

**ADVISOR: AN ADAPTIVE FRAMEWORK
FOR TEST ORACLE AUTOMATION
OF VISUAL OUTPUT SYSTEMS**

A Thesis

by

Ahmet Esat Genç

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Electrical and Electronics Engineering

Özyeğin University
January 2019

Copyright © 2019 by Ahmet Esat Genç

**ADVISOR: AN ADAPTIVE FRAMEWORK
FOR TEST ORACLE AUTOMATION
OF VISUAL OUTPUT SYSTEMS**

Approved by:

Assoc. Prof. Hasan Sözer (Advisor)
Department of Computer Engineering
Özyeğin University

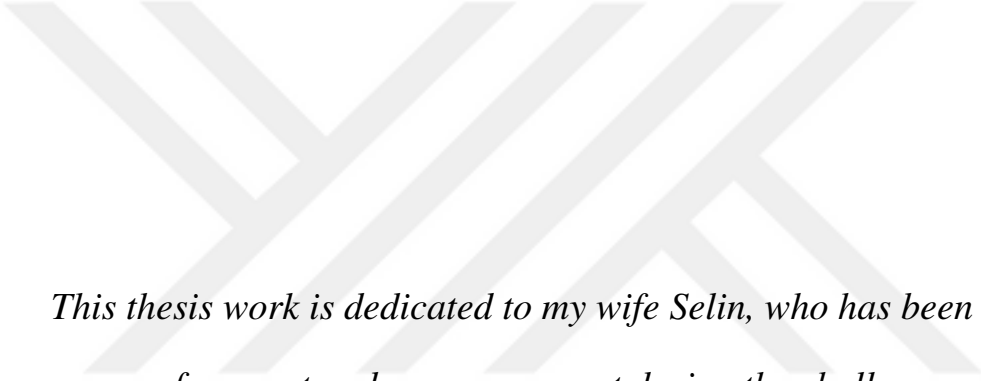
Asst. Prof. Furkan Kırac
Department of Computer Engineering
Özyeğin University

Asst. Prof. Ahmet Tekin
Department of Electrical and Electronics
Engineering
Özyeğin University

Assoc. Prof. Mehmet Aktaş
Department of Computer Engineering
Yıldız Technical University

Date Approved: 10 January 2019

Asst. Prof. Cenk Demiroğlu
Department of Electrical and Electronics
Engineering
Özyeğin University



This thesis work is dedicated to my wife Selin, who has been a constant source of support and encouragement during the challenges of graduate school and life. I am truly thankful for having her in my life.

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 observed output during test is compared with respect to a reference image. Hereby, the captured snapshot can be subject to variations due to, for instance, scaling, shifting, rotation, or color saturation. These variations lead to incorrect evaluations. Existing approaches in the literature employ a combination of techniques from the computer vision domain to address a specific set of variations. 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, introducing an unnecessary performance overhead. In this paper, we introduce ADVISOR, an adaptive 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 evaluated several instances of our framework with respect to state-of-the-art tools. We achieved up to 3% better overall accuracy based on a benchmark dataset collected during the tests of real Digital TV systems. We also observed that the accuracy of tools can differ for particular variations in the captured images.

ÖZETÇE

Test kahinleri, doğru ve yanlış sistem davranışını ayırt ederler. Görsel çıktı sistemleri için test kahinleri esas olarak test sırasında gözlenen çıktının anlık görüntüsü ile referans görüntünün karşılaştırıldığı, görüntü karşılaştırmayı içermektedir. Bu yaklaşımda anlık görüntü, yakalama metoduna bağlı olarak ölçekleme, kayma, dönme, veya renk doygunluğu gibi değişikliklere tabi olmuştur. Bu değişiklikler, yanlış değerlendirmelere neden olmaktadır. Bilgisayarlı görme alanında bu konulara değinen birçok teknik vardır. Literatürdeki mevcut yaklaşımlar, belirli bir varyasyon setini ele almak için bilgisayarlı görme alanındaki tekniklerin kombinasyonunu kullanır. Ancak bu tekniklerin bazıları, belirli bir varyasyonu ele almak için en etkili olanı olmayabilirken, bazı diğer teknikler, belirli bir varyasyonun yokluğunda gerekli olmayabilir ve bu nedenle gereksiz bir performans yükü oluşturur. Bu çalışmada, görsel çıktı sistemlerinin test kahini otomasyonu için uyarlamalı bir sistem olan ADVISOR'ı tanıtıyoruz. Sistem, bilgisayarlı görme alanından esnek bir kombinasyon ve alternatif tekniklerin yapılandırılmasına izin verir. Çalışmamızda, sistemimizin birkaç örneğini güncel araçlara karşı değerlendirdik. Gerçek Dijital TV sistemi testinde elde edilen kıyaslama veri setine dayalı 3% daha iyi genel doğruluğa ulaştık. Ayrıca, yakalanan görüntülerdeki belirli değişiklikler için araçların doğruluklarının farklılık gösterebileceğini de gözlemledik.

ACKNOWLEDGMENTS

Firstly, I would like to thank my advisor, Dr. Hasan Sözer for all his help and guidance that he has given me over the past year. Secondly, I would also like to thank Dr. Furkan Kır aç and Dr. Barıř Aktemur for providing me support during this period. Finally, I would like to thank my colleagues in Design Verification and Test Group at Vestel Electronics for sharing datasets with me and supporting my case study.

TABLE OF CONTENTS

DEDICATION	iii
ABSTRACT	iv
ÖZETÇE	v
ACKNOWLEDGMENTS	vi
LIST OF TABLES	viii
LIST OF FIGURES	ix
I INTRODUCTION	1
II BACKGROUND	4
2.1 Keypoint Detection and Descriptor Extraction	5
2.2 Descriptor Matching	11
2.3 Transformation	12
2.4 Image Matching	12
III RELATED WORK	15
IV TEST ORACLE IMPLEMENTATION	19
V INDUSTRIAL CASE STUDY	21
5.1 Dataset	22
5.2 Evaluation	23
VI RESULTS AND DISCUSSION	27
6.1 Threats to Validity	39
VII CONCLUSIONS AND FUTURE WORK	40
REFERENCES	41

LIST OF TABLES

1	The set of selected ADVISOR configurations when perspective transform is used as transformation method for evaluation.	24
2	The set of selected ADVISOR configurations when affine transform is used as transformation method for evaluation.	25
3	Evaluation of a verdict according to image set categories	26
4	Results of the configurations of [1] and SURFAndroid and VISOR(All the listed numbers represent rounded up percentage (%) values).	28
5	Results of the configurations when perspective transform is used as transformation method (All the listed numbers represent rounded up percentage (%) values).	30
6	Results of the configurations when affine transform is used as transformation method (All the listed numbers represent rounded up percentage (%) values).	31
7	Results of each image comparison algorithms for $conf_1, conf_{11}, conf_4, conf_{14}, conf_5, conf_{15}, conf_{10}$ and $conf_{20}$ according to F ₁ -Score.	32
8	Results of the 6 window sizes for for $conf_1, conf_5, conf_{10}, conf_4, conf_{14}$ according to F ₁ -Score.	33

LIST OF FIGURES

1	Feature diagram of ADVISOR.	6
2	The overall process.	19
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.	20
4	Matching scores of 640x480 window size of Template matching for $conf_1$.	34
5	Matching scores of 480x270 window size of Template matching for $conf_1$.	35
6	Matching scores of 16x20 window size of Template matching for $conf_1$. .	36
7	Keypoint against Good Matches graphs of <i>Saturation</i> dataset for $conf_1$, $conf_3$, $conf_3$ from left to right.	37
8	Keypoint against Good Matches graphs of <i>Saturation</i> dataset for $conf_5$, $conf_6$, $conf_7$ from left to right.	37
9	Keypoint against Good Matches graphs of <i>Saturation</i> dataset for $conf_8$, $conf_9$, $conf_{10}$ from left to right.	38
10	A sample image pair from the <i>Saturation</i> dataset for which the verdict was false positive	39
11	A sample image pair from the <i>Fail</i> dataset for which the verdict was false positive	39

CHAPTER I

INTRODUCTION

Automation of testing activities is a commonly preferred approach for reducing the costs of testing [2, 3], which can account for at least half of the overall development costs [4]. One of these activities is performed by a test oracle [5], which differentiates between the correct and incorrect system behavior. An analysis of the literature [6] reveals that test oracle automation has received significantly less attention compared to the automation of other testing activities. However, automating test oracle is an essential stepping-stone towards 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 [7]. 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 [8]. Accordingly, test oracle implementations for visual output systems tend to be fragile and they lead to many *false positives* [9], where an error is reported although an error does not exist.

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. 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 [10, 11] 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. Hence, a generically applicable, efficient and effective test oracle implementation for visual output systems must be adaptable 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 adaptive 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. We reviewed these techniques in terms of their pros and cons for applicability in various settings and 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 [1]. 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. 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 showed 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. We also observed that the accuracy of tools can differ for particular variations in the captured

images.

The remainder of this paper is organized as follows. In the following chapter, we summarize the related studies. In Chapter 2, 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 Chapter 4, we explain the implementation of ADVISOR. In Chapter 5.2, we present an empirical evaluation of our framework, where we compare a set of these implementations with respect to the state-of-the-art tools. Finally, in Chapter 7, we provide our conclusions.

