A visual artifact detector for ITS.APE

An extension tool for ITS.APE to provide vision for the suite

Lab Report

by

Felix Rossmann
2471236

submitted to
Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik IV
Arbeitsgruppe für IT-Sicherheit

in degree course
Computer Science (M.Sc.)

First Supervisor:	Prof. Dr. Michael Meier University of Bonn
Second Supervisor:	Dr. Matthias Frank University of Bonn
Sponsor:	Arnold Sykosch, M.Sc. University of Bonn

Bonn, March 29, 2019

Abstract

An additional tool for the ITS.APE suite is presented to detect whether visual cues of deployed artifacts are visible on a Windows user's desktop. The task is done by image recognition in a screenshot using OpenCV, consisting of feature detection, feature matching and outlier rejection with the goal to find a transformation matrix between the features of the observed screenshot and reference images. The implemented tool is then evaluated with respect to quality of the matches and resource consumption, proving the successful implementation with about 90% successful matches within reasonable time and resources for the targeted execution environment.

Contents

1 Introduction

The *IT-Security Awareness Penetration Testing Environment* (ITS.APE)[Syk15] is a tool to analyse the awareness of a computer's user regarding IT-security. It measures the user's response to certain deployed *artifacts* which differ from the user's familiar environment. Most of them are similar to real-world attack scenarios or share certain principles with them.

There are multiple aspects to the user's response, one of them being the reaction speed. For measuring the reaction speed one must determine the point in time from which on it is possible for the user to interact with the artifact. As ITS.APE is a tool for the Microsoft Windows operating systems (which focus on a graphical user interface, a *GUI*), a user is able interact with an artifact as soon as he or she can visually perceive it on the screen, or in Windows terms: on the *desktop*.

The process of finding this point in time can not be done reliably by observing the currently running processes or even meta information about it. Only the visual information of the current desktop is useful in this case: An extra feature or tool was needed to perform an image detection algorithm on the current screen and decide whether the visual cues of the currently deployed artifact are present on the user's desktop.

The most important measure for this tool would be the *sensitivity*, describing the probability of successful detection of artifacts when they are actually present (*true positives*). This term will be called *detection rate* in the rest of this work and should ideally be at 100%. The counter-measure for this is the *specificity* which describes the rate of *true negatives*. A second, equally important measure is the resource consumption for execution, especially the execution time. As the detection rate is an obvious measure for such a tool, the execution time and resources are almost equally important to not disrupt the user's experience or distort the reaction analysis by ITS.APE. The limitation on the resources is given by the environment that ITS.APE is mainly used in: low-tier office hardware.

The work at hand describes the design considerations and implementation details of such a tool, called *Visual Artifact Detector* (*VAD*), and presents the results of an evaluation given the previously proposed measures. For this, chapter 2 outlines some related publications and the differences to our approach. In chapter 3, the software design and technical details of the VAD are explained. The evaluation setup for the given measures and the results can then be found in chapter 4, followed by a conclusion in chapter 5.

As ITS.APE is designed for computers running *Windows 7* in the 32-bit version, the VAD was built for this operating system specifically but can run on newer Windows versions as well. Written in C#, it uses Microsoft's *.NET framework* in version 4.7.2 [NET] and the *OpenCV*-wrapper *Emgu CV* in version 3.4.3 [EMGU].

2 Related Work

For evaluation of UI patterns Bakaev et al. [Bak+18] use image recognition functions to analyse a website's screenshot and produce a machine-readable representation of it. A grayscale version of the screenshot is used for rectangle detection, text sections are identified and recognized, before detecting special UI element types using a trained feature extractor. The results are then used to analyse composite structures and output a textual representation of the website. While the task seems similar to the one presented in this report, the proposed analysis has different goals and does not suit the requirements for the VAD: The full classification of the screenshot before identifying possible candidates for artifacts would imply a high demand on resources and is not a necessary step. Since those candidates would then be compared to reference images of artifacts, one can skip the classification steps and use an image feature detector on the screenshots directly.

Aradhye et al. [AMH05] present a way to classify image-based e-mails as spam using image analysis. Their approach features a way to analyse complex images without resorting to OCR: First, text regions are extracted by performing connectivity analysis on thresholded intensities of the grayscale image. With these regions identified, they categorize the image based on certain features which are then used to train *Support Vector Machines* (*SVMs*) to detect different mail classes. The key advantage of this way to classify images is that the spam images have distinct features to distinguish them from other images in mails. On one hand the ITS.APE artifact database is diverse and training SVMs for every artifact type would be demanding. On the other hand it is not easy to distinguish the more advanced ITS.APE artifacts from other applications by a rigidly defined feature set, because most of the artifacts ITS.APE employs try to imitate a usual application.

The goal of automated quality assurance lead Mozgovoy et al. [MP18] to the usage of image recognition on GUIs. They identified a problem for conventional GUI-testing suites for games with non-standard GUI and other elements. Their solution decides against exact bitmap matching, as the elements could be transformed, and instead match them approximately against a template. Since their approach is unable to find scaled matches, they first scale the reference to the observed image and then match them in total. Here, the main difference to the task at hand lies in the references to match against: In their case the matched output should have limited variation compared to the references so that they are able to match the whole screenshot onto the scaled reference. This is not applicable for the use cases in ITS.APE, as the artifacts are usually present in only a part of the whole screenshot. Since the rest of the screenshot is subject to high variation, building an adequate reference image collection for whole screens is highly infeasible.

Since none of the related work matches the goals for the Visual Artifact Detector, a new tool had to be implemented.

