Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/MBE

MBE, 16(6): 7526–7545.
DOI: 10.3934/mbe.2019378
Received: 21 May 2019
Accepted: 29 July 2019
Published: 19 August 2019

Research article

Apply computer vision in GUI automation for industrial applications

Yung-Pin Cheng*, **Ching-Wei Li and Yi-Cheng Chen**

Department of Computer Science and Information Engineering, National Central University, Zhongli District, Taoyuan City 32001, Taiwan

* **Correspondence:** Email: ypcheng@csie.ncu.edu.tw; Tel: +88634227151 ext 35210; Fax: +88634222681.

Abstract: Technology has reshaped the workplace and the rapid improvements have transformed how we work nowadays. In the pursuit of industry 4.0, we build smart machines and robots to replace manual labor. While the manual labor is replaced by machines, in many cases, humans are transformed into desktop software users. Jobs such as testing, quality inspection, data monitoring, data entry, and routine editing remain to be done by humans in front of desktop computers. The operations to software applications in principle can be reduced to screen output understanding and mouse and keyboard operations. When the characteristics of these jobs are repetitive, tedious, and monotonous, they can be replaced by GUI automation techniques. GUI automation can be achieved by different underlying technologies, each has its pros and cons. In this paper, we describe a tool-Korat, which uses computer-vision to achieve maximum cross-platform capability for industrial applications, including test automation and robotic process automation. Although Korat has been successfully adopted by several industrial customers, difficult problems remain to be addressed. The problems and difficulties in applying computer vision for GUI automation are discussed and studied in this paper, particularly the experiences of applying open source OCR to GUI automation over color screenshots. By introducing critical pre-processing stages and algorithms, the recognition rate is significantly increased and becomes feasible for practical usage.

Keywords: GUI automation; computer vision; test automation; optical character recognition; image analysis

1. Introduction

In the areas of software engineering or industrial applications, which involves hardware and software integration, testing is often performed by testers or programmers using keyboards and mouses. When graphical user interface (GUI) is involved, a tester may use a keyboard and a mouse to drive a test run and then monitor the system behaviors at the same time to assert the correctness of a test run. Since the steps of a test run are often repetitive and monotonous, testers often try to find a suitable testing tool to automate the task. The basic features of a software testing tool are reproducing the keyboard and mouse events that drive a test on a system under test (SUT) reliably at good timing and then asserting the correctness of the system under test in the run.

Technically speaking, what these testing tools perform is actually *regression testing*. The goal of a test is trying to expose the bugs hidden in the test run but a regression test is a repeated test to make sure system features that functioned correctly remains unaffected by the changes of the code due to bug fixes, feature extension, refactoring, etc. True GUI test automation where test cases are generated and tested automatically remains to be too hard to be practical in foreseeable future [1]. On the other hand, regression testing is practically doable, necessary, and indispensable for industrial applications. So, in software industry or other industrial areas nowadays, testers remain to be responsible for designing the test cases and preparing the test data to test the code of a SUT.

The underlying technology of test automation is GUI automation. Test Automation is one of the well-known industrial applications of GUI automation. Other applications are robotic proccess automation (RPA) or grinding in video gaming, where grinding refers to the playing time spent doing repetitive tasks within a game to unlock a particular item or to build the experiences needed to progress. GUI automation, at first glance, should be matured and commercialized to some extent already and applicable to practical industrial and personal needs. However, GUI automation has limited its usage in a few narrow domains for several reasons.

First, automating the keyboard and mouse events can be straightforward if operating systems or the platforms (e.g., web browser) under test supports GUI component accessing and invocation APIs. Popular commercial or open source testing tools typically adopt such an approach. To click a GUI button, these approaches actually invoke its *click()* method instead of moving mouse cursor onto the button and then performing a click action. We call such an approach platform-dependent (PD) GUI automation. For instance, popular open-source testing tool, Selenium [2], relies on the testing support from Firefox browser in its earlier versions. In Windows, commercial testing tools [3, 4, 5, 6] and robotic process automation (RPA) tools [7] mainly rely on some Windows APIs from .Net framework.

Techniques that adopt such an approach, inevitably limit their use to a specific platform but not all the operating systems or platforms provide needed API support for GUI automation. The problems of relying on such an approach are:

- backward compatibility, for example, not applicable to old applications which were built by the early platform libraries,
- inapplicable to cross platform testing or automation, and
- unpredictable time delay to keep up with new emerging technology advances.

On the other hand, instead of adopting these platform dependent approaches, it is intuitive to approach the problem by mimicking humans. Humans process the screen output and then control the mouse and keyboard to drive the GUI automation. We call these approaches, computer-vision (CV) based GUI automation. For example, test automation tools adopting these approaches include Sikuli [8], T-Planrobot [9], Egg-plant[10], and Korat [11]. The merits of this approach are:

1. not sensitive to platform support and technology advances,
2. applicable to several platforms such as Windows, Linux, and Web,

3. capable of verifying the program's GUI errors, such as layouts, and
4. capable of verifying program output that are rendered in image pixels.

However, the drawbacks of CV approach are:

1. GUI component properties (numeric values or strings) are not accessible in programming,
2. GUI component event triggering is not guaranteed,
3. GUI structured information, such as the elements in a list box, is not available in programming, and
4. performance can be slower.

Performance in item 4 in general is not a critical issue in practice. First, reliability of a black-box test is often more important than speed. Second, the bottlenecks of testing performance are often or mainly attributed to the response time of systems under test.

Technically speaking, CV approaches in several cases make the GUI automation problem more difficult to deal with, i.e., CV approaches turn a programming problem into an image processing problem. Particularly, programming is still an indispensable part in applications such as test automation and RPA. So, it is not surprising that these applications favor PD approach.

However, through years of development and experiences in applying and promoting Korat in several industrial applications, we now believe CV approach can be a competitive solution in many scenarios. We have encountered several industrial customers who tried to adopt PD-based solutions but eventually gave up for different reasons, such as backward compatibility and cross-platform capability. Many industrial applications have legacy and cross-platform issues, which makes PD approaches inapplicable or failed from time to time.

In this paper, we will describe the computer vision problems and how we address these problems practically in Korat. One of the most critical computer vision problem and challenges for CV GUI automation is optical character recognition (OCR). OCR is in general remains to be a difficult problem in practice. Commercial product such as ABBYY [12] or Google Vision [13] have already dominated the market in various domains. However, although these services have been available for some time and they provide adequate accuracy in general, cost and their technical constraints are always problems. For example, applying Google Vision cloud service requires internet connections, but in our applications to industry, most Korat users are not allowed to have internet connections in their factories. In addition, Korat's test cases may run in 24 hours. Cost becomes a real issue and could not be ignored. So, it is inevitable for us to seek out open-source solutions as an alternative. In this paper, we describe how we improve and enhance an open source OCR tool-Tesseract [14] for the character recognition on the screenshots.

This paper is organized as follows. In Section 2, we compare the techniques of test automation. Section 3 gives an overview of Korat, which adopts computer vision in GUI automation for testing. The major computer vision problems in GUI automation are discussed in Section 4. The improvements and evaluations are at the end of Section 4. Section 5 ends the paper with conclusion remarks.