CHAPTER II

BACKGROUND

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 [12] 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* [13]. 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.

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. During tests, 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, keypoints¹ of an image is extracted and utilized. Hence, an image is described as a combination of features, 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 features are detected by using *keypoint detector* which is also called a *feature detector*. Surroundings of the extracted feature 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 about keypoints and the related algorithms in further detail in the following subchapters.

2.1 Keypoint Detection and Descriptor Extraction

An image may be represented by local keypoints and global keypoints [14]. Local keypoints include corners, edges and lines that take place in images [15], whereas global

¹We 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.

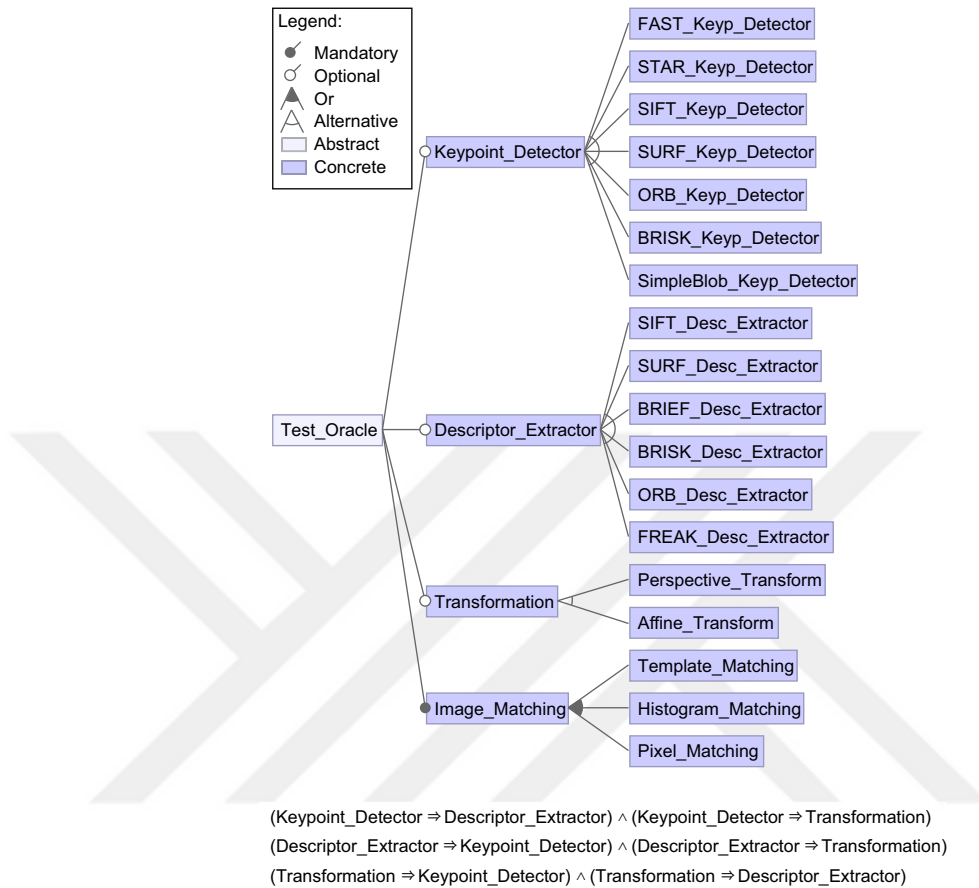


Figure 1: Feature diagram of ADVISOR.

keypoints include contour representations, shape descriptors, and texture [16]. ADVISOR involves many keypoint detector alternatives including FAST [17], Star [18], SIFT [19], SURF [20], ORB [21], BRIEF [22], BRISK [23] and SimpleBlob that find local keypoints of an image. Implementations are available as part of the OpenCV library². 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

²<http://opencv.org>

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 [22], BRISK [23], FREAK [24], ORB [21], SIFT [19] and SURF [20] as descriptor extractors.

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

2.1.1 FAST

Features from Accelerated Segment Test (FAST) [17] 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 [25] with an entropy optimizer is employed to determine if a pixel is a corner or not. The main strength of the FAST 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.

2.1.2 Star

Star keypoint detector is derived from the Center Surrounded Extrema (CenSurE) detector [18]. Aim of the Star keypoint detector is to find extrema in different scales and locations. In order to overcome computational load, bi-level filters multiplying intensity value of image pixels with 1 or -1 are used. Circles, octagons, hexagons and boxes are used as different bi-level filters. Performances of filters decrease according to their symmetry. Octagon filter has better performance while box filter has better computational speed. Non-maxima suppression method is used per different scale to find candidate local keypoints. Detected keypoints are further filtered by a method such as Harris corner detector. Remaining points are the local keypoints of Star keypoint detector.

2.1.3 SIFT

Scale Invariant Feature Transform (SIFT) [26] is used for both keypoint detection and descriptor extraction. Scale space theory [27] is employed for keypoint detection. This approach finds keypoints in different scales by utilizing a Gaussian pyramid. Algorithm works in different 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 [28] (HOG). The histogram bin with the maximum value is chosen as a normalization point. Histogram is rotated in such a manner that 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.

2.1.4 SURF

Speeded Up Robust Features (SURF) [20] 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 than SIFT but the descriptors are slightly less performant representations of the neighborhood when compared to SIFT.

2.1.5 BRIEF

Binary Robust Independent Elementary Features (BRIEF) [22] 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 hard-coded 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 128, 256 and 512.

2.1.6 ORB

Oriented FAST and Rotated BRIEF (ORB) [21] 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 [29] 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.

2.1.7 BRISK

Binary Robust Invariant Scalable Keypoints (BRISK) [23] 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 formed in binary form. This is achieved by employing intensity comparison tests that is proposed in BRIEF [22]. However, an advancement is proposed that the pattern is 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 scenarios.

2.1.8 SimpleBlob

An old 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 the image is grayscale. Therefore, color information is erased from the image as a first step. SimpleBlob method applies several different binarization thresholds and corresponding binary images. Blobs are detected, and their centroids are used as keypoint locations. Blob's different properties such as average brightness, bounding box size, pixel area, eccentricity are combined and used as a descriptor [30].

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

2.1.9 FREAK

Fast Retina Keypoint (FREAK) [31] 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 with underlying transformations of a more difficult nature.

2.2 *Descriptor Matching*

The most time consuming part of the image comparison system is descriptor matching. Once the keypoints are found with feature detectors, they have to be matched with the help of the feature detector. A few novel method are proposed such as FLANN matcher [32] based on randomized *k-d trees* and *hierarchical k-means trees*, Best-Bin First algorithm [33] based on *k-d trees* with modification of search ordering. Technics based on Brute-force matcher as a part of OpenCV library are proposed as in [34]. Brute-force matcher offers two paremeters, Euclidean or Hamming distance, for distance calculations and is combined with k-Nearest Neighbors(knn) to remove outliers. Due to take much time to calculate of

distances for each correspondings, Brute-force matcher is a slow technic but the result is precise. After matches between image features are found, good matches according to pre-defined distance are extracted in order to be used in transformation. This idea provides better transformation of captured image regarding reference image.

2.3 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 degrees-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.

2.4 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, registration, etc. In the following subsections, we discuss the types of image matching techniques we employ as part of ADVISOR.

2.4.1 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 [35] [36] 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 [37]. 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 [38]. 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 tried as the template region for histogram extraction.

2.4.2 Histogram Matching

A histogram contains the number 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. Only important feature becomes 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* [39], *Kullback and Leibler divergence* [40], *Hellinger distance* [41], *Chi-Square* [42] and *Earth Movers distance* [43]. 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.

2.4.3 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, obtained values are normalized between 0 and 1 to be compared with respect to a threshold value. Performance of image matching functions depend on the success of preliminary alignment transformation. However, other effects 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.

CHAPTER III

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 [44] 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 [45, 46] and they assume that GUI components (e.g., buttons, labels) or a document object model (e.g., as in HTML) for Web applications [47, 48] 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 [49]. This approach has become popular among researchers in the last few years in particular [9, 50, 51, 52, 11]. Many of these recent studies focus on Web applications [50, 51, 52]. For example, Selay et al. proposed the use of image comparisons to detect layout failures in these applications [50]. 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 [51, 52]. Their first study [51] employs pixel-to-pixel comparison and ignores variations among images as well. Later, they propose the use of *perceptual image differencing* [53] to compare images [52]. 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 with respect to other recent test oracle implementations [11, 1].

Sub-image searching was used in a visual testing tool called Sikuli [54], 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.

Image comparison has been used for automating test oracles in other application domains as well. For instance, such an approach was used in automotive industry [55]. 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

¹<http://pdiff.sourceforge.net>

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 [9, 56]. 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 re-usability of automated test oracles have been recently evaluated for android devices [10]. 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 5.2).

We previously implemented an automated test oracle [11] 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 [6] 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 [57].



CHAPTER IV

TEST ORACLE 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.

