study, *Correlation* method is employed to calculate matching score of histograms.

3) Pixel Matching: Pixel matching is a straight-forward method, where each pixel of image pairs' are compared in each channel. This is done by summing up the square difference (SSD) of pixel values. The lower the value calculated with SSD, the better the matching is. However, image pairs should be tightly in the same planar surface as a precondition of obtaining meaningful results with this method. In order to use the result of pixel matching with other comparison algorithms, the obtained values are normalized between 0 and 1 to be compared with respect to a threshold value.

In case the compared images have to be aligned before *image matching*, then *keypoint detector*, *descriptor extractor* and *transformation* features must be selected to be employed by the test oracle. These are all used, if selected, before *image matching* and as such, they impact the accuracy of *image matching* and the overall process. In addition, other effects

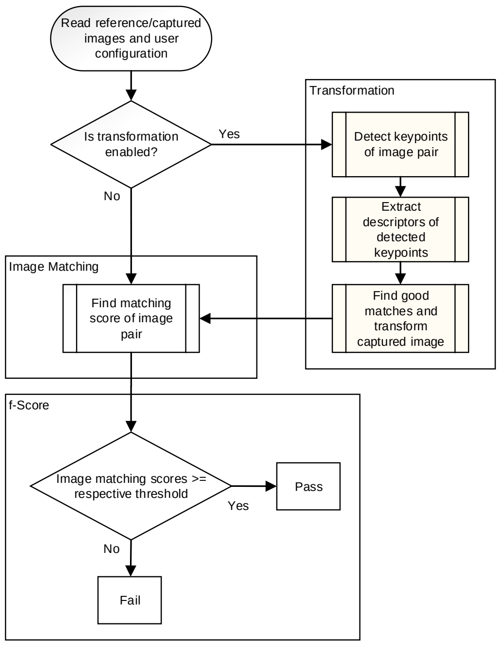

Read reference/captured images and user configuration
Is transformation enabled?
Yes
No
Transformation
Detect keypoints of image pair
Extract descriptors of detected keypoints
Find good matches and transform captured image
Image Matching
Find matching score of image pair
f-Score
Image matching scores >= respective threshold
Yes
Pass
No
Fail

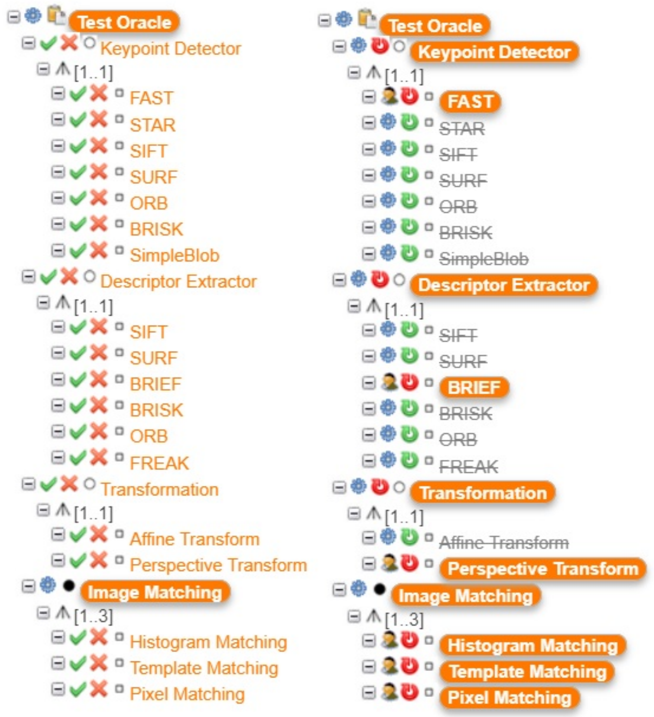
Fig. 2: The overall process.

such as illumination difference between grabbed and reference images might still negatively influence the results. Hence, it is expected that the results should be better if all the matching algorithms are employed together.

IV. Implementation

The overall process followed by ADVISOR is depicted in Figure 2. ADVISOR takes three inputs: a reference image, a captured image, and a configuration. ADVISOR compares the two images according to the parameters given in the configuration input, and gives a verdict of *pass* or *fail*. A pass verdict means that ADVISOR decided the captured image is sufficiently similar to the reference image to indicate no error in the SUT. Conversely, a fail verdict means that the captured image was found sufficiently different from the reference image to indicate an error in SUT.

ADVISOR integrates all the techniques discussed in Section III and makes them available to the user as features. The configuration input of ADVISOR is a CSV file that specifies which of these techniques are to be used for image comparison. The configuration file is prepared via a web-based graphical user interface, where the user can select or deselect features on the feature diagram of ADVISOR (see Figure 1). We implemented the feature diagram using an online feature modeling environment, SPLOT[4] (Software Product Lines Online Tools). One can use this environment to create a configuration file by importing the model and selecting or deselecting features. Consistency with respect to constraints is ensured by the tool during this process. The snapshot on the left-hand side of Figure 3 shows the SPLOT model we created. One can select or deselect features on this tree structure to define a configuration as shown on the right-hand side of Figure 3. The configuration in this example takes *FAST* as the keypoint detector, *BRIEF* as the descriptor extractor, and *Perspective Transform* as the transformation. It employs all the *image matching* techniques together. The final configuration can be exported as a CSV file to be fed into ADVISOR.