2. Related work

Technically speaking, GUI automation is only an underlying technology to reproduce UI events (such as mouse and keyboard events) so that a GUI desktop software can be automated without human intervene. Among its applications, regression test automation is the most well-known one. Regression test automation can be done by programming, tools, or a combination of them. Systems with sophisticated GUI are often difficult to automate. Besides, system testing must wait until an executable program is built. So, writing xUnit test code has become a popular approach to support regression test automation at the programming level. However, xUnit testing requires programming skills and considerable code maintenance efforts [15]. For several reasons, unit tests can never replace system tests.

Instead of programming test cases, Capture/Replay (CR) is an advanced feature in software regression testing. Its basic idea is to record the operations of users, particularly in a GUI, into a test script, mainly consisting of keyboard and mouse events. When a system under test (SUT) is modified, the test script is replayed to see if the SUT is damaged by the change. In the replay, the previously saved keyboard and mouse events are sent to SUTs to emulate the testing behaviors of a tester. If a test can be successfully replayed, the test run is passed. Since Capture/Replay is a very practical approach, many commercial CR tools have been built. Examples of some CR tools are: T-Plan Robot [9], HP QuickTest [3], Rational Robot [16], etc..

Capture/Replay approaches may sound like a straightforward method for software regression testing but in practice they are often complicated by a lot of problems [17]. First, an SUT's execution time for the same test can be different in two separate runs. The timing to trigger a series of events may not always be the same between two different runs. So, a straightforward replay is often infeasible. Second, to intercept the mouse/keyboard events, CR tools often cause performance interference to the SUTs. There are a few ways to intercept mouse events. If monitored events are not local to the applications, a global hook is often used to intercept the mouse events from O.S. Unfortunately, hooks tend to slow down the system because they increase the amount of processing the system must perform for each message. The interference can be so significant that the temporal synchronization between a capture run and a replay run becomes difficult. Mouse dragging events are the most common GUI events that often fail to synchronize replayed runs in a precise manner [17]. Automating the CR tests for 3D games, for example, is even more challenging. Since no GUI components can be captured in a 3D scene, an intrusive CR approach was proposed [18].

In most software applications, human testing behaviors are mainly keyboard/mouse operations guided by human sight. A tester not only work as a test driver but also a test oracle to monitor and assert the correctness of a test run. So, in a human test, the human brain, eyes, and hands combine to play the roles of test driver and test oracle. So, to emulate what humans do, one intuitive CR approach is to analyze screen output (image pixels) and guide the mouse and keyboard to repeat a capture run. This CR approach is CV-based in this sense. The representative image-analysis CR tool is Sikuli [8]. Sikuli proposes a GUI scripting language called Sikuli, in which the sequence of operations of a test, called visual workflow, is recorded. The advantage of Sikuli script is its visual and easy-to-understand characteristics. In principle, limited programming skills are required for using Sikuli for software testing. One major drawback of Sikuli is the performance interference caused by intercepting mouse events with global hooks. Another drawback is that Sikuli's test scripts are often sensitive to the failures of image analysis and recognition. The major image recognition approach provided by Sikuli is template matching from OpenCV, which can fail by false positive cases (i.e., objects are incorrectly identified), and false negative cases (i.e., objects are incorrectly rejected) in practice.

Korat's initial work was first published in [11]. In this previous work, the prototype of Korat was presented to address the BIOS testing scenarios in industrial personal computer manufacturing, where a test case is across from BIOS, OS boot up, to Windows/Linux GUI systems. Note that in the stages of BIOS and OS boot up, there are no commercial operating systems to support testing need. Korat introduced a two-machine testing architecture to address the problem and achieved test scenarios which are capable of crossing platforms. In the best of our knowledge, Korat is unique in such a domain.

Another work that is closely related to Korat is T-Plan Robot [9]. Unlike Sikuli, T-Plan Robot is run at a separate machine from SUT, which is the same approach as Korat. Running CR tools on a separate machine can guarantee no performance interference on the execution of SUT. However, the problem is how to intercept mouse/keyboard events from SUTs. The answer of T-Plan Robot is to run a VNC (Virtual Network Computing) server on the SUT machine. VNC is a common tool for users to remotely control another computer. It transmits the keyboard and mouse events from one computer to another and graphical screen updates back in the other direction over a network. So, T-Plan robot can be platform independent wherever a VNC server can be installed. Compared to Korat's approach, a VNC server can still cause performance interference to SUTs. Besides, VNC servers are only allowed in platforms which often require a general purpose O.S. For example, it is inapplicable to BIOS testing and many other embedded devices. The platform independence is considered more limited than Korat.

3. Maximize cross-platform capability by CV GUI automation

Mouse devices, keyboards, and touch screens are now the major input devices for most software products. These input devices typically work with a GUI system. As a result, most system test automation tools aim for GUI automation driven by keyboards and mouse devices. One critical fact is that most software products, even mobile applications, are developed in major desktop environments, such as Windows, Macs, or Linux. These environments are mainly operated by a mouse device and a keyboard.

For the purpose of explaining GUI automation approaches, it is convenient to use an example to help explain the technical details behind it. This example is to move the cursor to click a button. When a GUI automation technique tries to reproduce the example move, it is possible to make *a real mouse click*, *an emulated mouse click*, or *a fake mouse click*. Making a fake mouse click is to invoke a GUI component's event handler such as its *click()* method without any real/emulated mouse events. This approach has been widely discussed in previous sections as the platform-dependent approaches.

Making a real mouse click can be done by producing a series of mouse USB signals without a real mouse device but let the SUT recognize it as a mouse device. Tools which adopt this approach are capable of being independent of operating systems and platforms. If USB signals are keyboard signals, it is even applicable to non-GUI environment such as BIOS and DOS. Let the approach be called Method-USB. The major challenge of such an approach is tracking the mouse cursor in an SUT and determining when and where to perform a click. Korat [11] supports such an approach.