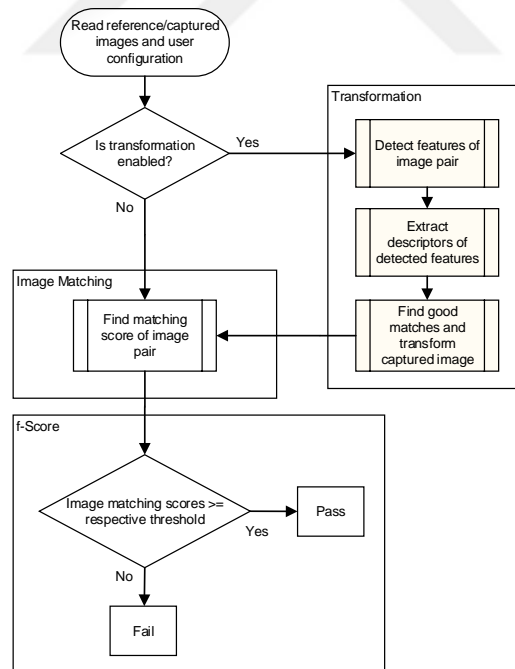


Figure 2: The overall process.

ADVISOR integrates all the techniques discussed in Chapter 2 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, SPLOT¹ (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.



Figure 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 framework².

¹<http://www.splot-research.org>

²<https://github.com/ahmetesatgenc/Test-Oracle>

CHAPTER V

INDUSTRIAL CASE STUDY

In Vestel R&D, different types of tests such as conformance, performance, functional are being performed by Design Verification and Test Group daily. Some of these tests can be adopted to Vestel automation system called as VesTa. Thanks to this, overnight tests or tests that have plenty of cases can be performed without wasting tester's time. Testers prepare test environment before starting the automation and check the verdict of the VesTa system via web-based test tracking page run on local computer. Hence, sources can be managed pretty easily during test a certification or mass production process.

VesTa, basically, sends remote control key codes to the DUT to follow instructions of pregenerated test steps to reach expected result of test cases. In doing so, it captures snapshots of predefined checkpoints in order to compare with reference images which are snapshot of predefined checkpoints of previous test at the same step. Comparison algorithm that VesTa using is peak signal-to-noise ratio (PSNR) during comparison of captured and reference images. The tester prepares test environment; prepares the device to be tested with correct panel size if the device is a TV and profile options which define the behaviour of the device during the test. The profile consist of two parts. One of which is software profile that includes for instance audio codec, DVB type support, country, language, customer, connectivity options. The other is the hardware profile that includes for instance source types and counts, mainboard and remote control type options. After the device is prepared for the test, image capturing board called Harran and is developed by Vestek, a subsidiary of Vestel Electronic is connected to the device via Ethernet input. Finally, the result of the test obtained by VesTa is controlled by the tester via web-based test tracking page to eliminate false negatives and false positives.

Even VesTa automation system is being used over years, it has some drawbacks which leads testers to spend more time than expected to check the results which motivate us to improve the system. One of the drawback of the VesTa is that it never checks whether captured and reference images are in the same planar surface or not. Vestel as a TV manufacturer produces TVs with panel size in range 24 to 75 inches. Due to the Harran board's nature, captured images are becoming subject of scale, shift, saturation or all of them when the panel size of the DUT is different comparing to the panel size of the DUT used to obtain reference images. The other drawback is to use PSNR algorithm is employed to validate the performance of the denoising algorithms [58] and to compute PSNR between images. However, because PSNR is not scale invariant, overall performance is getting lower when the captured image is subject to scale, shift or saturation.

Alternative techniques from the computer vision domain are employed as proposed in this work to improve the VesTA verdict. In order to measure the performance of the proposed test oracle, a dataset from real Digital TV test is used.

5.1 Dataset

We used a data set that is collected during the testing process of commercial Digital TV systems as previously introduced and used in [1]. The set contains a total of 1000 image pairs (captured and reference images), each of which is manually labeled 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.

Note that in principle it is possible for a reference image to be subject to multiple effects (e.g. pixel shifting plus scaling). When manually labeling the pairs, the human inspector chose the most apparent effect in such cases.

5.2 Evaluation

In this section, we experimentally evaluate the *accuracy* of ADVISOR. To this end, we compare several test oracle configuration instances derived from ADVISOR with previously introduced tools. 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?

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

5.2.1 Subject Systems

We include all the subject systems that were previously evaluated [1] on the dataset we are using. In addition, we evaluate two recently proposed test oracles, SURFAndroid [10] and VISOR [11]. Finally, we create and apply 20 configurations of ADVISOR, shown in Table-1 and Table-2, built by combining the available features in various ways.

Table 1: The set of selected ADVISOR configurations when perspective transform is used as transformation method for evaluation.

Config.	Feature Detector	Descriptor Extractor	Transform	Image Matching
conf ₁	BRISK	BRISK	Perspective	(*)
conf ₂	FAST	BRIEF	Perspective	(*)
conf ₃	FAST	FREAK	Perspective	(*)
conf ₄	ORB	ORB	Perspective	(*)
conf ₅	SIFT	SIFT	Perspective	(*)
conf ₆	SimpleBlob	BRIEF	Perspective	(*)
conf ₇	SimpleBlob	FREAK	Perspective	(*)
conf ₈	Star	BRIEF	Perspective	(*)
conf ₉	Star	FREAK	Perspective	(*)
conf ₁₀	SURF	SURF	Perspective	(*)

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

5.2.1.1 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 [59]:

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*.

Table 3 associates the four image pair categories with the possible test oracle verdicts.

In order to measure validity of the measurement, we employed *Precision* and *Recall* as defined in [60] as ratio of relevance of retrieved data and ratio of retrieved data that are relevant; respectively. Below are the equations of *Precision* and *Recall*;

$$Precision = \frac{|TP|}{|TP|+|FP|} \quad (1)$$

Table 2: The set of selected ADVISOR configurations when affine transform is used as transformation method for evaluation.

Config.	Feature Detector	Descriptor Extractor	Transform	Image Matching
conf ₁₁	BRISK	BRISK	Affine	(*)
conf ₁₂	FAST	BRIEF	Affine	(*)
conf ₁₃	FAST	FREAK	Affine	(*)
conf ₁₄	ORB	ORB	Affine	(*)
conf ₁₅	SIFT	SIFT	Affine	(*)
conf ₁₆	SimpleBlob	BRIEF	Affine	(*)
conf ₁₇	SimpleBlob	FREAK	Affine	(*)
conf ₁₈	Star	BRIEF	Affine	(*)
conf ₁₉	Star	FREAK	Affine	(*)
conf ₂₀	SURF	SURF	Affine	(*)

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

$$Recall = \frac{|TP|}{|TP|+|FN|} \quad (2)$$

With the help of *Precision* and *Recall*, *F-score*, harmonic mean of them, can be calculated as below;

$$F_{\beta} = (1 + \beta^2) \times \frac{(precision \times recall)}{(\beta^2 \times precision + recall)} \quad (3)$$

β is the weight of *F-score* and used to emphasize *Precision* when $\beta < 1$ and *Recall* when $\beta > 1$. Hereby, *true negative*, *true positive*, *false negative*, and *false positive* values for each threshold in range from 0 to 1 with 0.1 steps are calculated and *F-score* is extracted. As *F-score* equation indicates that *F-score* in it's highest value for a threshold set gives the optimum threshold value. Since, as long as the *true positive* and *true negative* have high values, *F-score* is getting higher. In ideal case, *true positive* and *true negative* converges to 1 and *false positive* and *false negative* converges to 0 which means that *F-score* will be zero. In our case study, we used $F_{0.5}$ -score to emphasizes *Precision* in order to minimize the *fp*, F_2 -score to emphasizes the *Recall* in order to minimize the *fn* and F_1 -score.

Table 3: Evaluation of a verdict according to image set categories

Image Pair Category	Test Oracle Verdict	Evaluation
<i>fail</i>	Fail	TP
	Pass	FN
<i>pixel shift, scale, saturation</i>	Fail	FP
	Pass	TN

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.

The metric to be judged the system effectiveness is *accuracy* which is calculated as a ratio of *true positive* and *true negative* to retrieved data as below:

$$Accuracy = \frac{|TP|+|TN|}{|TP|+|TN|+|FP|+|FN|} \quad (4)$$

CHAPTER VI

RESULTS AND DISCUSSION

Proposed test oracle system integrates all technics that OpenCV library includes and plenty of configurations can be generated by using SPLOT. However, some of the technics already provide both keypoint detection and descriptor extraction such as SURF, SIFT, BRISK and ORB. Additionally, some of them are based on already available keypoint detector or descriptor extractor technics. For instance, ORB is a combination of FAST and BRIEF with some enhancement. Due to the fact, configurations are generated in order to compare the performance of transformation, image matching and keypoint detector or descriptor extractor functions.

The overall results are listed in Table-4, Table-5 and Table-6. While listing all result for the configurations, template matching with 640x480 window size is used. 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 3). 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. Following tables list the number of *fp*, *tn*, *tp* and *fn* verdicts for all image pairs and the overall accuracy computed based on these values in the last column. Each row of Table-4 corresponds to a subject system as previously evaluated [1]. The last set of rows list the results for SURFAndroid [10] and VISOR [11] in Table-4. Rows of Table-5 and Table-6 lists the result of the configurations of ADVISOR as listed in Table-1 and Table-2. Additionally, *tp*, *tn*, *fp*, *fn* and *accuracy* values are added to tables according to F_1 -score, F_2 -score and $F_{0.5}$ -score. Recall that, $F_{0.5}$ -score is used to minimize the *fp* and F_2 -score is used to minimize the *fn* values.