- Test Oracle
  - Keypoint Detector
    - [1..1]
      - FAST
      - STAR
      - SIFT
      - SURF
      - ORB
      - BRISK
      - SimpleBlob
  - Descriptor Extractor
    - [1..1]
      - SIFT
      - SURF
      - BRIEF
      - BRISK
      - ORB
      - FREAK
  - Transformation
    - [1..1]
      - Affine Transform
      - Perspective Transform
  - Image Matching
    - [1..3]
      - Histogram Matching
      - Template Matching
      - Pixel Matching

- Test Oracle
  - Keypoint Detector
    - [1..1]
      - FAST
      - ~~STAR~~
      - ~~SIFT~~
      - ~~SURF~~
      - ~~ORB~~
      - ~~BRISK~~
      - ~~SimpleBlob~~
  - Descriptor Extractor
    - [1..1]
      - ~~SIFT~~
      - ~~SURF~~
      - BRIEF
      - ~~BRISK~~
      - ~~ORB~~
      - ~~FREAK~~
  - Transformation
    - [1..1]
      - ~~Affine Transform~~
      - Perspective Transform
  - Image Matching
    - [1..3]
      - Histogram Matching
      - Template Matching
      - Pixel Matching

Fig. 3: Snapshot images taken from the online SPLOT tool that depict a part of the created feature diagram for ADVISOR and a configuration defined on this diagram.

We implemented ADVISOR in Python; it is available as an open-source framework[5]. In the following section, we present an empirical evaluation of our approach based on a benchmark dataset, where we compare the accuracy of several test oracle configuration instances derived from ADVISOR with those of previously introduced tools used as test oracles.

V. Evaluation

In this section, we experimentally evaluate ADVISOR. To this end, we compare several test oracle configuration instances derived from ADVISOR with previously introduced tools. Our first evaluation criterion is accuracy. Hereby, we are interested in both the overall accuracy and the accuracy for image pairs that are subject to particular types of variations. Our second concern is runtime performance. A test suite might process thousands of images and the efficiency of test oracle impacts the overall test duration. As such, runtime performance can play a critical role in the selection of a tool or its configuration. Therefore, we would like to compare the runtime performance of various ADVISOR configurations among themselves and with respect to state-of-the-art test

[4]http://www.splot-research.org

[5]https://github.com/ahmetesatgenc/Test-Oracle

oracles, SURFAndroid [9] and VISOR [10]. Accordingly, we ask the following research questions.

RQ1: How the overall accuracy of various test oracle implementations compare?

RQ2: How the accuracy of various test oracle implementations compare when the input images are subject to pixel shifting, scaling, and color saturation?

RQ3: How the runtime performance of ADVISOR configurations and state-of-the-art test oracle implementations compare?

In the rest of this section, we present the dataset, subject systems, and the metrics used for evaluation. Then we discuss the obtained results and threats to validity.

A. Experimental Setup and Dataset

We conducted our experiments on a regular laptop computer that has Intel I5-6200 2.30 GHz CPU and 8GB RAM. The computer has JRE 1.8 and 64-Bit OpenJDK installed. We used PyCharm version 2017.3.2 and OpenCv version V3.4.0.12.

We used a data set that is collected during the testing process of commercial Digital TV systems as previously introduced and used for evaluating test oracles in [11]. The set contains a total of 1000 image pairs (captured and reference images), each of which is manually labeled by the consensus of two test engineers in the company as belonging to one of the following classes:

- *failure*: the captured image differs from the reference image, and the pair indicates an error in the system. There are 456 image pairs in this category.
- *pixel shift*: the captured image differs from the reference image because of pixel shifting effects; there is no indication of an error in the system. There are 42 image pairs in this category.
- *scale*: the captured image differs from the reference image because of scaling effects; there is no indication of an error in the system. There are 359 image pairs in this category.
- *saturation*: the captured image differs from the reference image because of saturation effects; there is no indication of an error in the system. There are 143 image pairs in this category.

We should note that our dataset is collected during functional tests. Hereby, a failure is detected when the system crashes or provides wrong output, e.g., a particular list of channel names is expected to appear during channel search but it does not. Image variations such as pixel shifting, scaling and saturation occur due to the capturing method or due to the variations in the platform like screen size/resolution. Hence, these are not considered as failures. Also note that in principle it is possible for a captured image to be subject to multiple effects (e.g. pixel shifting plus scaling). When manually labeling the image pairs, the test engineers chose the most apparent effect in such cases.

B. Subject Systems

We include all the subject systems that were previously evaluated [11] on the dataset we are using. In addition, we evaluate two recently proposed test oracles, SURFAndroid [9] and VISOR [10]. Finally, we create and apply 6 configurations of ADVISOR, shown in Table I, built by combining the available features in various ways. These subject systems and configurations are explained in more detail in the following.

TABLE I: The set of selected ADVISOR configurations that are used for evaluation.

Config.	Keypoint Detector	Descriptor Extractor	Transform	Image Matching
$conf_1$	BRISK	BRISK	Perspective	(*)
$conf_2$	FAST	BRIEF	Perspective	(*)
$conf_3$	SURF	SURF	Perspective	(*)
$conf_4$	Star	ORB	Perspective	(*)
$conf_5$	SimpleBlob	FREAK	Affine	(*)
$conf_6$	SURF	SURF	Affine	(*)

(*) In all configurations "Histogram - Template - Pixel" features were selected for image matching.