3 Methods

This chapter begins with a description of the general software design, including a brief requirement analysis and the resulting decisions regarding the implementation. The second part introduces the technical backgrounds and algorithms used, and especially explains the choice of the image recognition algorithms and their parameters.

3.1 Software Design

The client of ITS.APE that gathers the user's reaction to the deployed artifacts runs on a PCs as a Windows service. Instead of implementing the Visual Artifact Detector as an additional feature of the ITS.APE service it is implemented as an external tool. This is to keep a modular approach in which software parts can easily be exchanged, and secondly to be able to give the individual processes an independent priority for execution, thus guaranteeing enough resources for the execution of the ITS.APE service.

The main reason the VAD was designed to run on the same machines as the ITS.APE service instead of e.g. a server infrastructure with better hardware or other external hardware was the user's privacy: As ITS.APE aims to collect the data anonymously or pseudonymously, transactions of highly sensitive data (such as screenshots of the current desktop) would contradict this goal.

The information on artifact types to detect are provided by a *recipes repository* on the same computer containing reference images of the artifacts to be deployed. Furthermore, the VAD only has to look for visual cues of a single artifact type per run, as the ITS.APE service will call it with information on the currently deployed artifact type and the path to a screenshot to analyse.

3.1.1 Requirement analysis

To implement the VAD, the following main requirements were identified:

1. Robust and high detection rate for visual cues of artifact in screenshots of a Windows desktop.
2. Low runtime of at most few seconds on low-tier office hardware.

These requirements reflect the measures given for this task in the first chapter. The 1. requirement mainly has influence on the choice of image recognition algorithms, while the 2. contradicts the use of a very resource-heavy algorithm (which would provide better results, generally speaking) and has influence on the parameters of the used algorithm. This could lead to a trade-off between the 1. and the 2. requirement which is referred to in section 3.2.

Further analysis of the nature of ITS.APE and refinement of the main requirements lead to the following secondary requirements:

3. No usage of parallel computing on the graphics card (e.g. with the CUDA Toolkit[CUDA]), CPU only.
4. Windows 7 console program (32-bit).

The 3. requirement is a specification of the 2., as low-tier office computers usually don't include a CUDA-enabled graphics card or a non-integrated graphics card at all. This requirement results in better versatility of the tool, but comes at cost of a higher runtime.

Given the deployment area for ITS.APE in the present and near future, the 4. requirement is due to the environment ITS.APE is mainly run in. This is also important for the nature of the visual cues of artifacts which mainly are those of standard Windows GUI objects.

3.1.2 Implementation decisions

It was initially decided to use an existing library for image recognition rather than implementing the algorithms anew. The choice for an image recognition library fell to OpenCV, as it is licensed as *Open Source* under the BSD license and arguably the most maintained and popular of such libraries with almost 20 million downloads [CVDL19]. Other libraries for similar tasks exist, but they either focus more on machine learning like *Google's Tensorflow* [GTF], are web-based APIs (mostly non-free) like *Google's Vision API* [GVAPI] or *Amazon Rekognition* [ARK], or are not well maintained, such as *VLFeat* [VLF].

Because the runtime performance is so important for this tool, it was not suitable to implement the program using an interpreted programming language. The usage of OpenCV limits the available compiled languages to C, C++ or Java, and using the wrapper Emgu CV also allows to use C#.

The advantage of C# is the possibility to use Microsoft's *.NET Framework* [NET]. It allows to efficiently implement Windows 7 programs while the latest *redistributable packages* of .NET needed for executing .NET programs are available per default (given the system is up-to-date). Furthermore, the speed differences between the execution of .NET's intermediate language in their *Just-in-Time* compiler are nowadays insignificant to programs written in e.g. native C++. Therefore the VAD is written in C# using .NET 4.7.2 and OpenCV 3.4.3 via the Emgu CV wrapper.

During evaluation it was noticed that the used post-processing function by Emgu CV lead to erroneous results and it was decided to code an own implementation of it as a replacement (see subsection 3.2.3 for details).

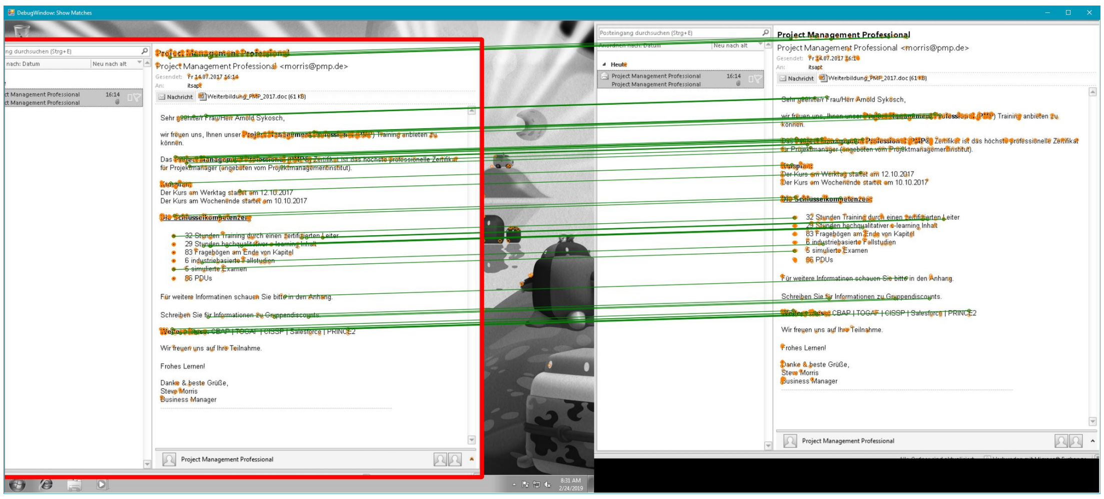
Furthermore, a Windows installer setup was composed to allow easy deployment of the VAD after compilation.