Table 4: Results of the configurations of [1] and SURFAndroid and VISOR(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	
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
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

As can be seen from the Table-5 that $F_{0.5}$ -score successfully minimizes the fp values comparing to F_1 -score result. On the other hand, fn values are becoming higher as a side affect of this. In the same manner, F_2 -score successfully minimizes the fn values comparing to F_1 -score result but fp values are becoming higher. Thus, F_β -Score provides an option to choose between fp and fn . In real life applications, it is expected from test oracle system to minimize the fn values which provides to the tester to check only *fail* verdicts of the system. This avoids the tester to be in doubt whether the *pass* verdict is actually *pass* or not.

Table-5 also allows us to understand the performance of keypoint detectors and descriptor extractors. Recall that the image pairs included in the datasets (*Pixel Shift* and *Scale*) are subject to a kind of transformation. Hence, the result from these datasets point the performance of keypoint detectors and descriptor extractors regarding transformation method. Since, the image matching methods are not scale invariant and transformation which is computed by good matches should be successfully executed. When transformation method

is perspective transform and all image matching features of test oracle is selected in configurations, $conf_1$, $conf_2$, $conf_3$, $conf_5$ and $conf_{10}$ provides maximum accuracy for both *Pixel Shift* and *Scale* datasets. *Fail* dataset contains 456 image pairs that are all associated with failure cases. All configurations with perspective transform fail to find them as different images but have same performance on *Fail* data set. It's known that actual comparison result of the *Saturation* data set that contains 143 image pairs is successful. However, due to the different color intensities according to the related reference images, automatic results can be obtained as unsuccessful *fp*. As can be seen from the Table-5, all configurations have similar result for *Saturation* dataset while $conf_2$, $conf_8$ and $conf_{10}$ have better result with %81.4 percentage. In overall accuracy, $conf_{10}$ and $conf_2$ has the maximum accuracy with %96.7 percentage.

Table-6 presents the result of the same configuration of Table-1 except transformation method which is now affine transform. Performance of the configurations with affine transform on *Pixel shift* dataset is similar to configurations with perspective transform. The important difference is that the performance on the *Scale* dataset. As mentioned, keypoint detector and descriptor extractor functions are important to transform images into same planar surface. As can be seen from the Table-6 that the configurations with affine transform are not successful which means that perspective transformation is better than affine transformation when images are subject to scale. For other datasets, same judgment can be reached about F_β -score, *Fail* and *Saturation* dataset. In overall, $conf_{20}$ which has the same keypoint detector, descriptor extractor and image matching method with $conf_{10}$ has the maximum accuracy with %95.9 percentage.

It is expected that the best result is obtained when all comparison algorithms are used together as in discussed in Chapter 2. Table-7 shows the result of comparison algorithm combinations of $conf_1$, $conf_{11}$, $conf_4, conf_{14}$, $conf_5$, $conf_{15}$, $conf_{10}$ and $conf_{20}$. As can be seen from the Table-7 that the template matching algorithm gives the best results according to accuracies. For each dataset, template matching has better score rather than histogram

Table 5: Results of the configurations when perspective transform is used as transformation method (All the listed numbers represent rounded up percentage (%) values).

Configurations	F_{β} -Score	Pixel Shift		Saturation		Scale		Fail		Overall Accuracy
		TN	FP	TN	FP	TN	FP	TP	FN	
conf ₁	F ₁ -Score	100.0	0.0	80.0	20.0	100.0	0.0	98.6	1.4	96.5
	F ₂ -Score	45.0	55.0	62.8	37.2	92.3	7.7	100.0	0.0	89.6
	F _{0.5} -Score	100.0	0.0	93.5	6.5	100.0	0.0	92.9	7.1	95.8
conf ₂	F ₁ -Score	100.0	0.0	81.4	18.6	100.0	0.0	98.6	1.4	96.7
	F ₂ -Score	45.0	55.0	60.0	40.0	92.9	7.1	100.0	0.0	89.2
	F _{0.5} -Score	100.0	0.0	91.4	8.6	100.0	0.0	92.8	7.2	95.5
conf ₃	F ₁ -Score	100.0	0.0	80.0	20.0	100.0	0.0	98.6	1.4	96.5
	F ₂ -Score	47.5	52.5	62.8	37.2	92.3	7.7	100.0	0.0	89.7
	F _{0.5} -Score	100.0	0.0	95.7	4.3	100.0	0.0	92.6	7.4	96.0
conf ₄	F ₁ -Score	100.0	0.0	80.0	20.0	99.4	0.6	98.6	1.4	96.3
	F ₂ -Score	45.0	55.0	66.4	33.6	92.2	7.8	100.0	0.0	90.2
	F _{0.5} -Score	100.0	0.0	92.1	7.9	100.0	0.0	92.8	7.2	95.6
conf ₅	F ₁ -Score	100.0	0.0	80.7	19.3	100.0	0.0	98.6	1.4	96.6
	F ₂ -Score	45.0	55.0	60.0	40.0	92.2	7.8	100.0	0.0	89.2
	F _{0.5} -Score	100.0	0.0	92.8	7.2	100.0	0.0	92.8	7.2	95.7
conf ₆	F ₁ -Score	100.0	0.0	80.7	19.3	99.7	0.3	98.6	1.4	96.5
	F ₂ -Score	100.0	0.0	78.5	21.5	97.4	2.6	98.6	1.4	95.4
	F _{0.5} -Score	100.0	0.0	92.1	7.9	100.0	0.0	93.5	6.5	95.9
conf ₇	F ₁ -Score	97.5	2.5	80.7	19.3	99.4	0.6	98.6	1.4	96.3
	F ₂ -Score	50.0	50.0	62.1	37.9	92.2	7.8	100.0	0.0	89.7
	F _{0.5} -Score	97.5	2.5	90.7	9.3	100.0	0.0	93.1	6.9	95.4
conf ₈	F ₁ -Score	100.0	0.0	81.4	18.6	99.7	0.3	98.6	1.4	96.6
	F ₂ -Score	45.0	55.0	61.4	38.6	92.2	7.8	100.0	0.0	89.4
	F _{0.5} -Score	100.0	0.0	95.0	5.0	100.0	0.0	92.8	7.2	96.0
conf ₉	F ₁ -Score	100.0	0.0	78.5	21.5	99.4	0.6	98.6	1.4	96.1
	F ₂ -Score	45.0	55.0	62.8	37.2	92.2	7.8	100.0	0.0	89.6
	F _{0.5} -Score	100.0	0.0	96.4	3.6	100.0	0.0	93.1	6.9	96.3
conf ₁₀	F ₁ -Score	100.0	0.0	81.4	18.6	100.0	0.0	98.6	1.4	96.7
	F ₂ -Score	45.0	55.0	60.0	40.0	92.2	7.8	100.0	0.0	89.2
	F _{0.5} -Score	100.0	0.0	92.8	7.2	100.0	0.0	92.8	7.2	95.7

matching except *Pixel shift* dataset. Pixel matching is more sensitive than the other matching algorithms from pixel intensities. Since, it finds a lot more unsuccessful result in the datasets that have a pixel differences and images are subject to *scale* or *shift* but has better result on fail data set. In overall, template matching has more stable performance when

Table 6: Results of the configurations when affine transform is used as transformation method (All the listed numbers represent rounded up percentage (%) values).