Making an emulated mouse click is to insert mouse actions into a GUI system's input queue. For example, Windows is an operating system bundled with a GUI system. It provides APIs to allow processes to insert mouse actions into system input queue to control mouse cursors. Typically, these APIs allow you to move a mouse cursor to a specific screen coordinate and perform mouse button actions. Let the approach be called Method-OS. Tools relying on such an approach inevitably depend on the operating system. Building a tool by Method-OS is comparatively less difficult than Method-

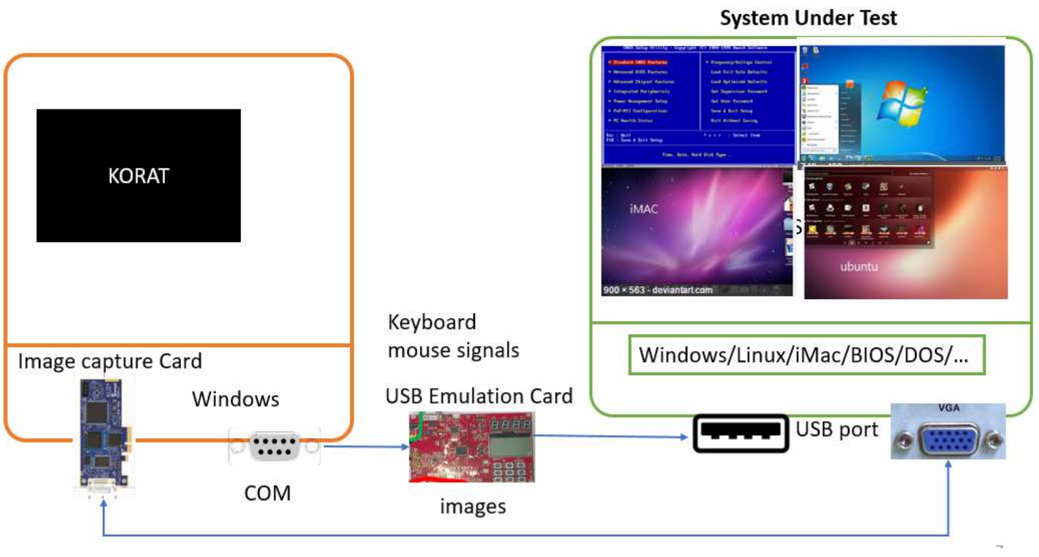


Figure 1. Korat architecture in Method-USB mode.

USB because you can precisely navigate the mouse cursor. However, the problem of determining when and where to perform a click is the same as Method-USB. Tools that adopt this approach include Korat, Sikuli [8]. Sikuli and Korat are capable of driving SUTs under the same operating system. Basically, this approach needs image analysis and computer vision methods to determine where to click.

3.1. Modes of cross-platform

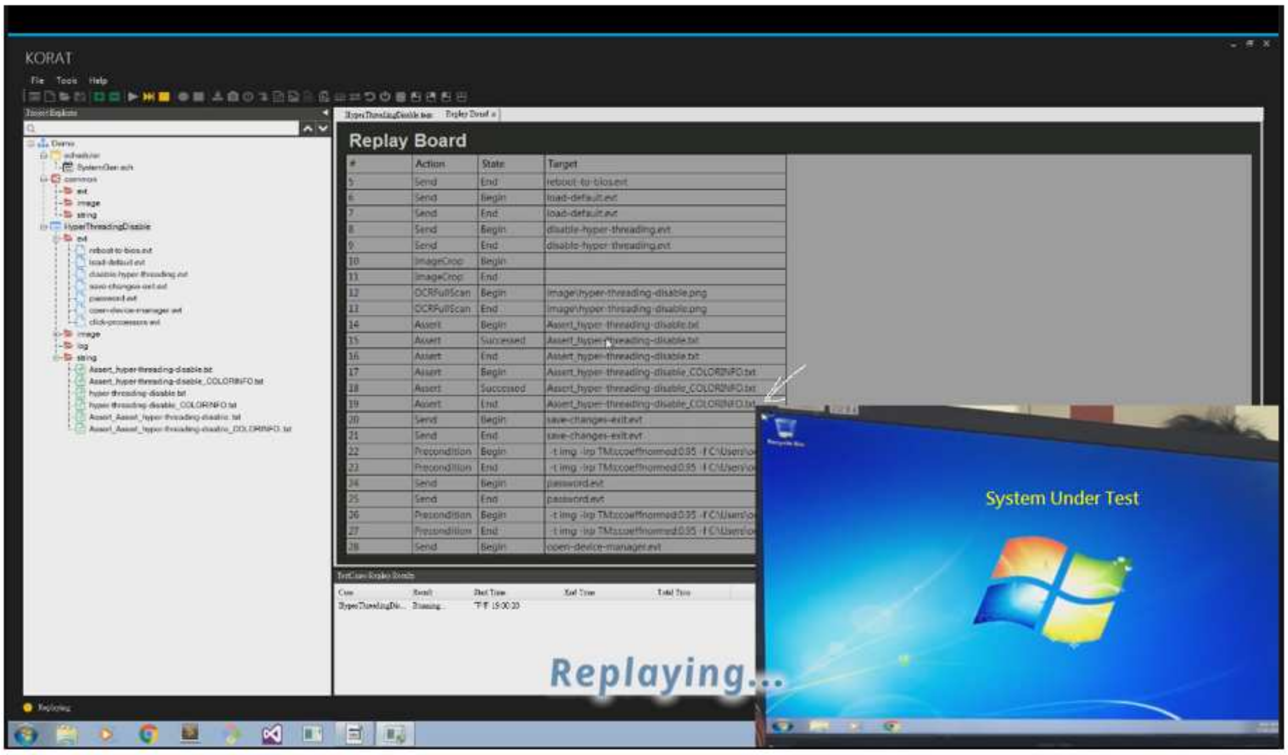
In Figure 1, the architecture of Korat in Method-USB mode is illustrated. As shown in the figure, Korat is run at a separate machine from SUT but the SUT's video output is connected to a video image capture card installed in the Korat machine. In this mode, Korat is a two-system testing tool but it is completely non-intrusive. An ARM-cortex M4 development board is used as a USB emulator to intercept and send keyboard/mouse events to SUT's USB ports.

Korat's main GUI program is shown in Figure 2. When a tester clicks the "record" button, the screen of the SUT is shown in a window (captured from the video signal) for the tester to operate. In recording mode, a tester's operations, including mouse and keyboard events, are all intercepted and saved by Korat but sent to the SUT at the same time.

Figure 3 shows the Korat in Method-OS mode. When Method-USB is too expensive to deploy, Korat can run in this mode to capture the image from screen and control mouse without the USB emulator. In this mode, Korat is not limited to test Windows programs. Programs in other platforms can be tested via virtual machines, VNC, or TeamViewer as well.

3.2. Combinations of image recognition methods

Similar to other CV-based tools, Korat uses image recognition methods such as template matching to find the target to click or drag. Even though the captured image is clean and no distortion, image processing techniques are inevitably troubled by false-positive and false-negative errors. When an image recognition fails, Korat allows users to adjust the threshold of the recognition. While most cases can be resolved in this way, adjusting thresholds in some cases is a knife with two edges. For example,



#	Action	State	Target
5	Send	End	reboot-to-bios.evt
6	Send	Begin	load-default.evt
7	Send	End	load-default.evt
8	Send	Begin	disable-hyper-threading.evt
9	Send	End	disable-hyper-threading.evt
10	ImageCrop	Begin	
11	ImageCrop	End	
12	OCRFullScan	Begin	image\hyper-threading-disable.png
13	OCRFullScan	End	image\hyper-threading-disable.png
14	Assert	Begin	Assert_hyper-threading-disable.txt
15	Assert	Succeeded	Assert_hyper-threading-disable.txt
16	Assert	End	Assert_hyper-threading-disable.txt
17	Assert	Begin	Assert_hyper-threading-disable_COLORINFO.txt
18	Assert	Succeeded	Assert_hyper-threading-disable_COLORINFO.txt
19	Assert	End	Assert_hyper-threading-disable_COLORINFO.txt
20	Send	Begin	save-changes-exit.evt
21	Send	End	save-changes-exit.evt
22	Precondition	Begin	-t img -irp TM:ccoeffnormed:0.95 -f C:\Users\o
23	Precondition	End	-t img -irp TM:ccoeffnormed:0.95 -f C:\Users\o
24	Send	Begin	password.evt
25	Send	End	password.evt
26	Precondition	Begin	-t img -irp TM:ccoeffnormed:0.95 -f C:\Users\o
27	Precondition	End	-t img -irp TM:ccoeffnormed:0.95 -f C:\Users\o
28	Send	Begin	open-device-manager.evt

Figure 2. The screen-shot of Korat and an SUT screen window.