3.2 Technical backgrounds

The process of image recognition in the VAD expects two input images per run, one model image and one observed image, and tries to recognize the model in the observed image. The model image is a reference image of the artifact type to detect and the observed image is a screenshot provided to the program at execution time. Since the given artifact type may have multiple reference images for several characteristics, this process might be performed as many times as there are reference images, but exits as soon as a match between a reference and the screenshot was found. This is especially important for the performance, as the order of the artifact's reference images are directly affecting the runtime (see chapter 4).

The goal of the recognition process is to find a valid transformation matrix to the model's features for the largest possible subset of the observed image's features. If such a matrix can not be found for a subset of certain size, it is decided that no match exists. There are three major steps involved in the image recognition process: *Feature detection and description*, *feature matching* and *outlier rejection*, while the last step includes finding a transformation matrix.

The first step yields a set of features extracted from the observed image. This is also done for all of the model images, but their feature set could be fetched from a cache if extracted in a previous run, which effects the performance (see chapter 4). The features are represented as feature descriptors which are then matched with the model's descriptors, resulting in a set of candidates for features found in both images. These are then post-processed during outlier rejection to find a transformation matrix from one feature subset to the other while matching the most candidates. If there is such a matrix fitting to a certain percentage of the candidates (see subsection 3.2.3) the VAD will return the successful detection of the artifact (return value 0). Otherwise, it will return that no match was found (return value 1).



Project Management Professional
Project Management Professional <morris@pmp.de>
Gesendet: Fr 14.07.2017 16:14
An: itsapt
Nachricht | Weiterbildung_PMP_2017.doc (61 KB)

Sehr geehrte/r Frau/Herr Arnold Sykosch,

wir freuen uns, Ihnen unser Project Management Professional (PMP) Training anbieten zu können.

Das Project Management Professional (PMP®) Zertifikat ist das höchste professionelle Zertifikat für Projektmanager (angeboten vom Projektmanagementinstitut).

Kursplan:
Der Kurs am Werktag startet am 12.10.2017
Der Kurs am Wochenende startet am 10.10.2017

Die Schlüsselkompetenzen:

- 32 Stunden Training durch einen zertifizierten Leiter
- 29 Stunden hochqualitativer e-learning Inhalt
- 83 Fragebögen am Ende von Kapitel
- 6 industriebasierte Fallstudien
- 6 simulierte Examen
- 86 PDUs

Für weitere Informatinen schauen Sie bitte in den Anhang.

Schreiben Sie für Informationen zu Gruppendiscounts.

Weitere Kurse: CBAP | TOGAF | CISSP | Salesforce | PRINCE2

Wir freuen uns auf Ihre Teilnahme.

Frohes Lernen!

Danke & beste Grüße,
Steve Morris
Business Manager

Project Management Professional

Figure 1: *Example debug output of the VAD when finding a match. On the left side is the observed image, on the right the model. Yellow circles show match candidates, green lines connect actual matched feature points. A red rectangle is drawn around the recognized area.*

In figure 1 an example of debug output for a successful match can be found with the observed screenshot drawn in the left half and the reference screenshot in the right. There are more match candidates marked than are actually used as match, which shows the results of the outlier rejection step. The transformation matrix has been used to draw the rectangle around the area where the artifact has been successfully recognized. Since the artifact's window partly lays outside of the screenshots boundaries, the red rectangle appears cut off as well.

The retrieval of the transformation matrix can take advantage of the special properties of the deployed artifact's being mostly standard GUI elements: There are only few possible transformations for Windows GUI objects: First, they can be translated versions of the artifact's reference images. In this case, the transformation matrix is easy to find and this transformation is the most common one. Secondly the GUI objects can be scaled in a sometimes complex manner, e.g. by shifting sub-parts of the object to different locations and changing the aspect ratio between feature points. The combination of both transformation is the third possibility. It is hard to impossible to find a single transformation matrix for the last two of the mentioned transformations, but sometimes sub-parts of the image can be recognized more easily. For this reason the VAD in the current implementation mainly focusses on detecting the first kind of transformation reliably and leaves the improvements in the recognition of the complex transformation for future work.

The following sections will describe the choice of algorithms for each step and their configuration details.

3.2.1 Feature detection with ORB

The set of available algorithms for feature detection were provided by the implementations in Emgu CV, respectively OpenCV. From those, as shown in secondary literature such as Tareen et al. [TS18], ORB configured to find a limited amount of features provides the best combination of resource usage and result quality compared to the other three algorithms.

The ORB (*Oriented FAST and Rotated BRIEF*) keypoint detector and descriptor is a "very fast binary descriptor based on BRIEF [...], which is rotation invariant and resistant to noise" [Rub+11, p. 1]. It is specifically designed for real-time systems and low computing power, thus being faster than most of the other currently available methods [Rub+11; TS18]. As the name states, it combines an enhanced version of the *FAST* [RD06] keypoint detector with *rotation-aware BRIEF* (*rBRIEF*) [Cal+10] descriptors. [Rub+11; RD06; Cal+10]

The FAST (*Features from Accelerated Segment Test*) detector [RD06] is a corner detector designed for real-time applications, working on pixel-regions with fixed radius. The ORB detector utilizes an extended version of the FAST detector with a patch radius of 9. This radius has proven to yield the most reliable results, where reliability is defined as the rate of detecting the same corners in multiple views of an object. This reliabilty is proven in the evaluation results in section 4.2.1 as well. [RD06; Rub+11]

The extended FAST detector employed in ORB is used to detect corners on a pyramid scheme for scale variants [KM08] of the image. At instantiation of Emgu CV's implementation of ORB one can adjust the scale factor and number of levels for those pyramids. For the VAD the preconfigured scale factor 1.2 with 8 levels was used. Since the VAD currently responds poorly to scaled versions of the artifact's reference images these settings could be adjusted in future versions (see chapter 4).