Configurations	F_{β} -Score	Pixel Shift		Saturation		Scale		Fail		Overall Accuracy
		TN	FP	TN	FP	TN	FP	TP	FN	
conf ₁₁	F ₁ -Score	100.0	0.0	77.8	22.2	97.1	2.9	98.6	1.4	95.2
	F ₂ -Score	45.0	55.0	60.0	40.0	96.2	3.8	100.0	0.0	90.7
	F _{0.5} -Score	100.0	0.0	89.2	10.8	99.7	0.3	93.1	6.9	95.2
conf ₁₂	F ₁ -Score	100.0	0.0	78.5	21.5	96.8	3.2	98.6	1.4	95.3
	F ₂ -Score	97.5	2.5	77.8	22.2	96.8	3.2	98.6	1.4	95.0
	F _{0.5} -Score	100.0	0.0	90.7	9.3	99.7	0.3	92.8	7.2	95.3
conf ₁₃	F ₁ -Score	100.0	0.0	77.8	22.2	96.8	3.2	98.6	1.4	95.1
	F ₂ -Score	45.0	55.0	59.2	40.8	96.2	3.8	100.0	0.0	90.6
	F _{0.5} -Score	100.0	0.0	86.4	13.6	99.4	0.6	93.1	6.9	94.6
conf ₁₄	F ₁ -Score	100.0	0.0	80.7	19.3	97.1	2.9	98.6	1.4	95.6
	F ₂ -Score	45.0	55.0	65.7	34.3	96.2	3.8	100.0	0.0	91.5
	F _{0.5} -Score	100.0	0.0	88.5	11.5	98.8	1.2	93.5	6.5	95.0
conf ₁₅	F ₁ -Score	100.0	0.0	78.5	21.5	97.1	2.9	98.8	1.2	95.4
	F ₂ -Score	100.0	0.0	77.8	22.2	96.8	3.2	98.8	1.2	95.2
	F _{0.5} -Score	100.0	0.0	89.2	10.8	99.7	0.3	94.0	6.0	95.6
conf ₁₆	F ₁ -Score	100.0	0.0	79.2	20.8	97.1	2.9	98.6	1.4	95.4
	F ₂ -Score	47.5	52.5	59.2	40.8	96.2	3.8	100.0	0.0	90.7
	F _{0.5} -Score	100.0	0.0	85.0	15.0	98.8	1.2	94.0	6.0	94.6
conf ₁₇	F ₁ -Score	87.5	12.5	80.0	20.0	97.1	2.9	98.6	1.4	95.0
	F ₂ -Score	45.0	55.0	65.0	35.0	96.0	4.0	100.0	0.0	91.3
	F _{0.5} -Score	92.5	7.5	88.5	11.5	99.7	0.3	93.7	5.3	95.1
conf ₁₈	F ₁ -Score	100.0	0.0	80.0	20.0	97.1	2.9	98.6	1.4	95.5
	F ₂ -Score	100.0	0.0	77.8	22.2	96.8	3.2	98.6	1.4	95.1
	F _{0.5} -Score	100.0	0.0	90.7	9.3	99.7	0.3	92.8	7.2	95.3
conf ₁₉	F ₁ -Score	100.0	0.0	77.8	22.2	96.8	3.2	98.6	1.4	95.1
	F ₂ -Score	45.0	55.0	59.2	40.8	94.2	5.8	100.0	0.0	90.6
	F _{0.5} -Score	100.0	0.0	88.5	11.5	99.7	0.3	92.4	7.6	94.7
conf ₂₀	F ₁ -Score	100.0	0.0	87.1	12.9	97.1	2.9	97.3	2.7	95.9
	F ₂ -Score	100.0	0.0	77.8	22.2	96.8	3.2	98.6	1.4	95.1
	F _{0.5} -Score	100.0	0.0	90.7	9.3	99.4	0.6	94.2	5.8	95.8

perspective transform is applied to the datasets rather than affine transform.

As indicated in Chapter 2, template matching uses 6 different window sizes in this paper. In every table in experimental result discussion, 640x360 window size is used. This is because of this window size gives better result. However, when the window size

Table 7: Results of each image comparison algorithms for $conf_1$, $conf_{11}$, $conf_4, conf_{14}$, $conf_5$, $conf_{15}$, $conf_{10}$ and $conf_{20}$ according to F_1 -Score.

Configurations	Pixel Shift		Saturation		Scale		Fail		Overall Accuracy
	FP	TN	FN	TP	FP	TN	FP	TN	
Comparison algorithms for $conf_1$									
Histogram	100.0	0.0	67.1	32.9	90.5	9.5	59.3	40.7	73.2
Template	65.0	35.0	93.5	6.5	98.2	1.8	60.2	39.8	78.7
Pixel	7.5	92.5	42.1	57.9	68.2	31.8	75.1	24.9	65.2
Comparison algorithms for $conf_{11}$									
Histogram	100.0	0.0	62.8	37.2	72.8	27.2	69.1	30.9	70.8
Template	90.0	10.0	94.2	5.8	96.2	3.8	58.6	41.4	78.4
Pixel	0.0	100.0	17.1	82.9	0.0	100.0	100.0	0.0	48.3
Comparison algorithms for $conf_4$									
Histogram	100.0	0.0	67.1	32.9	90.5	9.5	59.7	40.3	73.4
Template	62.5	37.5	94.2	5.8	98.8	1.2	58.8	41.2	78.3
Pixel	22.5	77.5	53.5	46.5	86.2	13.8	63.3	36.7	68.4
Comparison algorithms for $conf_{14}$									
Histogram	100.0	0.0	67.8	32.2	91.4	8.6	60.2	39.8	74.0
Template	95.0	5.0	95.0	5.0	96.2	3.8	62.0	38.0	80.3
Pixel	0.0	100.0	27.5	72.5	0.0	100.0	100.0	0.0	48.8
Comparison algorithms for $conf_5$									
Histogram	100.0	0.0	67.1	32.9	90.5	9.5	59.1	40.9	73.1
Template	45.0	55.0	67.8	32.2	92.5	7.5	68.8	31.2	76.2
Pixel	7.5	92.5	42.1	57.9	71.1	28.9	74.6	25.4	66.0
Comparison algorithms for $conf_{15}$									
Histogram	100.0	0.0	62.8	37.2	72.8	27.2	70.8	29.2	71.6
Template	95.0	5.0	95.0	5.0	96.2	3.8	62.0	38.0	80.3
Pixel	0.0	100.0	16.4	83.6	0.0	100.0	100.0	0.0	48.2
Comparison algorithms for $conf_{10}$									
Histogram	100.0	0.0	67.1	32.9	90.5	9.5	59.7	40.3	73.4
Template	50.0	50.0	85.0	15.0	95.4	4.5	62.8	37.2	77.1
Pixel	7.5	92.5	42.1	57.9	69.1	30.9	74.6	25.4	65.3
Comparison algorithms for $conf_{20}$									
Histogram	100.0	0.0	62.8	37.2	72.8	27.2	69.1	30.9	70.8
Template	95.0	5.0	95.0	5.0	96.2	3.8	62.0	38.0	80.3
Pixel	0.0	100.0	17.1	82.9	0.0	100.0	100.0	0.0	48.3

gets smaller, the results are not getting any better. Table-8 presents the result of each window size for $conf_1, conf_5, conf_{10}, conf_4$ and $conf_{14}$. As can be seen from the table that the best result is obtained when window size 640x360 is used for only template matching result. The minimum window size fails to give correct verdict on *Fail* dataset while having same performance with other window sizes on *Pixel shift*, *Scale* and *Saturation* datasets. Additionally, the accuracy on *Fail* dataset decreases rapidly when the window sizes are becoming smaller. It can be understand that the small window sizes are not suitable to judge image pairs which are not similar. Surprisingly, 480x270 window size has not same

performance with for instance 320x180 or 640x480 window sizes.

Table 8: Results of the 6 window sizes for for $conf_1, conf_5, conf_{10}, conf_4, conf_{14}$ according to F_1 -Score.

Configurations	Pixel Shift		Saturation		Scale		Fail		Overall Accuracy
	FP	TN	FN	TP	FP	TN	FP	TN	
Window sizes for $conf_1$									
640x480	65.0	35.0	93.5	6.5	98.2	1.8	60.2	39.8	78.7
480x270	100.0	0.0	87.8	12.2	99.7	0.3	10.2	89.8	56.9
320x180	62.5	37.5	92.8	7.2	98.2	1.8	59.3	40.7	78.1
160x108	100.0	0.0	93.5	6.5	99.7	0.3	31.4	68.6	67.4
120x90	100.0	0.0	87.8	12.2	99.7	0.3	10.2	89.8	56.9
16x20	100.0	0.0	85.7	14.3	98.8	1.8	9.1	90.9	55.8
Window sizes for $conf_5$									
640x480	45.0	55.0	67.8	32.2	92.5	7.5	68.8	31.2	76.2
480x270	100.0	0.0	88.5	11.5	99.7	0.3	8.3	91.7	56.1
320x180	62.5	37.5	95.0	5.0	97.4	2.6	50.8	49.2	74.2
160x108	100.0	0.0	94.2	5.8	99.7	0.3	22.3	77.7	63.3
120x90	100.0	0.0	88.5	11.5	99.7	0.3	8.2	91.8	56.1
16x20	100.0	0.0	86.4	13.6	99.1	0.9	7.7	92.3	55.4
Window sizes for $conf_{10}$									
640x480	50.0	50.0	85.0	15.0	95.4	4.5	62.8	37.2	77.1
480x270	100.0	0.0	87.8	12.2	99.7	0.3	10.6	89.4	57.1
320x180	65.0	35.0	93.5	6.5	97.1	2.9	59.3	40.7	77.9
160x108	100.0	0.0	95.0	5.0	99.4	0.6	27.7	72.3	65.9
120x90	100.0	0.0	87.8	12.2	99.7	0.3	10.6	89.4	57.1
16x20	100.0	0.0	86.4	14.3	99.1	1.8	9.3	90.7	56.1
Window sizes for $conf_4$									
640x480	62.5	37.5	94.2	5.8	98.8	1.2	58.8	41.2	78.3
480x270	100.0	0.0	89.2	10.8	99.7	0.3	6.6	93.4	55.5
320x180	57.5	42.5	94.2	5.8	99.1	0.9	57.3	42.7	77.5
160x108	100.0	0.0	95.0	5.0	99.4	0.6	25.3	74.7	64.7
120x90	100.0	0.0	89.2	10.8	99.7	0.3	6.6	93.4	55.5
16x20	100.0	0.0	86.4	13.6	98.8	1.8	5.3	94.7	54.1
Window sizes for $conf_{14}$									
640x480	95.0	5.0	95.0	5.0	96.2	3.8	62.0	38.0	80.3
480x270	100.0	0.0	94.2	5.8	100.0	0.0	7.7	92.3	56.8
320x180	95.0	5.0	95.0	5.0	96.8	3.2	57.3	42.7	78.3
160x108	100.0	0.0	95.0	5.0	97.1	2.9	42.4	57.6	71.8
120x90	100.0	0.0	94.2	5.8	100.0	0.0	7.7	92.3	56.8
16x20	100.0	0.0	91.4	8.6	100.0	0.0	6.8	93.2	56.0

In order to understand the template matching accuracies, we examined the matching score of the template matching for 640x480, 480x270 and 16x20 window sizes with $conf_1$. Figure 4, Figure 5 and Figure 6 shows the matching score of template matching scores of

window sizes for each datasets.