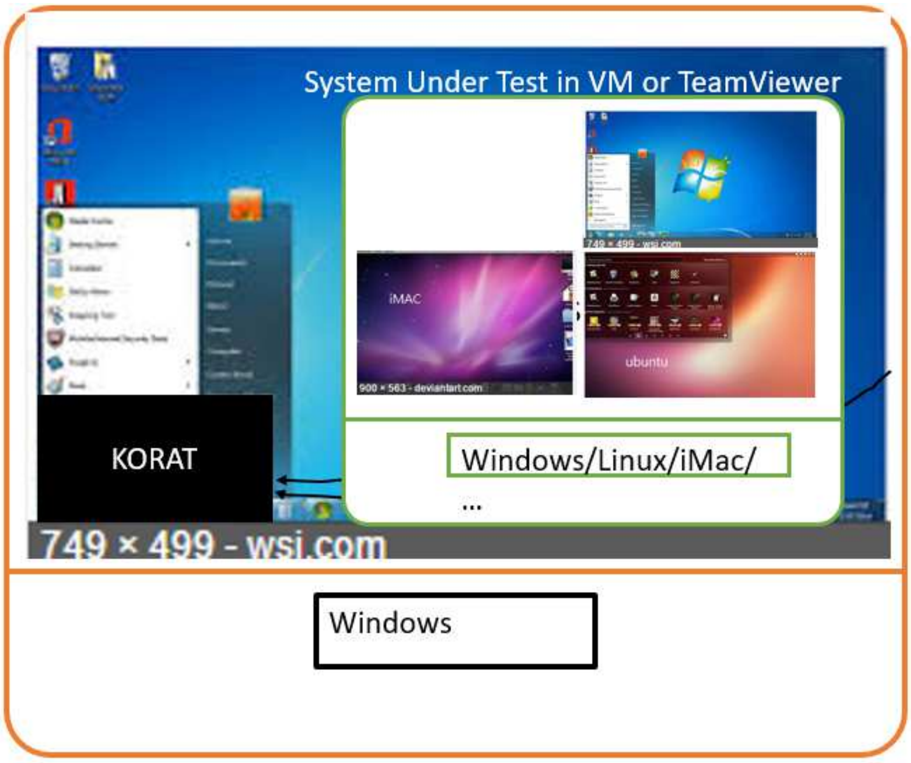
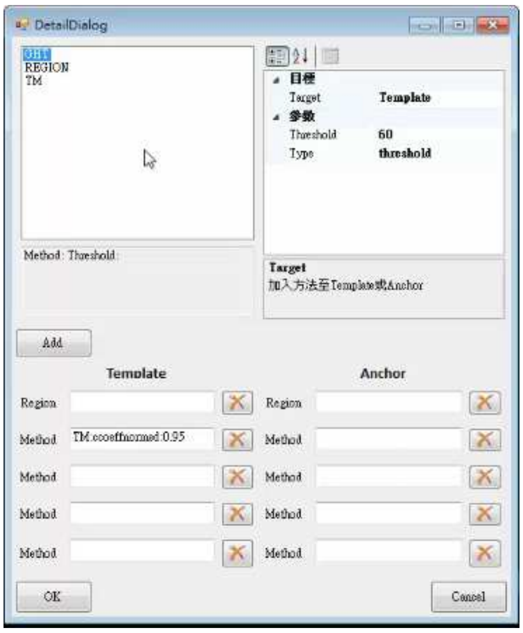


Figure 3. Korat in Method-OS mode.



目標
Target Template
參數
Threshold 60
Type threshold

Method: Threshold

Target
加入方法至Template或Anchor

Template Method: TM.ccoeffnormed:0.95

Figure 4. The dialog to create combinations of image recognition methods.

fixing a false-positive problem may cause more false-negative errors to occur. This dilemma often can be resolved by using other image recognition method in Korat. Korat currently provide several basic image recognition methods:

- Template matching (colors are critical)
- Template matching without background (only foreground template pixels are computed)
- Generalized Hough transform (contours are critical but color and scale are irrelevant)
- Anchored Images (use an additional image to filter the duplicated candidates)

Korat provides a tool to allow flexible combinations of above methods (see Figure 4 for the user interface). Basically, Korat allows you to create a combination of image analysis method which could best fit your computer vision problem. Its basic concept is that each method identifies a set of prioritized candidates and you can combine additional methods to filter candidates to recognize the real target. For example, suppose there are two folders with same image icons in the desktop. Template matching may match two targets to be clicked. You can use a recycling bin icon as an anchor to locate the exact target. For example, a selected item may change background color (see Figure 7 in the later section), causing the template matching to fail. A straightforward solution is to use two template matching on two templates (one with white background and one with color background) and then union the results of the two by OR operation so that either one of the two is recognized.

3.3. Assertions

In the test automation scenarios, a tester may judge the correctness of a test run by different methods. It is a tool's responsibility to provide support for common assertions but in the meantime provide an extensible architecture for new assertions to add in. For common assertions, Korat provides the following methods:

- Image recognition

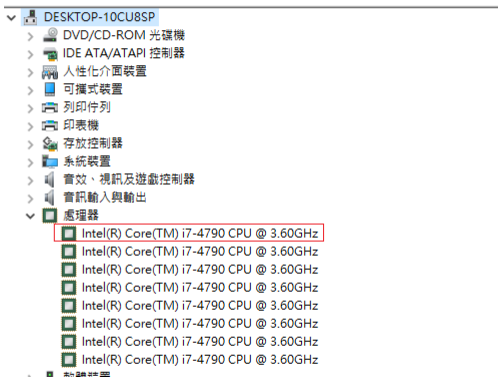

DESKTOP-10CU8SP
DVD/CD-ROM 光碟機
IDE ATA/ATAPI 控制器
人性化介面裝置
可攜式裝置
列印佇列
印表機
存放控制器
系統裝置
音效、視訊及遊戲控制器
音訊輸入與輸出
處理器
Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz

Figure 5. Assertion by counting the number of targets in Windows device manager.