After detection, a *Harris corner filter* [HS88] for rejecting edges is applied to the top keypoints with the strongest FAST responses [Rub+11]. The feature detection process in the Emgu CV implementation of ORB is done until a configured amount of features are found (if possible), the preconfigured amount in Emgu CV being 500 features. This amount was doubled for the VAD to extract up to 1000 features per image, as a higher value allows for better differentiation between the reference images. An even higher amount of features affects the execution time negatively while not resulting in better matches for the provided reference screenshots.

There are more configuration parameters available for ORB, but the preconfigured values were found to be sufficient for this task.

3.2.2 Descriptor matching

The matching step tries to find one or more descriptor of the model image's keypoints (called the *training set*), for each of the observed image's keypoints (called the *query descriptors*). For feature matching, there are only two possibilities implemented in Emgu CV: Brute-force matching and *FLANN* (*Fast Library for Approximate Nearest Neighbor*) [ML09] based matching. There are different matching algorithms implemented for both of them such as the *k-nearest-neighbor* algorithm (*k-NN*).

FLANN based *k*-NN-matching performs best for large datasets with high dimensionality, but is less likely to find all possible candidates for matches [ML09]. The trade-off between the focus on speed and accuracy mentioned in section 3.1 lead to the choice of FLANN based k-NN-matching for better performance. For the VAD *k* was set to 2, so that for each query descriptor, the 2 nearest-neighbor descriptors from the training set are returned – if possible. The choice of *k* allows to apply several post-processing filters (see subsection 3.2.3) which reduce the overall recognition time.

3.2.3 Outlier rejection

The best *k* matches for each query descriptor are filtered in several steps before trying to find a transformation matrix using the *Random Sample Consensus* (*RANSAC*) [FB81] algorithm. These filtering steps try to exclude as many match candidates as possible from the given query descriptor set so that the process can be stopped early if there are not enough match candidates left. The match candidates that do not match the correct features are called *outliers*, while the ones that do are called *inliers*.

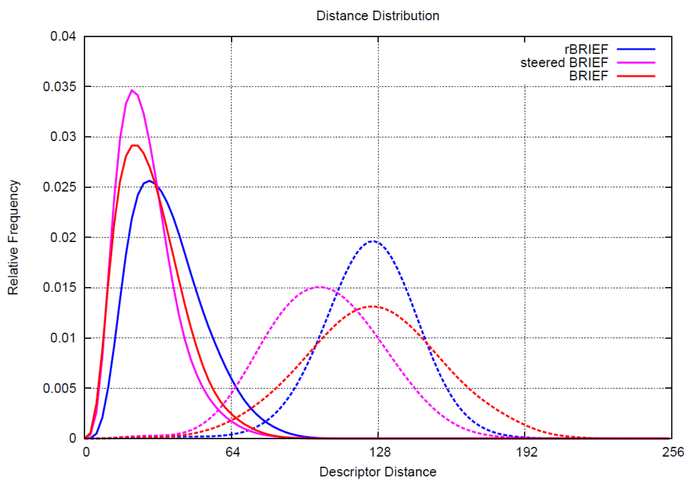


Figure 2: *Analysis of the relation between descriptor distance and frequency of the match being an inlier or outlier, dotted for outliers and solid for inliers [Rub+11, p. 4].*

Since the 2-NN-matching can yield less than two matches for some query descriptors, only those descriptors that have two matches are taken into account for the next steps. The first test is the distance threshold filter proposed in the original ORB publication [Rub+11]. Their analysis of the distance of a match compared to the probability of it being an inlier or outlier showed, that above a certain distance a match candidate is unlikely to be an inlier. For the VAD a threshold of 80 was chosen, as it is the approximate turning point of the probability for the candidate being an outlier rather than an inlier when using rBRIEF descriptors (see figure 2). The second filter is an implementation of *Lowe's ratio test* [Low04] with the recommended distance ratio of 0.7 between the first and the second match's distance. After these steps, two functions provided by OpenCV/Emgu CV are used to eliminate duplicates and to rule out candidates that do not have the size and orientation of the majority of match candidates.

Between and after those filtering steps the VAD determines the current count of match candidates and checks if there are still enough for a successful match. They are compared to a fixed threshold parameter min_{matches} of 25 divided by the ratio of the observed and model images' areas. This parameter min_{matches} has been found empirically to be the best trade-off between a suitably high detection rate given the currently implemented artifact types in ITS.APE and still skipping most of the matching process early if a match is improbable. The ratio of the image's areas allow model images to be found that have a small image size comparison to the observed image. This is a necessary factor, as the density of the feature descriptors per image area becomes more sparse if the image size becomes larger.

At the last step of image recognition, if there are enough candidates left, an own implementation of RANSAC is performed. Two match candidates are randomly sampled and are used to calculate translation and scaling factors for a hypothetical transformation matrix. This hypothesis is then tested on all match candidates allowing a transformation error of an $11x11$ pixel patch due to the inaccuracy of floating point arithmetic for C#. If a certain percentage of candidates support it, given by the confidence parameter, a match was found.