1) SURFAndroid: SURFAndroid is based on the SURF [35] algorithm that primarily targets image transformation. The approach extracts the *keypoints* of an image and uses these keypoints as reference points for transforming the image. SURFAndroid utilizes this approach for another purpose: image comparison. It judges the similarity of two images based on how much their keypoints resemble each other. SURF can also be used as part of ADVISOR if it is selected during configuration. However, ADVISOR uses SURF as a keypoint detector and descriptor extractor only. The output of SURF is used for transformation to align a pair of images rather than image matching to compute similarity between them.

2) VISOR: VISOR, our previous work, was developed in the form of a pipeline of several image processing filters followed by an image comparison. The first set of filters include *background color removal*, *noise removal* and *bounding box detection*. They address major scaling and shifting variations in images. Finally, two more filters, namely *upsampling* and *max-pooling* are applied to remedy minor scaling and translation problems. Up-sampling scales images to the full image resolution for roughly satisfying the translation and scale invariance. Max-pooling, decomposes the input image into a grid of small rectangular blocks. Each block is replaced with the maximum pixel value that it contains. Although this causes some information loss in images, it provides further translation and scale invariance. Image comparison is performed by simply taking the pixel-by-pixel difference of the images received from the max-pooling filter. VISOR is specialized for handling translation and scale invariance with algorithms dedicated for this purpose. ADVISOR is a generalization of that approach; it does not tie itself to a specific algorithm or a pipeline. Rather, it is a generic and configurable framework that can be tuned for a particular application by selecting and combining various computer vision techniques as the building blocks.

3) ADVISOR Configurations: The set of selected ADVISOR configurations are listed in Table I, Hereby, we aimed to combine features that can maximize the accuracy. For this

TABLE II: Evaluation of a verdict according to image set categories.

Image Pair Category	Test Oracle Verdict	Evaluation
fail	Fail	TP
	Pass	FN
pixel shift, *scale*, *saturation*	Fail	FP
	Pass	TN

reason, we selected all the sub-features of *image matching* (“Histogram - Template - Pixel”) in all configurations.

The choice between the two alternatives for the *transform* feature (i.e., *affine* vs. *perspective*) does not supposed to have a significant impact on accuracy. This is because the dataset includes images captured directly from the system rather than those captured with an external camera. Hence, they can only be subject to 2D transformations rather than 3D. We included configurations that employ both alternatives to test them. Note that the third and sixth configurations are the same except this feature.

We varied the selection of *keypoint detector* and *descriptor extractor* sub-features in the tested configurations such that each choice is tested at least once. Only SIFT was omitted. SURF, which is employed by SURFAndroid [9] shares the same approach with SIFT. We also tested configurations, where the same sub-feature is used for both *keypoint detector* and *descriptor extractor* (See the first, third and sixth configurations).

C. Evaluation Criteria

Recall from Figure 2 that ADVISOR, when given an image pair and a configuration, gives a verdict in the form of either *pass* or *fail*. This verdict is considered a *true negative* (TN), *true positive* (TP), *false negative* (FN), or *false positive* (FP) as defined below:

TN: There is no error and the verdict is *pass*.
TP: There is an error and the verdict is *fail*.
FN: There is an error and the verdict is *pass*.
FP: There is no error and the verdict is *fail*.

Note that these terms are defined from the perspective of the test oracle. There exist two possible sources of errors. An error can take place in the test oracle or in the system under test. The task of the test oracle is to detect errors in the system under test. Hence, “positive” means that an error in the system is detected by the test oracle, leading to a *fail* verdict. If the system is indeed subject to an error, this verdict is correct and it is classified as TP. In this case, the oracle itself is not subject to an error. However, if an error is detected although the system under test is not subject to any error, then the verdict is classified as FP.

Precision and *recall* are calculated based on the number of TP, FN and FP verdicts as follows:

$$Precision = \frac{|TP|}{|TP|+|FP|} \tag{1}$$

$$Recall = \frac{|TP|}{|TP|+|FN|} \tag{2}$$

Test oracle verdict is determined based on a threshold value, which is calculated beforehand with an offline training step. Hereby, we used a part of the dataset for training and used the F-score measure to find the optimum threshold level for this part of the dataset. *F-score* is calculated based on precision and recall as shown below:

$$F_1 = 2 \times \frac{(precision \times recall)}{(precision + recall)} \tag{3}$$

We repeated the process in Figure 2 for varying threshold values used during image matching. The image pairs in the dataset are already labeled with respect to two major categories: *i)* those that are associated with failed runs, and *ii)* those that are associated with successful runs (test should pass although images are subject to pixel shifting, scaling or color saturation). Test oracle is executed during training with all possible threshold values between 0 and 1 with step size 0.1. TP, FP and FN cases are determined based on the oracle verdicts and the categories of the image pairs. Table II associates the four image pair categories with the possible test oracle verdicts. For instance, considering the first category in Table II (image pairs that are associated with failed runs), if the verdict is fail, then this would be considered as true positive. The verdict would be false negative if the oracle output was a pass for such an image pair. We took the threshold value that led to the maximum F-score.