It is expected that the matching scores should be maximum on *Saturation*, *Pixel shift* and *Saturation* datasets and minimum for *Fail* dataset. As can be seen from the Figure 4 that the matching score is quite high when 640x480 window size is used on *Saturation*, *Pixel shift* and *Saturation* datasets. Although matching scores are expected to be low on *Fail* dataset, 640x480 window size has high matching score. This leads the test oracle gives verdict *pass*.

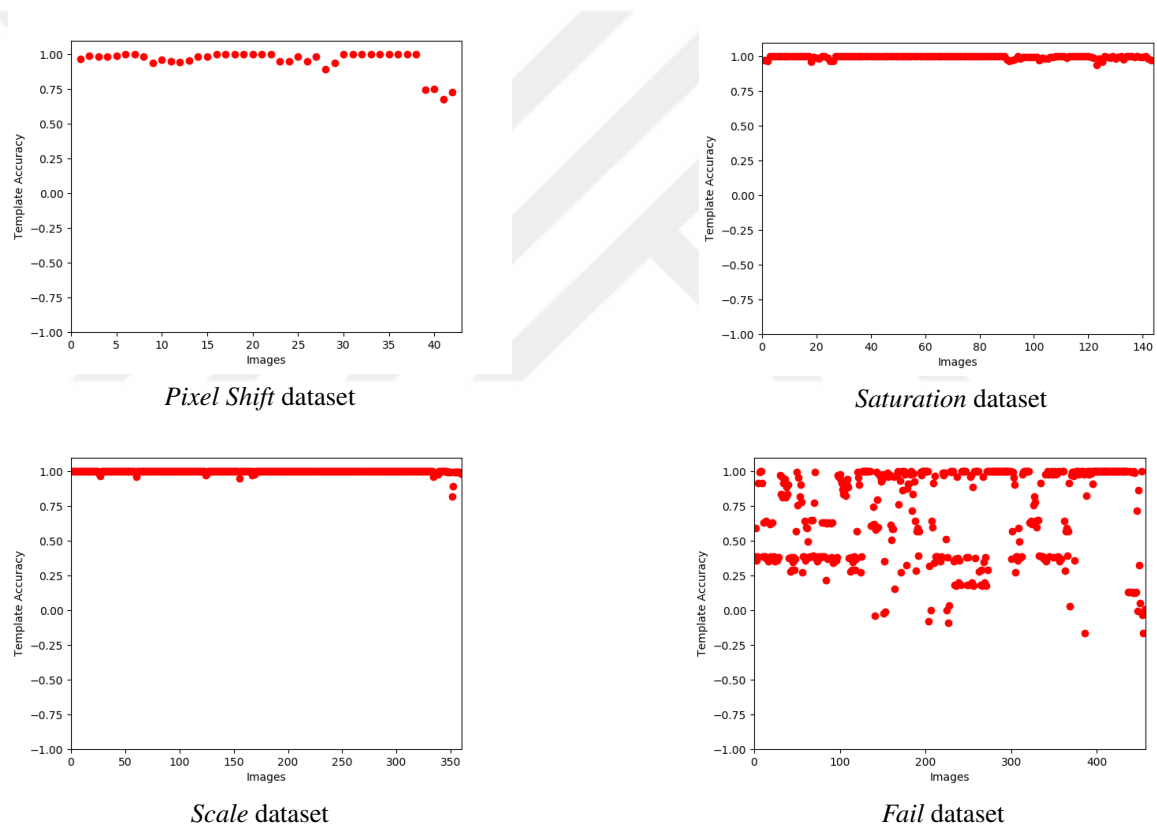


Figure 4: Matching scores of 640x480 window size of Template matching for $conf_1$

It can be understand the accuracy differences between *Saturation* and *Pixel shift* dataset of 640x480 and 480x270 window sizes from the Figure 4 and Figure 5. While 480x270 window size matching scores are better on *Pixel shift* dataset rather than 640x480 window size, it's matching scores for *Saturation* dataset are worse. This leads better *Pixel shift* accuracy but worse *Saturation* accuracy. As mentioned that it is expected the low matching

scores on *Fail* dataset in order to lead test oracle to give correct verdict. However, it's noteworthy that the matching scores of 480x270 window size on *Fail* dataset is quite high.

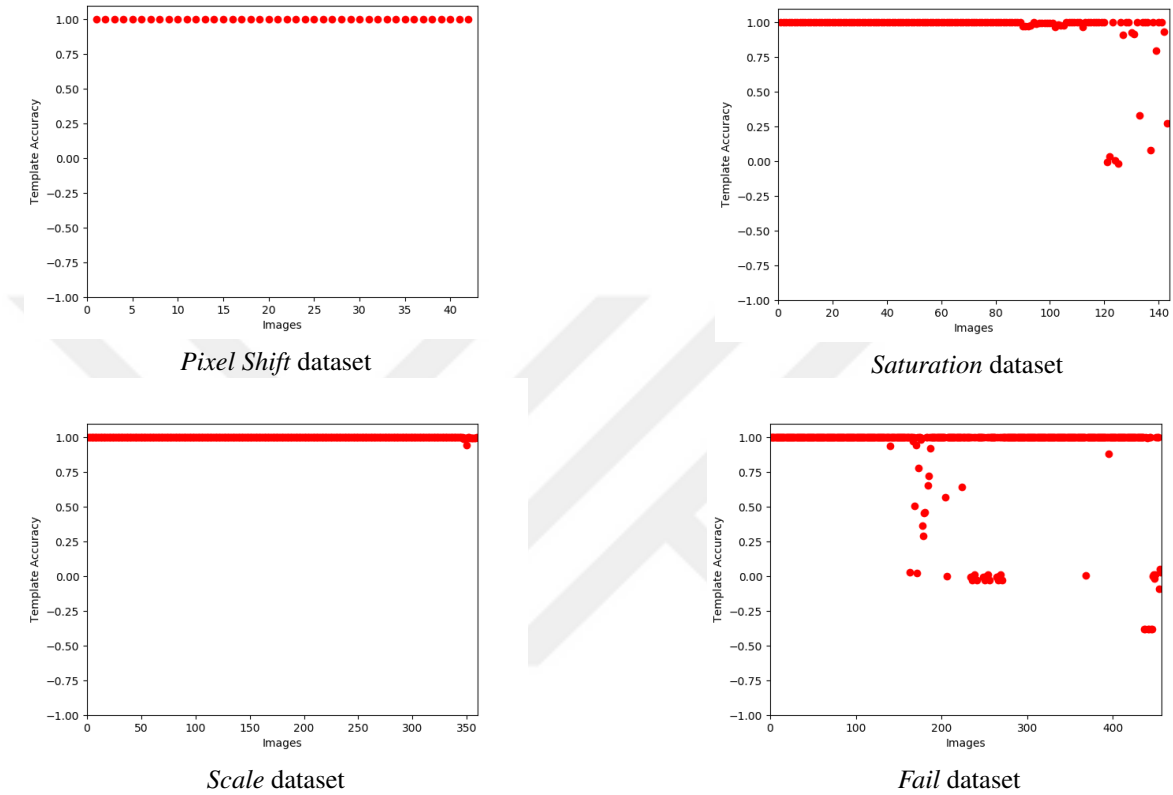


Figure 5: Matching scores of 480x270 window size of Template matching for $conf_1$

Accuracy differences on *Fail* dataset between window sizes are clearly the reason of performance of the dataset about matching scores. 16x20 window size has similar results with 480x270 window size which causes to decrease the accuracy on *Fail* dataset and also overall accuracy. It is conceivable that smaller window size provides better results, because the matching area is getting smaller and differences between images especially in *Fail* dataset can be dissociated easily. On the contrary, the results of the window sizes and matching scores graphs show that the bigger window sizes are better to compare images.