- Fixed font/size - Optical Character Recognition (OCR) (very high recognition rate tailored only for BIOS)
- Google cloud vision OCR
- Drag and select for string processing

The image recognition methods are basically the same as the previous section but the prioritized candidates identified by an image recognition method can further be counted or compared. For example, in many industrial personal computer (IPC) tests, after Korat opens the device manager in Windows, the tests need to unfold a tree entry and assert the number of items. See Figure 5 for example. So, Korat provides an assertion command to compare (≤, =, ≥) the number of prioritized candidates, which is heavily used in IPC's tests.

Since Korat is an CV-based tool, it cannot use a GUI component's property for assertions. So, using OCR to extract texts from images into ASCII texts is a very straightforward solution. However, a generic OCR tool that is capable of recognizing arbitrary fonts and sizes is in general difficult to build. We have studied and tried several OCR tools such as Tesseract [14] and FineReader[12]. They are usable but recognition rate is not precise enough for us to deal with IPC's requirements. In IPC's requirements, they need to recognize a string on screen, such as a product number, in their production lines but the precision must be very close to 100%.

In order to achieve this mission, we designed and implemented a fixed font/size OCR of our own with precision close to 100%. Note that this part of work is completely independent of Tesseract work which will be described later. The trick is to force users to tell Korat the font and size of the texts in the images; that is, by providing heuristic, we can greatly reduce the complexity of the problem. Such a trick allows us to use template matching and dynamic programming to guarantee the high recognition rate. Such an approach, of course, is only feasible to a specific domain for characters with fixed fonts/sizes. However, to IPC industry, they are happy to trade a little inconvenience for recognition rate.

The last interesting method is using Korat's GUI automation to drag and select a region of texts, copy-and-paste, and then extract the texts for assertions. This method is getting momentum in Korat's applications, particularly when the texts are not English characters. Using this method allows Korat to avoid Chinese character OCR, which is considered more difficult than English character OCR.

3.4. Applications

Korat has been applied and adopted for BIOS testing by world-famous Taiwanese companies, such as ADLINK and Inventec. ADLINK used Korat to record 282 BIOS regression test cases for design verification. Inventec used Korat to record more than 300 test cases in their server mother board manufacturing. Taiwan Semi-conductor Manufacturing Co. (TSMC) mainly use Korat for RPA purposes but we are sorry that the number of Korat scripts is not available to us.

4. Apply computer-vision in GUI automation

In general, the problems and challenges of applying computer vision in GUI automation can be summarized as follows:

- image recognition approaches
- image understanding for higher-level structure
- optical character recognition (OCR).

In theory, if the first and third item can be addressed successfully; that is, accruate and reliable, the CV approaches can at least level up with platform dependent approaches. The second item, on the other hand, can bring explicit benefits over platform dependent approaches. Unfortunately, these challenges still require more research and the difficulties behind the challenges will be thoroughly discussed in this section.

4.1. Image recognition

One basic feature of Korat is to recognize the image areas, such as a button, and then guide mouse cursor to perform operations, such as a click or a drag. Surprisingly, most image recognition can simply be achieved by template matching (TM) with adequate accuracy. In Korat's real, daily applications, reliability is critical. Korat may be scheduled to run hundreds of test cases daily in industry. The system under test may crash or behave abnormally. On the other hand, Korat should restart the system under test and then faithfully replay the remaining test cases. So far, template matching perform well in most of the cases.

The sources images for Korat are screenshots from a computer desktop or a frame grabber which captures individual, digital still frames from an analog video signal or a digital video stream. Frame grabbers are widely used in healthcare, manufacturing, astronomy, etc. In principle, if you capture a frame from the frame grabber, its RGB pixel values should be identical to the screenshot taken by a desktop application. In our experiences, this is nearly true at all time except when hardware components introduce very insignificant interference. The story is in the following:

In one of the Korat IPC applications, a defective VGA cable was used. Instead of producing obvious noises, this defective VGA cable only decreased pixel's RGB values by one or two randomly. Coincidentally, an IPC tester used Korat to test drive a Linux machine and the scenario was to wait for Linux logo in the boot up. So, the tester selected a large image area as a template. From his understanding, larger template contains more information and the probability of recognition failure should be less. Unfortunately, the two coincidences added up to trigger an error in image recognition. It was not straightforward and easy to find out why template matching failed because template and target

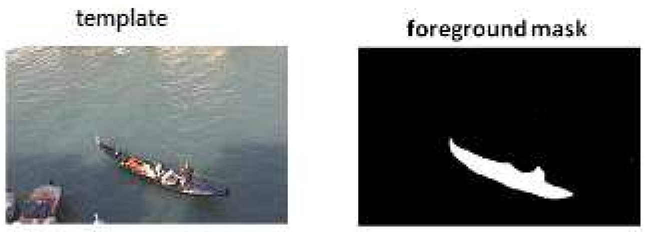


Figure 6. A foreground mask example.

images looked nearly the same on screen. We printed out the binary data of the two images, compared them, and eventually found the root cause.

Straightforward template matching, of course, in several scenarios, no longer works. The followings are some template matching variants that help to deal with the problem.

4.1.1. Foreground mask for template matching

Most computers have different desktop wall paper or background. So, to deal with different background in template matching, a foreground mask in Figure 6 can be produced in Korat. Similar situations also occurs in some fancy web design. This is the case with large background images–using a slide show with a variety of different photographs means that whatever color text you choose it will show up better on some than on others. So, the solution has been to place a filter or transparent layer between the image and the text, meaning that the text will show just as clearly on any image. These transparency and filters are getting more momentum in mobile applications as well. When these GUI rendering techniques are used, it increases a lot of difficulties in computer vision.

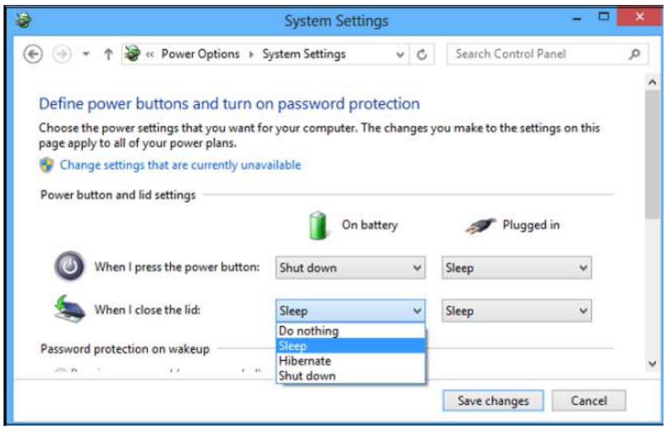
4.1.2. Background color switch

Figure 7 is another case where straightforward template matching fails. The “sleep” item in the listbox was originally in black and white but it changes color after it is clicked to memorize the state. If you want to click the “Sleep” item with black and white template, it no longer works. Foreground mask does not work for this case as well. This problem, of course, can be resolved by OCR, which will be elaborated in later sections. However, it can be simply searched by two templates, one with the black and white and one with the blue background. Korat can use (see Figure 4) two template matching and then union the searched results into a priority queue. Either one should trigger a match.