We applied 10-fold cross validation to eliminate the bias in the selection of the training dataset for optimizing the threshold value. That is, we partitioned the dataset into 10 randomly-selected, equally-sized, disjoint segments. Then, an optimal threshold value was calculated 10 times. Each time, a different combination of 9 disjoint segments was used for calculating the threshold; the remaining disjoint segment was used for testing. Accuracy of the test oracle in each of the 10 tests is calculated based on the *accuracy* metric, which is defined as follows.

$$Accuracy = \frac{|TP|+|TN|}{|TP|+|TN|+|FP|+|FN|} \tag{4}$$

The overall accuracy of the test oracle is calculated as the average of accuracy measures obtained in 10 test runs.

We simply measured the time it takes to compare each image pair in the order of milliseconds to evaluate the runtime performance. We present and discuss the results in the following.

D. Results and Discussion

In the following, we discuss the obtained results regarding the evaluation of accuracy (*RQ1* and *RQ2*) and runtime performance (*RQ3*).

1) Accuracy: The overall results are listed in Table III. Recall that the *fail* dataset contains 456 image pairs that are all associated with failure cases. Hence, the verdict can be either TP or FN regarding these pairs (See Table II). Image pairs included in the other 3 datasets (*pixel shift*, *saturation*, *scale*) are all associated with successful executions although the captured images are subject to distortions. Hence, the verdict can be either FP or TN regarding these pairs. Table III lists the

TABLE III: The overall results for all the configurations and other tools used as test oracles (All the listed numbers represent rounded up percentage (%) values).

Tools / Configurations	**Pixel Shift**		**Saturation**		**Scale**		**Fail**		**Overall Accuracy**
	TN	**FP**	**TN**	**FP**	**TN**	**FP**	**TP**	**FN**	
Previously Evaluated Subject Systems									
PSNR	69.0	31.0	80.4	19.6	0.0	100.0	86.6	13.4	**53.9**
SSIM	100.0	0.0	95.8	4.2	98.6	1.4	37.2	62.8	**70.3**
DSSIM	100.0	0.0	100.0	0.0	100.0	0.0	0.0	100.0	**54.4**
PDIFF	100.0	0.0	76.2	23.8	97.4	2.6	37.9	62.1	**67.4**
AF	100.0	0.0	100.0	0.0	100.0	0.0	5.3	94.7	**56.8**
BT	100.0	0.0	100.0	0.0	100.0	0.0	1.6	98.4	**55.1**
IM	100.0	0.0	83.9	16.1	96.9	3.1	44.5	55.5	**71.3**
PIL	100.0	0.0	97.2	2.8	97.2	2.8	42.9	57.1	**72.1**
CV2-y1	69.0	31.0	76.2	23.8	88.0	12.0	74.5	25.5	**79.4**
CV2-y2	100.0	0.0	83.2	16.8	98.8	1.2	20.7	79.3	**61.0**
CV2-y3	100.0	0.0	100.0	0.0	88.8	11.2	30.1	69.9	**64.1**
CV2-y4	78.5	21.5	100.0	0.0	100.0	0.0	22.2	77.8	**63.6**
CV2-y5	100.0	0.0	100.0	0.0	100.0	0.0	22.6	77.4	**64.7**
CV2-y6	2.4	97.6	9.1	90.9	0.3	99.7	100.0	0.0	**47.1**
Recently Proposed Test Oracle Tools									
SURFAndroid	50.0	50.0	61.5	38.5	87.1	12.9	83.3	16.7	**80.2**
VISOR	95.2	4.8	92.3	7.7	95.8	4.2	93.9	6.1	**93.9**
ADVISOR									
conf$_1$	100.0	0.0	80.0	20.0	100.0	0.0	98.6	1.4	**96.5**
conf$_2$	100.0	0.0	81.4	18.6	100.0	0.0	98.6	1.4	**96.7**
conf$_3$	100.0	0.0	81.4	18.6	100.0	0.0	98.6	1.4	**96.7**
conf$_4$	100.0	0.0	80.7	19.3	100.0	0.0	98.6	1.4	**96.6**
conf$_5$	87.5	12.5	80.0	20.0	97.1	2.9	98.6	1.4	**95.0**
conf$_6$	100.0	0.0	87.1	12.9	97.1	2.9	97.3	2.7	**95.9**

percentage of FP, TN, TP and FN verdicts for all categories of image pairs separately. The overall accuracy computed based on these values is provided in the last column. Rows of the table are separated into 3 groups. The first group of rows correspond to the list of subject systems that were previously evaluated with the benchmark dataset [11]. The second set of rows lists the results for SURFAndroid [9] and VISOR [10]. The last group lists the results for the 6 configurations of ADVISOR as listed in Table I.