The iteration count of RANSAC was set to 1000 iterations. This is derived from the square ofmin_{matches} adjusted upward to the next power of ten to allow for an error margin, respectively a high area ratio factor. In the best case scenario this iteration count allows the minimum of needed match candidates to be combined. The confidence factor was set to 85% via empirical analysis, meaning that only 15% of the remaining candidates may be outliers of the hypothetical matrix.

The calculation of the translation and scaling factors in this part is influenced by the observation of possible transformations for GUI elements. In the current implementation a high detection rate despite arbitrary translation of the Windows GUI elements is the main focus while detecting other, more complex transformations is a secondary concern.

4 Results

The *Artifact Detector* was evaluated in an environment that should closely resemble the real-world application domain of ITS.APE. In section 4.1 the evaluation setup is explained in detail.

There are two main focus points for the evaluation, reflecting the measures given in chapter 1: First, a qualitative analysis of the detection rate for 25 randomly chosen artifact types from the ITS.APE recipes repository is presented. Then a quantitative analysis of the used execution resources follows, using eight specially crafted artifact types to control as many factors as possible. The results for those two parts can be found in section 4.1.

The 2. requirement resulted in the implementation of a *cache* for already processed data of artifact images from previous runs. This leads to drastic runtime improvements in consecutive runs of the same artifact type, but is opposed by another factor: As soon as the size of the cache increases, the duration of loading and decoding increases as well. Even for a small count of artifact types of three or four types the loading time becomes larger than the time saved by caching. Therefore, the cache was discarded and not used during evaluation. Its concept could still be continued in future work (see chapter 5).

4.1 Evaluation setup

For the evaluation Windows 7 was setup on a virtual machine using Oracle VM VirtualBox[VB]. This allowed an easy setup and quick configuration of the available hardware. All official Windows 7 updates were installed on this VM up to those available at the 25th of March 2019. No additional software was installed other than VirtualBox's *Guest Additions*, which are necessary to share data between the host and the virtual machine.

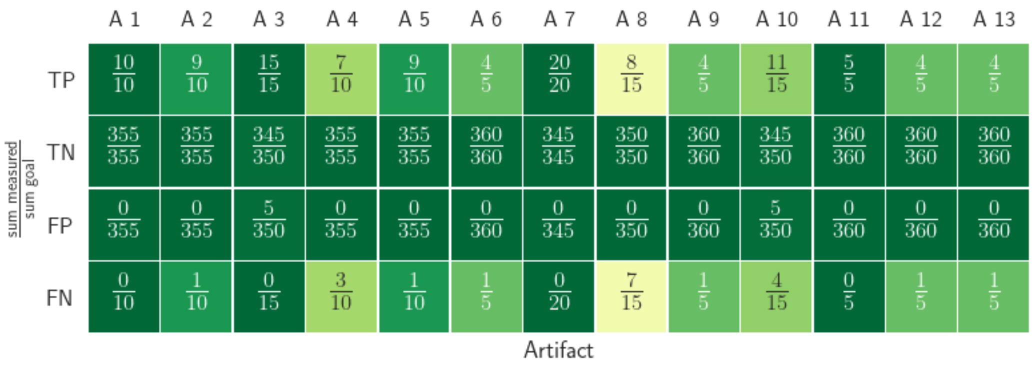
The virtual machine was run with 2GB of RAM available and one core of an Intel Core i5-6600K CPU (3,50 GHz) set to a 40% execution cap, resulting in a single virtual CPU at a speed of 1.4 GHz. These hardware settings were thought to approximate low-tier office hardware sufficiently enough to allow conclusions for the real-world application domain of ITS.APE.

After installing the VAD in the virtual machine using the installer provided, a Batch-script runs the executable multiple times using data supplied in shared folders by the host system. The supplied data was chosen such that it fitted the respective evaluation focus.

4.2 Evaluation results

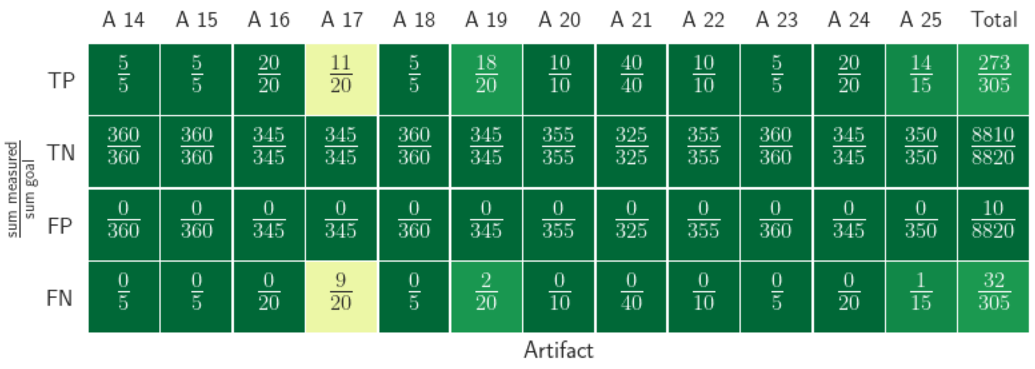
The evaluation results presented in this section prove the successful implementation of the VAD regarding the detection rate and resource usage, but hint to several aspects to improve in future work. This is especially true when it comes to the length of the execution time.

Given the test-setup described in subsection 4.2.1 the detection rate was found to be at about 90% in total. While the execution time for artifact types with few or conveniently ordered reference images has a median of less than 500ms per run, the amount and order of reference images seem to influence the execution time in a linear way. For each additional image that has to be loaded and processed, the VAD needs an extra 150ms to 250ms to compute a result (see subsection 4.2.2). This can lead to execution times deemed too long for application in the ITS.APE environment and thus needs improvement.