4.1.3. Anti-aliasing effect

Another interesting case that makes template matching fail is character’s anti-aliasing effect. Figure 8 shows the difference of two magnified images. Apparently, the two templates are still similar but the threshold required to pass the similarity can be as low as 0.5. Again, this problem, in principle can be resolved by a powerful OCR, but it can be resolved by changing template matching parameter. Korat’s default template matching parameter is CV_TM_CCOEFF_NORMED. By changing the template matching parameter to CV_TM_CCORR_NORMED, the anti-aliasing effects can be ignored and two images are determined as similar without lowing the threshold value.

The two methods to compute similarity are listed in Figure 9. CCOEFF matches a template relative



Define power buttons and turn on password protection

Choose the power settings that you want for your computer. The changes you make to the settings on this page apply to all of your power plans.

Change settings that are currently unavailable

Power button and lid settings

	On battery	Plugged in
When I press the power button:	Shut down	Sleep
When I close the lid:	Sleep	Sleep

Do nothing
Sleep
Hibernate
Shut down

Password protection on wakeup

Figure 7. Background color switch.

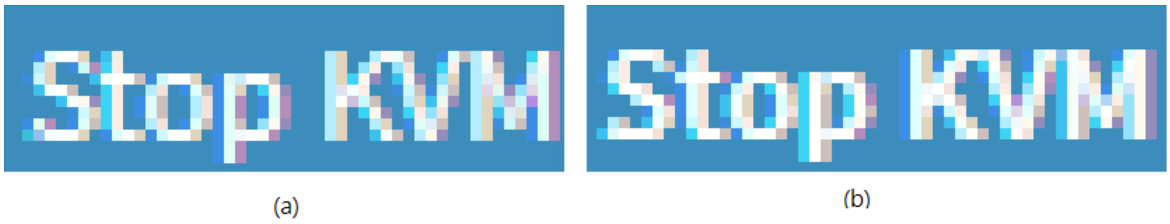


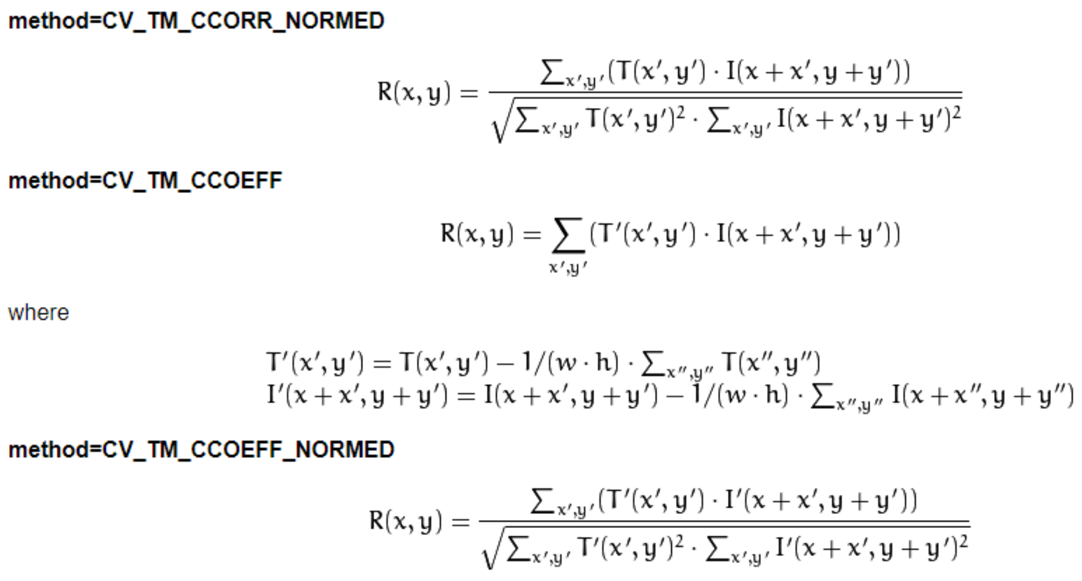
Figure 8. An example of anti-aliasing effects.

to its mean against the image relative to its mean, so a perfect match will be 1 and a perfect mismatch will be -1; a value of 0 simply means that there is no correlation. Comparatively, CCORR simply compute the cross-correlation among the template and target image, which is not as strict as CCOEFF and then resolve the anti-aliasing problem.

4.2. Image understanding

Platform-dependent approaches have their strength, particularly the programming level support to interact with GUI components. However, software applications are eventually rendered as image pixels in the desktop. There are several things can go wrong or unexpected in this process. For example, in web development, the cross browser compatibility testing is an important task for software testers. An implementation of web-based UI can be rendered differently in another browser or OS platform. For example, in browser compatibility check lists, a tester needs to verify the alignment of the elements in a web page; verify the spacing between the elements in a web page; and verify the page layout in different resolutions, etc. An example of layout problems in two different browsers is shown in Figure 10, where a web page is rendered differently in IE9 and Firefox3.7. Similar problems occur in mobile app development as well. It is often inevitable to test applications on real device.

Cross-browser compatibility testing [19] is a practical problem in software industry, where a lot of tools and cloud services have been provided [20]. Browser compatibility testing involves a lot of techniques to address the problem. For example, a web crawler or a test driver, such as Selenium [2] or Appium, must be used to explore the web pages. Then when a web page is loaded, a set of equivalence relations is checked, either by HTML structure or by screenshots. Tools [20] that perform compatibility testing mostly involve taking screenshots while running the web application inside a browser. Checking



method=CV_TM_CCORR_NORMED

$$R(x,y)=\frac{\sum_{x',y'}(T(x',y')\cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}}$$