We can see that in general ADVISOR configurations consistently outperform all the other tools under study. In particular, *conf*$_2$ and *conf*$_3$ reached the highest overall accuracy of 96.7% among all these. One exception to this observation is related to the results obtained for the *saturation* dataset. The highest accuracy for this dataset is obtained with *conf*$_6$ among other configurations of ADVISOR. We can see that many of the previously developed tools (*SSIM*, *DSSIM*, *AF*, *BT*, *PIL*, some configurations of *CV2*, VISOR) perform (significantly) better on this dataset; however, their overall accuracy is lower. We can also see that the overall accuracy of *conf*$_6$ is lower than many other ADVISOR configurations. In particular, the ratio of FN cases is the highest among all the configurations, leading to the worst accuracy for the *fail* dataset. This shows that there is an inherent trade-off regarding the selection of transformation and matching techniques. Techniques that are effective for a particular case can turn out be less effective for other cases. Hence, one should select these techniques based on the test setup, the frequency of observed cases and trade-off decisions such as favoring recall over precision, i.e., minimizing the number of FN verdicts in expense of increasing the number of FP verdicts.

All the ADVISOR configurations except *conf*$_5$ reached 100% accuracy for the *pixel shift* dataset. This means that both *affine* and *perspective* transformations are able to bring the captured image to the same planar surface as the reference image successfully. On the other hand, configurations that use *perspective transform* lead to better results for the *scale* dataset. This means that *perspective transform* performs better than *affine transform* for images that are subject to scaling effects.

It was assumed that the highest accuracy would be obtained when all the image matching features are utilized together (See Section III). We created 3 more configurations based on *conf*$_1$ to evaluate the performance of these features separately and as such validate our assumption. Each of these configurations employs the same set of features with others; however, it uses only one of the image matching features to be different than the other two. Table IV shows the results for these configurations. We can see that template matching performs significantly better than the others in terms of accuracy. However, histogram matching and pixel matching can still be preferred for certain test setups due to other concerns such as runtime performance. Results listed in Table IV show that pixel matching perform the worst among all. This technique is more sensitive to pixel intensities compared to the other matching algorithms. As a result, it performs significantly worse for the

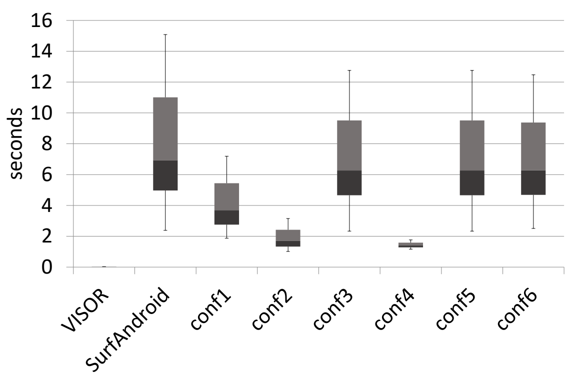


Fig. 4: The time it takes to process an image pair by the state-of-the-art test-oracles and evaluated ADVISOR configurations.

pixel shift dataset in particular.

2) Runtime Performance: Results regarding runtime performance are depicted as a box plot in Figure 4. The plot shows the time it takes to process an image pair in seconds. The state-of-the-art test-oracles and evaluated ADVISOR configurations are listed in the x axis.

We can see that VISOR is by far the fastest test oracle. It processes an image pair within 0.03 seconds on average. This result is expected since VISOR employs fast algorithms dedicated for particular variations. Hence, it is optimized for performance. Moreover, it is developed to utilize parallel processing and implemented in C++, instead of using an interpreted language [10]. Both SurfAndroid and ADVISOR are implemented in Python. Results show that SurfAndroid performs the worst both in terms of average image comparison time and in terms of variance in its runtime performance. The fourth configuration of ADVISOR (See Table I) is the second fastest after VISOR. It processes an image pair within 1.5 seconds on average.

The best accuracy values were obtained with the second and third configurations of ADVISOR. The second one seems to be better when runtime performance is considered. Overall, we see that differences among the tools and configurations are in the order of seconds per image comparison. Hence, differences in total testing time would be in the order of hours at most even if it involves the comparison of thousands of image pairs. This might be acceptable if the whole process is automated. In this case, one might consider maximizing the accuracy only and ignore runtime performance. Otherwise, there exists a trade-off between accuracy and runtime performance. However, we see that the most accurate tool or configuration is not necessarily the slowest one.

3) Attaining an Effective Configuration: We determined the initially selected 6 configurations by aiming at maximizing the overall accuracy. For this reason, all the sub-features of *image matching* (i.e. histogram, template, pixel) are included in all the 6 configurations. Later on, another experiment is performed with further 3 configurations derived from the first configuration. Hereby, all the feature selections for this configuration are kept the same but only one of the image matching features is selected at a time. Results (Table IV) showed that the accuracy is always lower than the one obtained with the original configuration, as expected.

In most of the configurations, we adopted *perspective transform* since it is superior to *affine transform*. We used *affine transform* only in the last two configurations. In particular, the last configuration is derived from the third configuration, where all the feature selections other than *transformation* are kept the same. We did not observe significant differences with respect to neither accuracy nor runtime performance. We expect that the difference in accuracy would be much higher if the images were captured by an external camera. The positioning of the camera can make it necessary to apply a perspective transformation to align images.

We varied the selection of *keypoint detector* and *descriptor extractor* sub-features in the tested configurations. Exhaustively testing every possible combination would take prohibitively long time. Hereby, we aimed at testing each alternative at least once. Only SIFT was omitted since it shares the same approach with SURF. We also tested configurations, where the the same sub-feature is used for both *keypoint detector* and *descriptor extractor*. Results showed that FAST keypoint detector (together with BRIEF descriptor) already leads to the best accuracy for images that are subject to scaling and pixel shifting. Hence, it makes sense to adopt this configuration since it leads to better runtime performance with respect to others. However, it is known that FAST may not be effective for noisy images. Images in our dataset were captured directly from the system and therefore they are not subject to noise that can impact the results.