	A 1	A 2	A 3	A 4	A 5	A 6	A 7	A 8	A 9	A 10	A 11	A 12	A 13
TP	10/10	9/10	15/15	7/10	9/10	4/5	20/20	8/15	4/5	11/15	5/5	4/5	4/5
TN	355/355	355/355	345/350	355/355	355/355	360/360	345/345	350/350	360/360	345/350	360/360	360/360	360/360
FP	0/355	0/355	5/350	0/355	0/355	0/360	0/345	0/350	0/360	5/350	0/360	0/360	0/360
FN	0/10	1/10	0/15	3/10	1/10	1/5	0/20	7/15	1/5	4/15	0/5	1/5	1/5

(a) Results for artifacts A1 to A12.



	A 14	A 15	A 16	A 17	A 18	A 19	A 20	A 21	A 22	A 23	A 24	A 25	Total
TP	5/5	5/5	20/20	11/20	5/5	18/20	10/10	40/40	10/10	5/5	20/20	14/15	273/305
TN	360/360	360/360	345/345	345/345	360/360	345/345	355/355	325/325	355/355	360/360	345/345	350/350	8810/8820
FP	0/360	0/360	0/345	0/345	0/360	0/345	0/355	0/325	0/355	0/360	0/345	0/350	10/8820
FN	0/5	0/5	0/20	9/20	0/5	2/20	0/10	0/40	0/10	0/5	0/20	1/15	32/305

(b) Results for artifacts A13 to A24 and their totals.

Figure 3: *Detection rate for all runs by artifacts A 1 to A 24 and their totals*

4.2.1 Detection rate

For measuring the detection rate, 25 randomly chosen artifact types from ITS.APE's original artifact library have been selected. They have been numbered A 1 to A 25 according to table 1 in the appendix. Each of their reference images has been merged with a collection of five different Windows desktop screenshots to create a large set of testing screenshots. This merging is done programmatically and places the reference image at a random position on the desktop screenshot such that at least 90% is still inside the images boundaries. In addition to the reference images, screenshots of objects that do not belong to any artifact type, but resemble common desktop applications have been processed in the same way. This leads to a total of 365 generated screenshots for the quantitative evaluation.

For the test run, the VAD was executed with each of the 25 artifact types as targets on each of the 365 screenshots (9125 executions in total). The returned value has then be compared to whether the artifact type is actually present in the screenshot. The summed up measurements for each artifact type can be found in figure 3, where the y-axis denotes the categories *true positive* (TP), *true negative* (TN), *false positive* (FP) and *false negative* (FP). Each cell shows the relation between the actual results (numerator) and the correct result (denominator). In the last column of subfigure 3(b), the summed up results of all artifacts can be found. The cells are color coded in a spectrum from red at 0% over yellow at 50% to green at 100% for TP and TN and vice-versa for FP and FN.

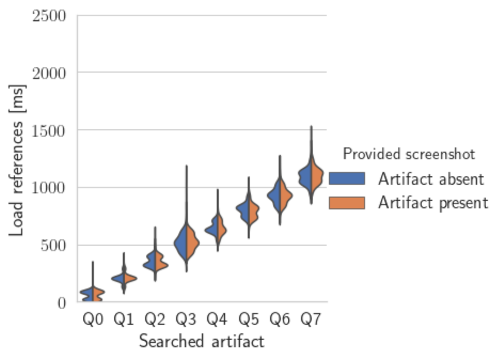
The results show that the sensitivity of the result is $> 88\%$ for the summed up total, while the specificity is at $> 99\%$. Although the sensitivity is above 50% for all tests, some artifact types show a significantly lower sensitivity. This is especially prominent for A 4, A 8, A 10 and A 17, for which the VAD has a detection rate between 53% and 70%. This is most probably due to these types' reference images, which have very few distinct features even for human perception. Anomalies in the specificity are only found for A 4 and A 10, with each of them having five false positive matches. Cross-checking the screenshot collection showed that a highly similar reference picture was used for both artifact types, only differing in a small text region and resulting in five evaluation screenshots each. Those were then identified as a match by the opposite artifact type. In conclusion, both error sources can be depleted by choosing more distinct and feature-rich reference images for the artifact types.

A second test run under the same conditions showed the exact same detection rates for all executions, proving the robustness of ORB's extracted features.

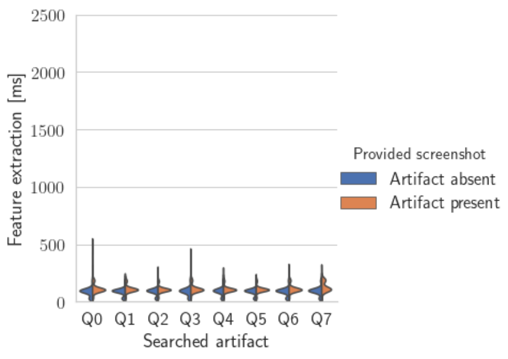
For measuring the detection rate of scaled versions of the artifact types, a second collection of screenshots was generated containing stretched versions of the reference images. This resulted in a sensitivity below 50%, showing that the algorithm does not reliably detect such scaling. But since the artifacts would not be transformed in this way for real-world use-cases this result is not substantial for the assessment of the overall success.