method=CV_TM_CCOEFF

$$R(x,y)=\sum_{x',y'}(T'(x',y')\cdot I(x+x',y+y'))$$

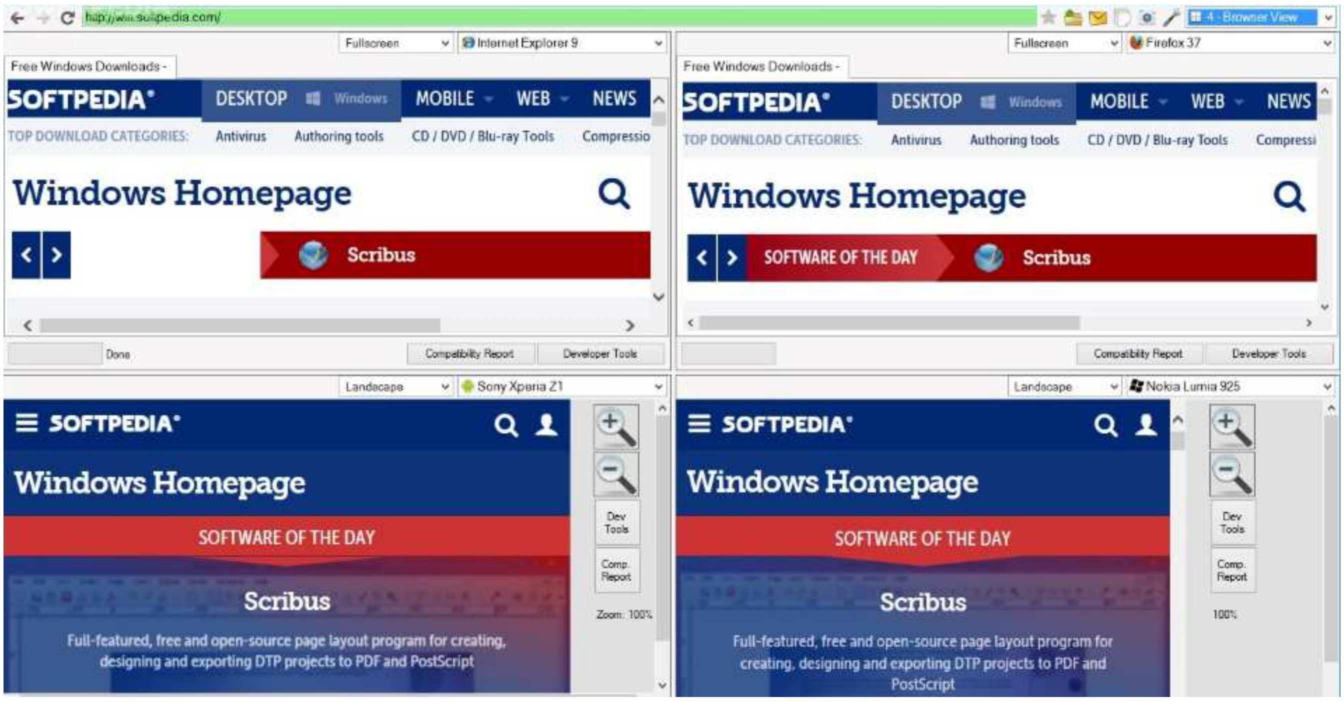
where

$$T'(x',y')=T(x',y')-1/(w\cdot h)\cdot\sum_{x'',y''}T(x'',y'')$$
$$I'(x+x',y+y')=I(x+x',y+y')-1/(w\cdot h)\cdot\sum_{x'',y''}I(x+x'',y+y'')$$

method=CV_TM_CCOEFF_NORMED

$$R(x,y)=\frac{\sum_{x',y'}(T'(x',y')\cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'}T'(x',y')^2\cdot\sum_{x',y'}I'(x+x',y+y')^2}}$$

Figure 9. The cross-correlation vs cross-correlation with co-variances.



Fullscreen — Internet Explorer 9

Fullscreen — Firefox 37

Landscape — Sony Xperia Z1

Landscape — Nokia Lumia 925

Full-featured, free and open-source page layout program for creating, designing and exporting DTP projects to PDF and PostScript

Figure 10. An example of cross browser compatibility testing.

the equivalence relation by computer vision is complicated and requires more advanced technology in image understanding to ensure that a page layout is “equally” or “structurally” the same in different browsers. Currently, browser compatibility testing is beyond the scope of Korat. However, computer vision approaches have an explicit edge over platform-dependent approaches.

4.3. Improve tesseract OCR for GUI automation

One major drawback that makes computer vision approaches less attractive than platform-dependent approaches is numeric and string data acquisition from the system under test. In platform-dependent approach, the numeric and string data can be accessed via GUI component properties, while in computer vision approaches, the data acquisition problem becomes an OCR problem.

Despite OCR techniques have been commercialized by companies such as ABBYY or Google, in our opinions, it remains to be a difficult problem in practice. Compared to recent deep learning achievement in image recognition (e.g., competitions for ImageNet database), OCR still has much room to improve. In practice, an image recognition algorithm which reaches 98% accuracy is considered an achievement, however, a 98% accuracy in OCR can still be classified as unusable in many industrial scenarios.

Although commercial OCR services or products have been available for some time, adopting these techniques can be impeded by different reasons. For example, Google Vision OCR is integrated in Korat as the default OCR engine. Unfortunately, applying Google Vision cloud service requires internet connections, but in Korat’s applications to industry, it is often not allowed to have internet connections in factories. In addition, Korat’s test cases may run in 24 hours. Cost becomes significant and could not be ignored. So, these reasons drove us to seek out open-source solutions as an alternative.

4.3.1. Tesseract to colorful images

Since general-purpose OCR is a difficult subject, Tesseract [21] could probably the only OCR engine that is free for download. The history of Tesseract can be accessed at [21], which can date back to 1990. It was originally an HP lab product and now owned by Google. Since HP had independently-developed page layout analysis technology that was used in products (and therefore not released for open-source) Tesseract never needed its own page layout analysis. Therefore, Tesseract assumes that its input is a binary image with optional polygonal text regions defined.

The latest version of Tesseract uses an end-to-end deep learning-based recognition model [22]. Tesseract takes an image as input and performs page layout analysis to find the regions that contains text. Each region is then segmented into images of individual lines of text. These line images are then fed to the deep learning model for text recognition. More technical details can be found at [22].

To evaluate the feasibility and usability of Tesseract for Korat, we first evaluated Tesseract by many web pages as a beginning. Some of test images are shown in Figure 12 and Figure 13. Note that, there were many more test images evaluated. Here, we select these images to explain the findings and how we improved Tesseract for Korat.

Tesseract performed poorly on several test images. In the first column of Table 1, it lists the character recognition rate by inputting the original image of screenshots. In Table 1, we attach the Google Vision OCR recognition rate as the last column for comparison purposes. Tesseract performed poorly on test images with color background and color characters, particularly in test images like (C) and (E).

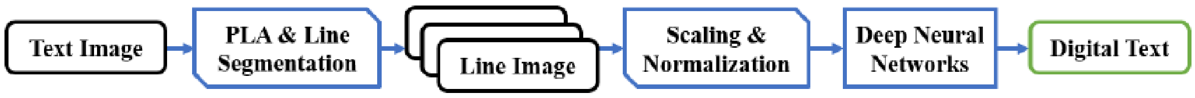


Figure 11. The OCR piple line.