In fact, the selection of features is not the only variation that affects the accuracy of test oracles. The utilized techniques have various parameters that can be tuned. In our experiments, we used the default values for almost all parameters for each of the utilized technique. These are the parameters that do not significantly impact the results. However, we investigated the effects of some parameters in more detail. The most important parameter is the *threshold* parameter that is used during *image matching* phase. This parameter is set automatically during the training phase. We also performed further experiments to determine the optimal value of the *window size* parameter used by the template matching algorithm. Table V lists the overall results for $conf_1$ when this parameter is varied. Results show that the accuracy increases as the window size increases. The maximum accuracy is achieved when it is set as 640x480. We have not listed the results for other values of this parameter since the accuracy was not improved further with increased window size values.

ADVISOR prevents some of the illegal configurations via constraints that are implemented as part of the feature diagram. It also adjusts the *threshold* parameter automatically during the training phase. Nevertheless, one has to manually configure the test oracle by (de)selecting features based on the test setup and the frequency of image effects taking place in the captured images. In principle, this configuration process can also be automated. The tool can automatically learn the type and frequency of image effects that cause differences between

TABLE IV: Overall results for *conf*$_1$ when only one of the image comparison algorithms is used.

	Pixel Shift		**Saturation**		**Scale**		**Fail**		**Overall**
Configurations	TN	FP	TP	FN	TN	FP	TN	FP	**Accuracy**
BRISK+Perspective+Histogram	100.0	0.0	67.1	32.9	90.5	9.5	59.3	40.7	**73.2**
BRISK+Perspective+Template	65.0	35.0	93.5	6.5	98.2	1.8	60.2	39.8	**78.7**
BRISK+Perspective+Pixel	7.5	92.5	42.1	57.9	68.2	31.8	75.1	24.9	**65.2**

TABLE V: Overall results for *conf*$_1$ when the window size parameter for template matching is varied.

	Pixel Shift		**Saturation**		**Scale**		**Fail**		**Overall**
Configurations	TN	FP	TP	FN	TN	FP	TN	FP	**Accuracy**
BRISK+Perspective+640x480	65.0	35.0	93.5	6.5	98.2	1.8	60.2	39.8	**78.7**
BRISK+Perspective+480x270	100.0	0.0	87.8	12.2	99.7	0.3	10.2	89.8	**56.9**
BRISK+Perspective+320x180	62.5	37.5	92.8	7.2	98.2	1.8	59.3	40.7	**78.1**
BRISK+Perspective+160x108	100.0	0.0	93.5	6.5	99.7	0.3	31.4	68.6	**67.4**
BRISK+Perspective+120x90	100.0	0.0	87.8	12.2	99.7	0.3	10.2	89.8	**56.9**
BRISK+Perspective+16x20	100.0	0.0	85.7	14.3	98.8	1.8	9.1	90.9	**55.8**

captured and reference images although the corresponding tests are not subject to a failure. This knowledge can steer the configuration process to minimize the number of false positives. On the other hand, the tool can also learn types of differences among the images that are associated with failed tests. This can be taken into account during image matching to minimize the number of false negatives. However, these approaches would require training with a much larger dataset to be effective. Also, this sample dataset should be representative of the data to be collected with the actual test setup in terms of the types of image effects and their frequencies of occurrence. Hence, automated configuration might not be cost-effective, considering the effort required for preparation of the manually-labelled dataset for training. This effort can be amortized only if the test setup remains unchanged for a long period of time. We leave the investigation of this trade-off as a possible future direction.

E. Threats to Validity

Our evaluation is subject to external validity threats [56] since it is based on a single benchmark dataset. This dataset was collected from a particular application domain.

Internal threats imposed by measurements are mitigated by using real image pairs collected during regular regression tests of real products in the industry. Our work did not involve any change of the dataset throughout the measurements. However, all the 1000 image pairs in this dataset were previously labeled manually. Hence, the accuracy and the bias of the labeling process is a concern that poses a threat to validity for our evaluation. This process was performed by two engineers in the company to mitigate this threat.

We compared our results with respect to previously made measurements on the same dataset to mitigate conclusion and construct validity threats. We also performed 10-fold cross validation on the dataset to mitigate these threats.

VI. Conclusions

We introduced ADVISOR, an adjustable framework for test oracle automation of visual output systems. The framework allows the use of a flexible combination and configuration of alternative techniques from the computer vision domain. We performed a domain analysis to review these techniques in terms of their pros and cons for applicability in various settings. ADVISOR can be configured to utilize a subset of these techniques that are tuned for a particular application context. We developed a feature model that defines commonalities and variations in test oracle implementations, the available techniques as well as constraints and conflicts among them. One can browse this model and define a test oracle instance configuration via a Web-based graphical user interface.