4.2.2 Execution resources

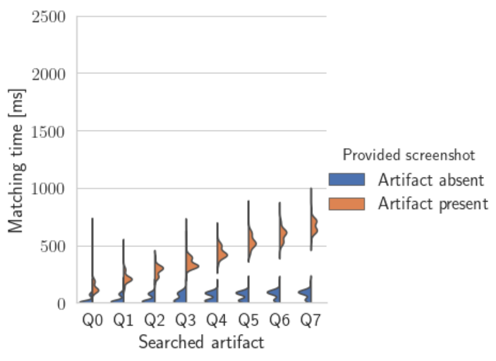
A different setup was used for the quantitative analysis of the VAD: Eight new artifact types were constructed, named Q0 to Q7. If the artifact is actually present in the tested screenshot the artifact's index denotes at which position of their provided reference images the only matching image to the queried screenshot is located. The matching reference image is always the last reference image to test, so that Q0 has one reference image in total, Q1 has two etc. There are only two screenshots used for the tests, one with the artifact present ($s_a p$) and one without, the latter being an empty desktop screen ($s_a a$). The test is set up such that only the last step of outlier rejection is skipped if the artifact is present in the screenshots, namely the execution of RANSAC to recover a transformation matrix. This is achieved by choosing a non-matching reference image which has



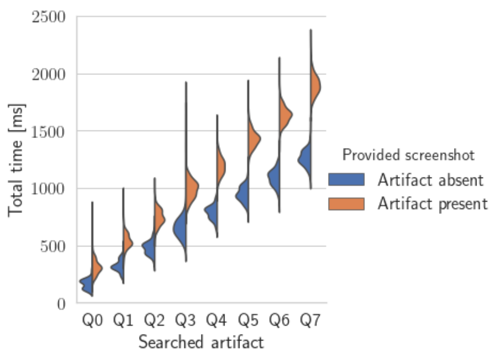
(a) Execution time for loading the searched artifact's reference images.



(b) Execution time for extracting the observed screenshot's features.



(c) Execution time for matching the feature descriptors and finding a transformation matrix.



(d) Execution time sum of all three image recognition parts.

Figure 4: *Execution times for artifacts Q0 to Q7*

enough similarities to the matching one and allows to assess the influence on the runtime of the RANSAC implementation.

Each of the two screenshot gets queried 1000 times for each artifact type Q0 to Q7, resulting in 16000 executions of the VAD in total. The high count of repetitions is used to even out randomly occurring errors that have an influence on the execution time so that the sample is as representative for the execution time as possible. The results are depicted in figure 4 using a violin plot with a kernel density estimator of 0.5.

The first plot in figure 4(a) shows the execution time in ms needed for loading all of the type's reference images and extracting their features. The increase of the duration dependent on the image count can be assumed to be linear given the implementation. This is an expected behaviour and matches the results of the second plot in figure 4(b) which is showing the duration of the same process for the observed screenshot. Since for all runs there is only one observed screenshot to extract features from, the time needed for this task is constant in a certain error margin. Both plots also show the independence of these steps from the actual absence or presence of an artifact.

Plot three in figure 4(c) shows the duration of the last two step of the recognition process (from here: *matching time*), namely feature matching and outlier rejection. One can see how the presence of the artifact in the image drastically increases the matching time. With only eight of such artifact types it is not possible to distinguish the type of increase depending on the reference count, but a linear increase is probable. The difference in execution time for $s_a p$ and $s_a a$ show, how the threshold checking of the match count prevents the execution of further filtering steps and thus saves execution time. It also shows a large influence by RANSAC on the execution time.

In figure 4(d) the sum of the previous three values is plotted. It shows that the image recognition process of the VAD takes at least 100ms ($s_a a$) to 250ms ($s_a p$) and can take up to 2000ms or more, if there are multiple reference images in an inconvenient order. The plot also shows that there is a significant difference for $s_a p$ and $s_a a$ in execution time which is only due to the difference in matching time.

There are major outliers for the measured times of each step, being up to double to triple the time of the median. The reason for this is likely to be be a delay in access to the file storage if it occurs while loading and extracting the references or screenshot. For all steps it is also possible, that an increased CPU-usage by some other program decreased the resources available for the VAD, although such an execution was tried to be suppressed in the virtual machine.

Since the tests are run in a virtual machine, the built-in tools for VirtualBox have been used for measuring the resource consumptions at random points during the execution. The RAM usage of one of VAD's instances was between 30MB and 45MB consistently, while the CPU usage was at almost 100% for all runs.

5 Conclusion

The evaluation showed the importance of the quality of reference images for the detection rate, although for most types in the recipes repository the reference images were *good enough* to yield a detection rate of $\geq 90\%$. This detection rate is most probably going to suffice for ITS.APE if it can be confirmed in the real-world. The chosen parameters of the VAD can still be adjusted based on new results to achieve a higher detection rate. An important aspect for successful employment is the low specificity which means that there are a minimum of false alarms when using the VAD.

The measured execution time can certainly be decreased in future work. Possible solutions could include a refined version of the implemented caching mechanism, e.g. by only caching a single artifact type at once, which is then lazy-loaded only if needed. The choice of parameters for skipping filtering steps during outlier rejection can be fine-tuned as well to achieve a better performance. Especially the implemented RANSAC algorithm could probably be optimized for execution speed.

In conclusion, an overall successful implementation of the VAD has been accomplished in regards to the measures of the detection rate and resource consumption (see chapter 1) and given the test setup. A real-world evaluation is nevertheless highly advisable to e.g. show the influence of other running processes on the execution time.

6 Bibliography

[AMH05] H. B. Aradhye, G. K. Myers, and J. A. Herson. “Image analysis for efficient categorization of image-based spam e-mail”. In: *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. Aug. 2005, 914–918 Vol. 2. DOI: 10.1109/ICDAR.2005.135.