Performance of image matching algorithms are associated with how precise the image pairs are transformed. Overall results show that the performance of transformation is

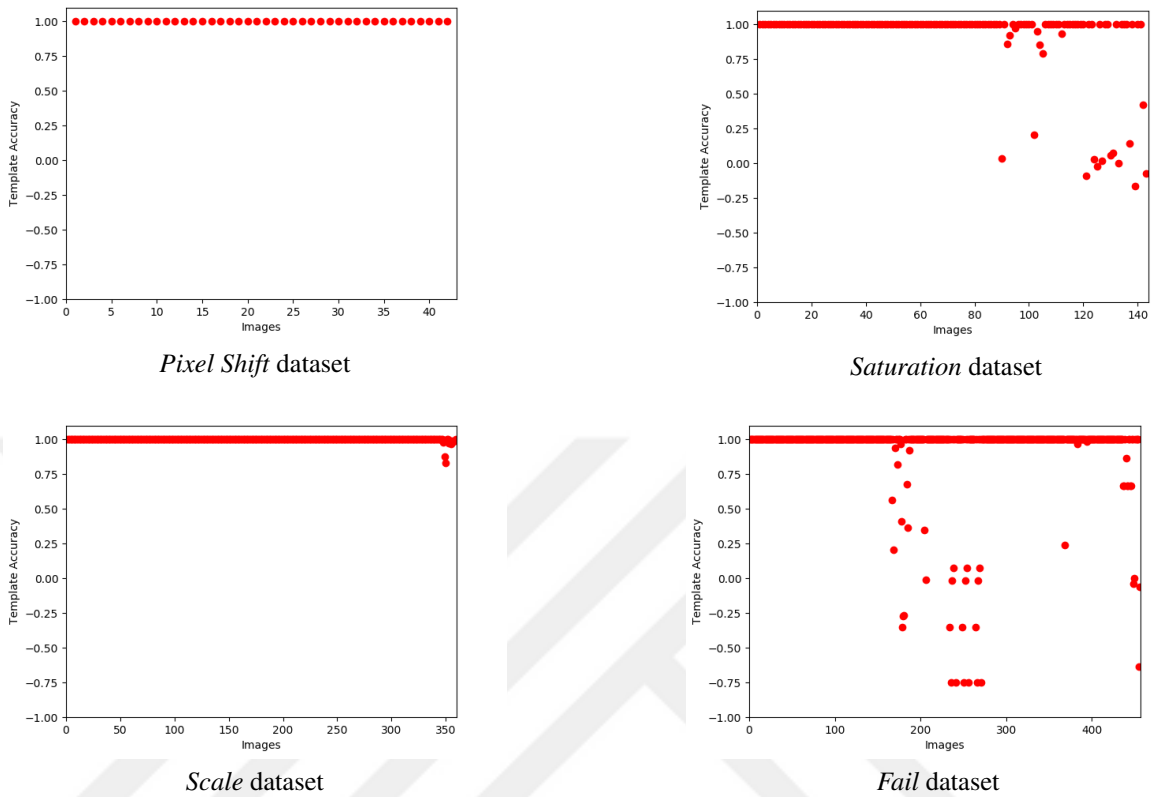


Figure 6: Matching scores of 16x20 window size of Template matching for $conf_1$

similar with different keypoint detectors, descriptor extractors and transformation methods. However, some of them have better result among them all. In order to compare their performance, found keypoints for each images and good matches are examined. Figure 7, Figure 8 and Figure 9 show the keypoints of captured image against good matches for *Saturation* dataset. $conf_4$ is not paid attention due to the ORB keypoint detector takes keypoint number as a parameter of the function. Because of the default parameters are used for every ORB keypoint detector and descriptor extractor, keypoint-goodmatches figures are meaningless to present. In ideal case, detected keypoints are also the good matches when keypoints of captured and reference images are matched. This provides a linear line that all points of the line have the same number of good matches and keypoints number. In addition to that the most number of computed good matches will benefit to transform image better. Since, the transform methods use good match points to compute transformation matrix.

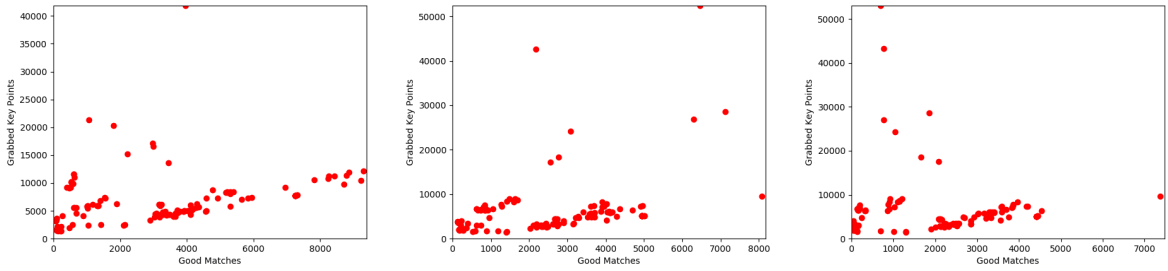


Figure 7: Keypoint against Good Matches graphs of *Saturation* dataset for $conf_1$, $conf_3$, $conf_3$ from left to right.

As shown in Figure 7, keypoints detected by keypoint detector of $conf_1$ are in range 0 to 40000. Calculated good matches are in range 0 to over 8000 and the distribution of the keypoints against good matches graph is similar to a linear line. On the other hand, keypoint detector of $conf_2$ and $conf_3$ which is FAST finds the keypoints in range 0 to 50000. However, good matches numbers are below against the found keypoints.

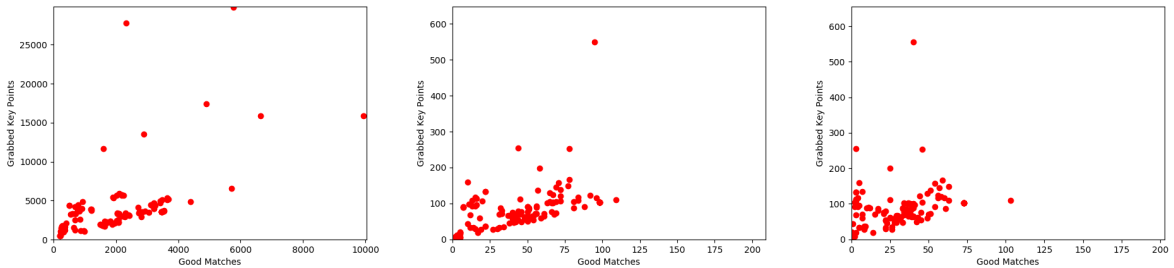


Figure 8: Keypoint against Good Matches graphs of *Saturation* dataset for $conf_5$, $conf_6$, $conf_7$ from left to right.

Although, SimpleBlob, keypoint detector of $conf_6$ and $conf_7$, finds a small number of keypoints from captured images, calculated good matches of $conf_6$ against the keypoints match according to Figure 8. This means that the descriptor extractor of $conf_6$ matches the keypoints precisely rather than the $conf_7$. On the other hand, performance of SIFT, which is employed for keypoint detection and descriptor extraction for $conf_5$, is better due to find plenty of keypoints and match them.

$conf_{10}$, employs *SURF* for keypoint detector and descriptor extractor, finds maximum number of over 17500 keypoints and matches these keypoints approximately in ideal case.

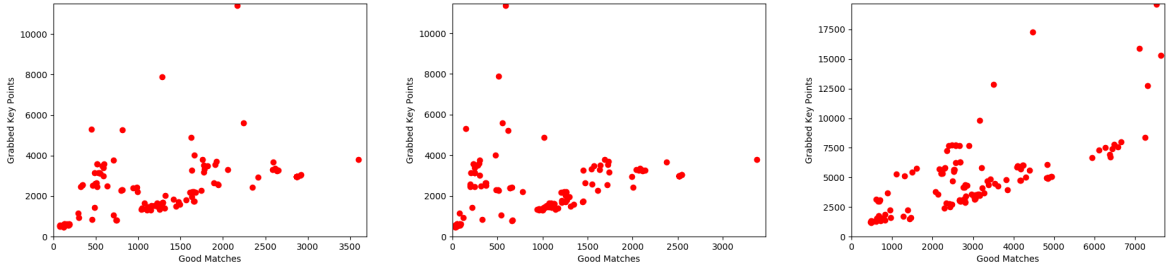


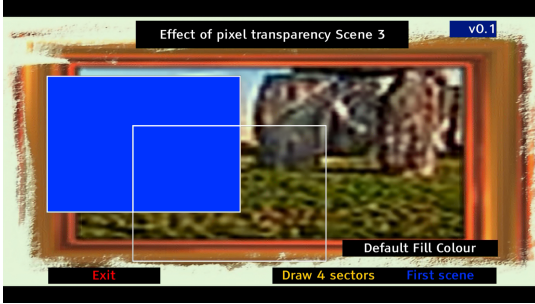
Figure 9: Keypoint against Good Matches graphs of *Saturation* dataset for $conf_8$, $conf_9$, $conf_{10}$ from left to right.

On the other hand, *Star* feature detector for both $conf_8$ and $conf_9$ finds maximum number of over 10000 keypoints but the matching performances of these configurations are not better than $conf_{10}$.

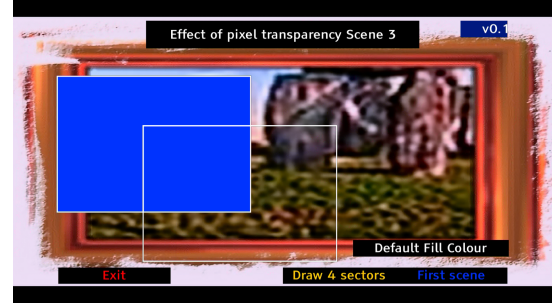
In overall, some of the feature detectors such as *SIFT* and *BRISK* find the most number of keypoints and some of the descriptor extractors match these keypoints of captured and reference images with better performance. It is expected that the $conf_1$, $conf_5$ and $conf_{10}$ have better performance on *Saturation* dataset. However, as can be remembered from the Table-5 that the $conf_2$, $conf_8$, and $conf_{10}$ have the maximum accuracy with %81.4 percentage. Even the result for the *Saturation* dataset is not directly related to keypoint detectors and descriptor extractors, their performances regarding to keypoints against good matches graphs are evaluated. The reason of varying accuracy results of *Saturation* dataset is due to not being saturation invariance of image matching methods. Figure 10 shows a sample image pair from *Saturation* dataset that ADVISOR gives false verdict.