We evaluated several instances of our framework with respect to state-of-the-art tools. We used a benchmark dataset that includes image pairs collected during regular regression tests of real Digital TV systems. ADVISOR configurations significantly outperformed the other tools in terms of the overall accuracy achieved. Results also showed that there is an inherent trade-off regarding the configuration options. Techniques that are effective for a particular image effect like pixel shifting can turn out be less effective for another effect such as color saturation. ADVISOR enables one to select these techniques based on the test setup, the frequency of observed cases and trade-off decisions.

We evaluated state-of-the-art tools and ADVISOR configurations with respect to runtime performance as well. Overall, results showed that differences among these are in the order of seconds per image comparison. Therefore, differences in total testing time would be in the order of hours assuming that it involves the comparison of thousands of image pairs. This might be acceptable if the whole process is automated, in which case one might consider maximizing the accuracy only. Otherwise, one has to consider the trade-off between accuracy and runtime performance. However, we should note that the most accurate tool or configuration is not necessarily the slowest one.

The framework is still open to extensions and improvements. In this work, we covered the main features of an image-comparison based test oracle and their variations. In fact, some of the features have further sub-features and alternatives in terms of the algorithms used. For example, the metric currently used for *template matching* is fixed as *correlation coefficient* although other alternatives are available such as *cross-correlation* and *sum of square difference*. So, the feature tree can go deeper if we take these lower-level variations into account. ADVISOR is an open source framework. It can also be extended with new techniques in the future, in alignment with advances in the computer vision domain.

References

[1] S. Berner, R. Weber, and R. K. Keller, "Observations and lessons learned from automated testing," in *Proceedings of the 27th International Conference on Software Engineering*, 2005, pp. 571–579.

[2] D. Rafi, K. Moses, K. Petersen, and M. Mäntylä, "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey," in *Proceedings of the 7th International Workshop on Automation of Software Test*, 2012, pp. 36–42.

[3] G. Myers, T. Badgett, and C. Sandler, *The Art of Software Testing*, 3rd ed. Hoboken, NJ, USA: John Wiley and Sons Inc., 2012.

[4] W. Howden, "Theoretical and empirical studies of program testing," *IEEE Transactions on Software Engineering*, vol. 4, no. 4, pp. 293–298, 1978.

[5] E. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507 – 525, 2015.

[6] B. Meyer, "Eiffel: A language and environment for software engineering," *Journal of Systems and Software*, vol. 8, no. 3, pp. 199–246, 1988.

[7] Z. Q. Zhou, S. Xiang, and T. Y. Chen, "Metamorphic testing for software quality assessment: A study of search engines," *IEEE Transactions on Software Engineering*, vol. 42, no. 3, pp. 264–284, 2016.

[8] M. Delamaro, F. de Lourdes dos Santos Nunes, and R. A. P. de Oliveira, "Using concepts of content-based image retrieval to implement graphical testing oracles," *Software Testing, Verification and Reliability*, vol. 23, no. 3, pp. 171–198, 2013.

[9] Y. D. Lin, J. F. Rojas, E. T. H. Chu, and Y. C. Lai, "On the accuracy, efficiency, and reusability of automated test oracles for android devices," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 957–970, Oct 2014.

[10] M. Kirac, B. Aktemur, and H. Sozer, "VISOR: A fast image processing pipeline with scaling and translation invariance for test oracle automation of visual output systems," *Journal of Systems and Software*, vol. 136, pp. 266–277, 2018.

[11] O. Erdil, I. Can, and H. Sozer, "Evaluation of image comparison algorithms as test oracles," in *Proceedings of the 11th Turkish National Software Engineering Symposium*, 2017, pp. 101–113.

[12] I. Banerjee, B. Nguyen, V. Garousi, and A. Memon, "Graphical user interface (gui) testing: Systematic mapping and repository," *Information and Software Technology*, vol. 55, no. 10, pp. 1679 – 1694, 2013.

[13] E. Alégroth, R. Feldt, and L. Ryrholm, "Visual GUI testing in practice: challenges, problems and limitations," *Empirical Software Engineering*, vol. 20, no. 3, pp. 694–744, 2015.

[14] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, "Approaches and tools for automated end-to-end web testing," ser. Advances in Computers, A. Memon, Ed. Elsevier, 2016, vol. 101, pp. 193 – 237.

[15] S. Sprenkle, L. Pollock, H. Esquivel, B. Hazelwood, and S. Ecott, "Automated oracle comparators for testing web applications," in *Proceedings of the 18th IEEE International Symposium on Software Reliability*, 2007, pp. 117–126.

[16] S. Choudhary, M. Prasad, and A. Orso, "X-PERT: Accurate identification of cross-browser issues in web applications," in *Proceedings of the 2013 International Conference on Software Engineering*, 2013, pp. 702–711.

[17] J. Takahashi, "An automated oracle for verifying GUI objects," *SIGSOFT Software Engineering Notes*, vol. 26, no. 4, pp. 83–88, 2001.

[18] E. Selay, Z. Q. Zhou, and J. Zou, "Adaptive random testing for image comparison in regression web testing," in *Proceedings of the International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 2014, pp. 1–7.

[19] E. Selay, Z. Zhou, T. Chen, and F.-C. Kuo, "Adaptive random testing in detecting layout faults of web applications," *International Journal of Software Engineering and Knowledge Engineering*, vol. 28, no. 10, pp. 1399–1428, 2018.

[20] S. Mahajan and W. Halfond, "Finding HTML presentation failures using image comparison techniques," in *ACM/IEEE International Conference on Automated Software Engineering*, 2014, pp. 91–96.