[ARK] Inc. Amazon.com. *Amazon Rekognition*. Website. https://aws.amazon.com/de/rekognition/. Mar. 2019.

[Bak+18] Maxim Bakaev et al. “HCI Vision for Automated Analysis and Mining of Web User Interfaces”. In: *ICWE*. Vol. 10845. Lecture Notes in Computer Science. Springer, 2018, pp. 136–144.

[Cal+10] Michael Calonder et al. “BRIEF: Binary Robust Independent Elementary Features”. In: *Computer Vision – ECCV 2010*. Ed. by Kostas Daniilidis, Petros Maragos, and Nikos Paragios. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792. ISBN: 978-3-642-15561-1.

[CUDA] NVIDIA Corporation. *CUDA Toolkit*. Website. https://developer.nvidia.com/cuda-toolkit. Mar. 2019.

[CVDL19] Slashdot Media. *OpenCV Download Statistics: All Files*. Website. https://sourceforge.net/projects/opencvlibrary/files/stats/timeline. Mar. 2019.

[EMGU] Canming Huang. *Emgu CV: OpenCV in .NET (C#, VB, C++ and more)*. Website. http://www.emgu.com/wiki/index.php/Emgu_CV. Dec. 2018.

[FB81] Martin A. Fischler and Robert C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782.

[GTF] Google LLC Google Brain Team. *TensorFlow*. Website. https://www.tensorflow.org/. Mar. 2019.

[GVAPI] Google LLC. *Vision API*. Website. https://cloud.google.com/vision/. Mar. 2019.

[HS88] Chris Harris and Mike Stephens. “A combined corner and edge detector”. In: *In Proc. of Fourth Alvey Vision Conference*. 1988, pp. 147–151.

[KM08] Georg Klein and David Murray. “Improving the Agility of Keyframe-Based SLAM”. In: *Computer Vision – ECCV 2008*. Ed. by David Forsyth, Philip Torr, and Andrew Zisserman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 802–815. ISBN: 978-3-540-88688-4.

[Low04] David G. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision* 60.2 (Nov. 2004), pp. 91–110. ISSN: 1573-1405.

[ML09] Marius Muja and David G. Lowe. “Fast approximate nearest neighbors with automatic algorithm configuration”. In: *In VISAPP International Conference on Computer Vision Theory and Applications*. 2009, pp. 331–340.

[MP18] Maxim Mozgovoy and Evgeny Pyshkin. “Unity Application Testing Automation with Appium and Image Recognition”. In: *Tools and Methods of Program Analysis*. Ed. by Vladimir Itsykson, Andre Scedrov, and Victor Zakharov. Cham: Springer International Publishing, 2018, pp. 139–150. ISBN: 978-3-319-71734-0.

[NET] Preeti Krishna. *Announcing the .NET Framework 4.7.2*. Website. https://devblogs.microsoft.com/dotnet/announcing-the-net-framework-4-7-2/. Apr. 2018.

[RD06] Edward Rosten and Tom Drummond. “Machine Learning for High-Speed Corner Detection”. In: *Computer Vision – ECCV 2006*. Ed. by Ale Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443. ISBN: 978-3-540-33833-8.

[Rub+11] Ethan Rublee et al. “ORB: An Efficient Alternative to SIFT or SURF”. In: *Proceedings of the 2011 International Conference on Computer Vision*. ICCV ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 2564–2571. ISBN: 978-1-4577-1101-5. DOI: 10.1109/ICCV.2011.6126544. URL: http://dx.doi.org/10.1109/ICCV.2011.6126544.

[Syk15] Arnold Sykosch. *IT-Security Awareness Penetration Testing – ITS.APT*. Website. https://itsec.cs.uni-bonn.de/itsapt. July 2015.

[TS18] S. A. K. Tareen and Z. Saleem. “A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK”. In: *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*. Mar. 2018, pp. 1–10. DOI: 10.1109/ICOMET.2018.8346440.

[VB] Oracle. *Oracle VM VirtualBox*. Website. https://www.virtualbox.org/. Mar. 2019.

[VLF] The VLFeat Authors. *VLFeat - Home*. Website. http://gs.statcounter.com/screen-resolution-stats. Mar. 2019.

*

Appendices

Table 1: *Mapping of the abbreviated artifact names to their full names*

Abbreviation	**Full name**
A 1	01_Jibberish-Mittel
A 2	02_Link_Only-Schwer
A 3	03_Targo_Bank-Schwer
A 4	04_Targeted_IT_Abteilung-Einfach
A 5	04_Targeted_IT_Abteilung-Schwer
A 6	05_Newsletter-Schwer
A 7	06_Versichertenkarte-Schwer
A 8	07_Paypal-Mittel
A 9	08_Weiterbildung-Schwer
A 10	12_IT_Ticket-Mittel
A 11	browser_advertisement
A 12	browser_defacing
A 13	email-bounce-exchange-de
A 14	exe_anti_virus_extended
A 15	exe_anti_virus_simple
A 16	exe_file_scanner
A 17	exe_login_window
A 18	exe_self_remove
A 19	exe_updater_generic
A 20	exe_updater_human
A 21	exe_updater_java
A 22	exe_updater_remote
A 23	exe_updater_simple
A 24	ms_word_macro
A 25	ms_word_protected_view

List of Figures

List of Tables

Statement of Authorship

I hereby confirm that the work presented in this lab report has been performed and interpreted solely by myself except where explicitly identified to the contrary. I declare that I have used no other sources and aids other than those indicated. This work has not been submitted elsewhere in any other form for the fulfilment of any other degree or qualification.

Bonn, March 29, 2019

Felix Rossmann