In addition to that ADVISOR gives false verdict on *Fail* dataset with maximum %2.7 percentage. A sample image pair from *Fail* dataset that ADVISOR gives false verdict can be seen from Figure 11.

Hereby, image matching methods can reach the maximum possible accuracy for the Pixel Shift and Scale datasets. This is due to the effective handling of transformation deficiencies and highly accurate transformations.

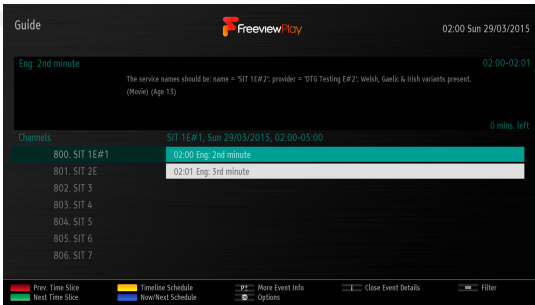


Captured Image

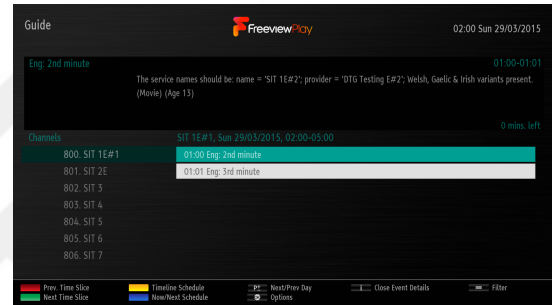


Reference Image

Figure 10: A sample image pair from the *Saturation* dataset for which the verdict was false positive



Captured Image



Reference Image

Figure 11: A sample image pair from the *Fail* dataset for which the verdict was false positive

6.1 Threats to Validity

Our evaluation is subject to external validity threats [61] 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 data set throughout the measurements.

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 treats.

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

We introduced ADVISOR, an adaptive 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.

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.

References

- [1] 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*, pp. 101–113, 2017.
- [2] 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*, pp. 571–579, 2005.
- [3] 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*, pp. 36–42, 2012.
- [4] G. Myers, T. Badgett, and C. Sandler, *The Art of Software Testing*. Hoboken, NJ, USA: John Wiley and Sons Inc., 3 ed., 2012.
- [5] W. Howden, “Theoretical and empirical studies of program testing,” *IEEE Transactions on Software Engineering*, vol. 4, no. 4, pp. 293–298, 1978.
- [6] 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.
- [7] B. Meyer, “Eiffel: A language and environment for software engineering,” *Journal of Systems and Software*, vol. 8, no. 3, pp. 199–246, 1988.
- [8] 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.
- [9] 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.
- [10] 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, pp. 957–970, Oct 2014.
- [11] 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.
- [12] P. Clements and L. Northop, *Software Product Lines: Practices and Patterns*. Boston, MA: Addison-Wesley, 2002.

- [13] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, “Feature-oriented domain analysis (FODA) feasibility study,” Tech. Rep. CMU/SEI-90-TR-21, Carnegie-Mellon University, Software Engineering Institute, 2010.
- [14] M. Hassaballah, A. Abdelmgeid, and H. Alshazly, *Image Features Detection, Description and Matching*, pp. 11–45. Cham: Springer International Publishing, 2016.
- [15] R. Szeliski, *Computer Vision: Algorithms and Applications*. London, UK: Springer-Verlag, 2011.
- [16] 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.
- [17] D. G. Viswanathan, “Features from accelerated segment test (fast),” 2009.
- [18] M. Agrawal, K. Konolige, and M. R. Blas, “Censure: Center surround extremas for realtime feature detection and matching,” in *European Conference on Computer Vision*, pp. 102–115, Springer, 2008.
- [19] A. Vedaldi, “An implementation of sift detector and descriptor,” *University of California at Los Angeles*, vol. 7, 2006.
- [20] 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.
- [21] 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*, pp. 2564–2571, IEEE, 2011.
- [22] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “Brief: Binary robust independent elementary features,” in *European conference on computer vision*, pp. 778–792, Springer, 2010.
- [23] S. Leutenegger, M. Chli, and R. Y. Siegwart, “Brisk: Binary robust invariant scalable keypoints,” in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2548–2555, IEEE, 2011.
- [24] A. Alahi, R. Ortiz, and P. Vandergheynst, “Freak: Fast retina keypoint,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 510–517, Ieee, 2012.
- [25] J. Quinlan, “Induction of decision trees,” *Machine Learning 1*, vol. 1(1), pp. 81 – 106, 1986.
- [26] D. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.
- [27] A. Witkin, “Scale-space filtering,” *International Joint Conference on Artificial Intelligence*, pp. 1019–1022, 1983.

- [28] 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, pp. 886–893, IEEE, 2005.
- [29] P. L. Rosin, “Measuring corner properties,” *Computer Vision and Image Understanding*, vol. 73(2), pp. 291 – 307, 1999.
- [30] A. Kaehler and G. Bradski, *Learning OpenCV 3: Computer Vision in C++ with the OpenCV*. O’Reilly Media, Inc., 2016.
- [31] A. Alahi, R. Ortiz, and P. Vanderghenst, “FREAK: Fast retina keypoint,” in *Proceedings of the 25th IEEE Conference on Computer Vision and Pattern Recognition*, pp. 510–517, 2012.
- [32] M. Muja and D. G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” in *Proceedings of the VISAPP International Conference on Computer Vision Theory and Applications*, pp. 331–340, 2009.
- [33] J. S. Beis and D. G. Lowe, “Shape indexing using approximate nearest-neighbour search in high-dimensional spaces,” in *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR 97)*, pp. 1000–, 1997.
- [34] A. Jakubovic and J. Velagic, “Image feature matching and object detection using brute-force matchers,” in *Proceedings of the 2018 International Symposium ELMAR*, pp. 83–86, 2018.
- [35] Y. M. Fouda, “A robust template matching algorithm based on reducing dimensions,” *Journal of Signal and Information Processing*, vol. 06(02), pp. 109 – 122, 2015.
- [36] A. Rosenfeld and G. Vanderbrug, “Coarse-fine template matching,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 7, pp. 104 – 107, 1977.
- [37] T. D. and L. C., “Fast normalized cross correlation for defect detection,” *Pattern Recognition Letters*, vol. 24, pp. 2625 – 2631, 2003.
- [38] 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.
- [39] T. Kailath, “The divergence and bhattacharyya distance measures in signal selection,” *IEEE Transactions on Communication Technolgy*, vol. COM-15 (1), pp. 52 – 62, 1967.
- [40] S. Kullback and R. Leibler, “On information and suiciency,” *The Annals of Mathematical Statistics*, vol. 22 (1), pp. 79 – 86, 1951.
- [41] V. H. E. Hellinger, “Neue begrundung der theorie quadratischer formen von unendlichvielen veranderlichen,” *Journal fur die reine und angewandte Mathematik*, vol. 136, pp. 210 – 271, 1909.

- [42] 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.
- [43] C. Y. Rubner and L. Guibas, “A metric for distributions with applications to image database,” in *Proceedings of the International Conference on Computer Vision*, pp. 59–66, 1998.
- [44] 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.
- [45] 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.
- [46] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, “Approaches and tools for automated end-to-end web testing,” vol. 101 of *Advances in Computers*, pp. 193 – 237, Elsevier, 2016.
- [47] 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*, pp. 117–126, 2007.
- [48] 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*, pp. 702–711, 2013.
- [49] J. Takahashi, “An automated oracle for verifying GUI objects,” *SIGSOFT Software Engineering Notes*, vol. 26, no. 4, pp. 83–88, 2001.
- [50] 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)*, pp. 1–7, 2014.
- [51] S. Mahajan and W. Halfond, “Finding HTML presentation failures using image comparison techniques,” in *ACM/IEEE International Conference on Automated Software Engineering*, pp. 91–96, 2014.
- [52] S. Mahajan and W. Halfond, “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*, pp. 1–10, 2015.
- [53] 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.

- [54] T. Chang, T. Yeh, and R. Miller, "GUI testing using computer vision," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1535–1544, 2010.
- [55] 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*, pp. 9–12, 2014.
- [56] 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*, pp. 199–204, 2014.
- [57] 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.
- [58] A. Kumar, N. Agarwal, J. Bhadviya, G. Mittal, , and G. Ramponi, "An efficient new edge preserving technique for removal of salt and pepper noise," in *Proceedings of the 8th International Symposium on Image and Signal Processing and Analysis*, 2013.
- [59] F. T., "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27(8), pp. 861 – 874, 2006.
- [60] C. D. Manning, P. Raghavan, and H. Schutze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.
- [61] C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer-Verlag, 2012.