[21] ——, "Detection and localization of html presentation failures using computer vision-based techniques," in *Proceedings of the 8th International Conference on Software Testing, Verification and Validation*, 2015, pp. 1–10.

[22] H. Yee, S. Pattanaik, and D. Greenberg, "Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments," *ACM Transactions on Graphics*, vol. 20, no. 1, pp. 39–65, 2001.

[23] T. Chang, T. Yeh, and R. Miller, "GUI testing using computer vision," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2010, pp. 1535–1544.

[24] D. Amalfitano, A. Fasolino, S. Scala, and P. Tramontana, "Towards automatic model-in-the-loop testing of electronic vehicle information centers," in *Proceedings of the 2014 International Workshop on Long-term Industrial Collaboration on Software Engineering*, 2014, pp. 9–12.

[25] R. Oliveira, A. Memon, V. Gil, F. Nunes, and M. Delamaro, "An extensible framework to implement test oracle for non-testable programs," in *Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering*, 2014, pp. 199–204.

[26] Q. Xie and A. Memon, "Designing and comparing automated test oracles for gui-based software applications," *ACM Transactions on Software Engineering and Methodology*, vol. 16, no. 1, pp. 1–36, 2007, Article No. 4.

[27] P. Clements and L. Northop, *Software Product Lines: Practices and Patterns*. Boston, MA: Addison-Wesley, 2002.

[28] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon University, Software Engineering Institute, Tech. Rep. CMU/SEI-90-TR-21, 2010.

[29] M. Hassaballah, A. Abdelmgeid, and H. Alshazly, *Image Features Detection, Description and Matching*. Cham: Springer International Publishing, 2016, pp. 11–45.

[30] R. Szeliski, *Computer Vision: Algorithms and Applications*. London, UK: Springer-Verlag, 2011.

[31] D. A. Lisin, M. A. Mattar, M. B. Blaschko, M. C. Benfiel, and E. G. Learned-Mille, "Combining local and global image features for object class recognition," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005.

[32] D. G. Viswanathan, "Features from accelerated segment test (fast)," 2009.

[33] M. Agrawal, K. Konolige, and M. R. Blas, "Censure: Center surround extremas for realtime feature detection and matching," in *European Conference on Computer Vision*. Springer, 2008, pp. 102–115.

[34] A. Vedaldi, "An implementation of sift detector and descriptor," *University of California at Los Angeles*, vol. 7, 2006.

[35] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[36] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE international conference on*. IEEE, 2011, pp. 2564–2571.

[37] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European conference on computer vision*. Springer, 2010, pp. 778–792.

[38] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2548–2555.

[39] A. Alahi, R. Ortiz, and P. Vandergheynst, "Freak: Fast retina keypoint," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. Ieee, 2012, pp. 510–517.

[40] J. Quinlan, "Induction of decision trees," *Machine Learning 1*, vol. 1(1), pp. 81 – 106, 1986.

[41] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[42] A. Witkin, "Scale-space filtering," *International Joint Conference on Artificial Intelligence*, pp. 1019–1022, 1983.

[43] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[44] P. L. Rosin, "Measuring corner properties," *Computer Vision and Image Understanding*, vol. 73(2), pp. 291 – 307, 1999.

[45] A.Kaehler and G.Bradski, *Learning OpenCV 3: Computer Vision in C++ with the OpenCV*. O'Reilly Media, Inc., 2016.

[46] A. Alahi, R. Ortiz, and P. Vandergheynst, "FREAK: Fast retina keypoint," in *Proceedings of the 25th IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 510–517.

[47] Y. M. Fouda, “A robust template matching algorithm basedon reducing dimensions,” *Journal of Signal and Information Processing*, vol. 06(02), pp. 109 – 122, 2015.

[48] A. Rosenfeld and G. Vanderbrug, “Coarse-fine template matching,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 7, pp. 104 – 107, 1977.

[49] T. D. and L. C., “Fast normalized cross correlation for defect detection,” *Pattern Recognition Letters*, vol. 24, pp. 2625 – 2631, 2003.

[50] K. S. Mahmood. A., “correlation-coefficient-based fast template matching through partial elimination,” *IEEE Transactions on Image Processing*, vol. 21 (4), pp. 2099 – 2108, 2012.

[51] T. Kailath, “The divergence and bhattacharyya distance measures in signal selection,” *IEEE Transactions on Communication Technolgy*, vol. COM-15 (1), pp. 52 – 62, 1967.

[52] S. Kullback and R. Leibler, “On information andsujciency,” *The Annals of Mathematical Statistics*, vol. 22 (1), pp. 79 – 86, 1951.

[53] V. H. E. Hellinger, “Neue begrndung der theorie quadratischer formen von unendlichvielen vernderlichen,” *Journal fr die reine und angewandte Mathematik*, vol. 136, pp. 210 – 271, 1909.

[54] K. Pearson, “On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling,” *Philosophical Magazine*, vol. 5, pp. 157 – 175, 1900.

[55] C. Y. Rubner and L. Guibas, “A metric for distributions with applications to image database,” in *Proceedings of the International Conference on Computer Vision*, 1998, pp. 59–66.

[56] C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer-Verlag, 2012.