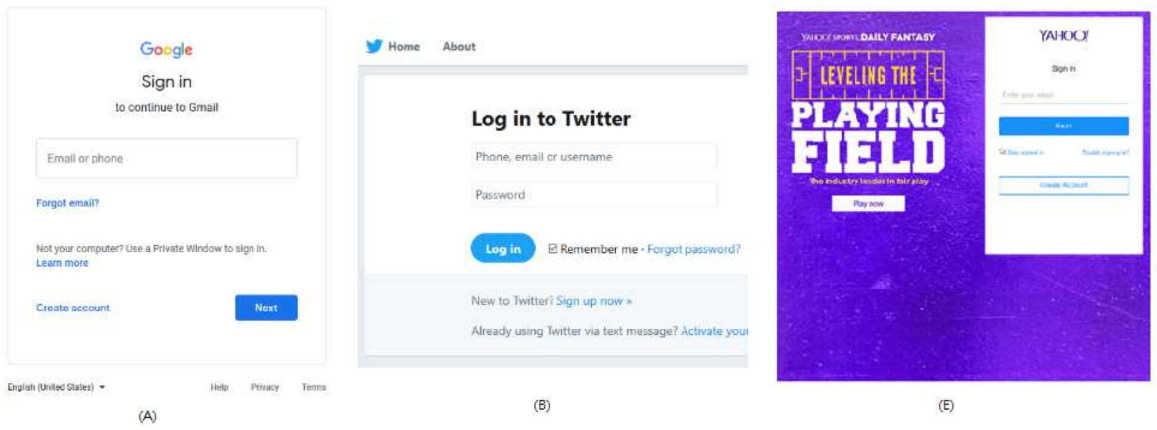

Google
Sign in
to continue to Gmail
Email or phone
Forgot email?
Not your computer? Use a Private Window to sign in.
Learn more
Create account
Next
English (United States)
Help
Privacy
Terms
(A)
Home
About
Log in to Twitter
Phone, email or username
Password
Log in
Remember me · Forgot password?
New to Twitter? Sign up now »
Already using Twitter via text message? Activate your
(B)
YAHOO! SPORTS DAILY FANTASY
LEVELING THE PLAYING FIELD
The industry leader in fair play
Play now
YAHOO!
Sign in
Create Account
(E)

Figure 12. Test images of (A)(B)(E).

In Tesseract's official website, there is information on tools and procedures to train the neural nets engine of Tesseact. The first thought to raise the recognition rate is training the Tesseract's neural net engine in iterations to correct the failed characters. Unfortunately, the recognition rate was nearly the same. This interesting results pushed us to find a answer. So, we explored and studied Tesseract's related publications and documentations.

According to [22], Tesseract implemented a pipeline shown in Figure 11. In this pipeline, Tesseract first performs page layout analysis (PLA) to detect the text in the image and segments the image into sub-images containing one line of text each. Then, each line image is scaled and normalized to match the training data of the recognition model to output the combined text prediction. Tesseract used 4 long-short-term memory (LSTM) layers with, respectively, 64,96,96, and 512 hidden units. This explains why its recognition model output the "combined" text prediction as compared to character-based recognition in the legacy version of Tesseract.

In our understanding of Tesseract history, the legacy Tesseract was designed to analyze a binary document. So, its latest-version page layout analysis and segmentation could be designed to process a binary document only, where background color is white and foreground color is black. These observations could explain why training did not increase the recognition rate. Our first improvement plan is to binarize the image before inputting to Tesseract.

Several image thresholding algorithms can be used to binarize the image automatically. One candidate we choose is Otsu thresholding [23] from opencv. The results of applying Otsu binarization is shown in the second column of Table 1 and Table 2. The results are nearly as poor as Tesseract. The recognition rate are higher in test image (C) and (E), but the increases in the recognition rate has little impact to the usability. Figure 14(A) explains why Otsu algorithm failed. At first glance, the image thresholding performed by Otsu algorithm is actually very good. However, the algorithm cannot tell

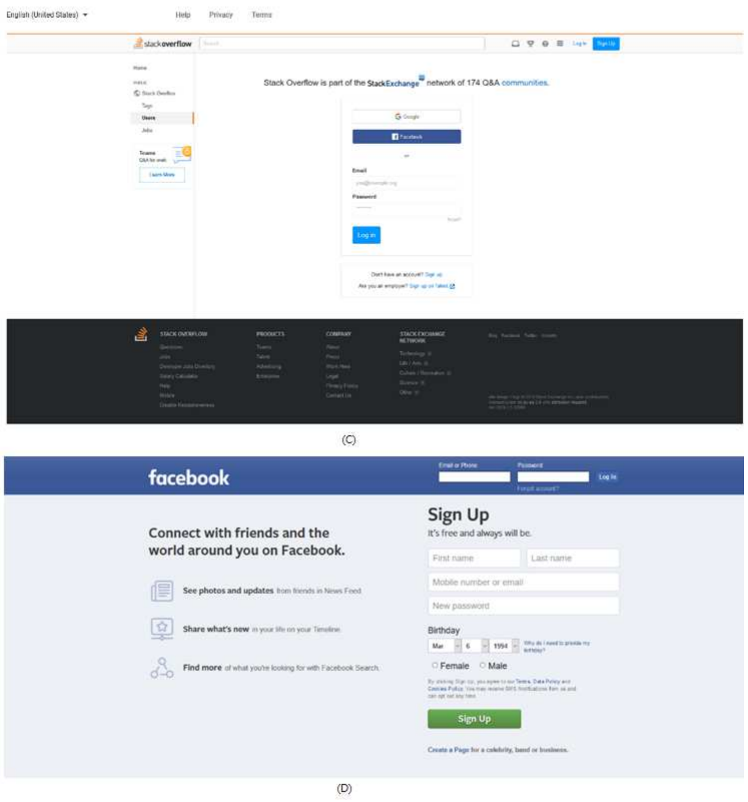

English (United States)
Help
Privacy
Terms
stackoverflow
Log in
Sign Up
Stack Overflow is part of the StackExchange network of 174 Q&A communities.
Google
Facebook
Email
Password
Log in
Don't have an account? Sign up
STACK OVERFLOW
PRODUCTS
COMPANY
STACK EXCHANGE NETWORK
(C)
facebook
Email or Phone
Password
Log In
Forgot account?
Sign Up
It's free and always will be.
Connect with friends and the world around you on Facebook.
See photos and updates from friends in News Feed.
Share what's new in your life on your Timeline.
Find more of what you're looking for with Facebook Search.
First name
Last name
Mobile number or email
New password
Birthday
Mar
6
1994
Female
Male
Sign Up
Create a Page for a celebrity, band or business.
(D)

Figure 13. Test images (C)(D).

the foreground pixels and background pixels in the context of text binarization. We are looking for some binarization method which could produce the results in Figure 14(B), where the pixels of texts should be drawn as black pixels regardless of its color and background.

To achieve the text binarzation results in Figure 14(B), we applied the font and background color independent (FBCITextBin) algorithm from [24]. We use the algorithm to preprocess the screenshots for Tesseract. The result (3rd column in Table 1) is a significant increase in recognition rate, nearly 50% improvement.

Table 1. Character recognition rate by different binarization algorithms.

	Terrseract	Otsu Bin	FBCITextBin	Google
Google Login page (A)	87.97	87.34	98.12	100.0
Twitter login page (B)	96.51	98.25	100.0	100.0
stack overflow (C)	23.33	34.56	81.38	98.67
Facebook (D)	91.72	91.37	91.95	98.62
Yahoo (E)	22.09	31.97	85.46	97.67
Average	64.32	68.69	91.38	98.99

Table 2. Character recognition rate after training.

	Tesseract (Trained)	Otsu Bin (trained)	FBCITextBin (trained)	Google
Google Login page (A)	89.24	87.97	99.36	100.0
Twitter login page (B)	97.67	98.25	100.0	100.0
stack overflow (C)	23.33	34.56	80.79	98.67
Facebook (D)	92.52	90.57	91.83	98.62
Yahoo (E)	22.25	31.97	85.88	97.67
Average	65.00	68.66	91.57	98.99

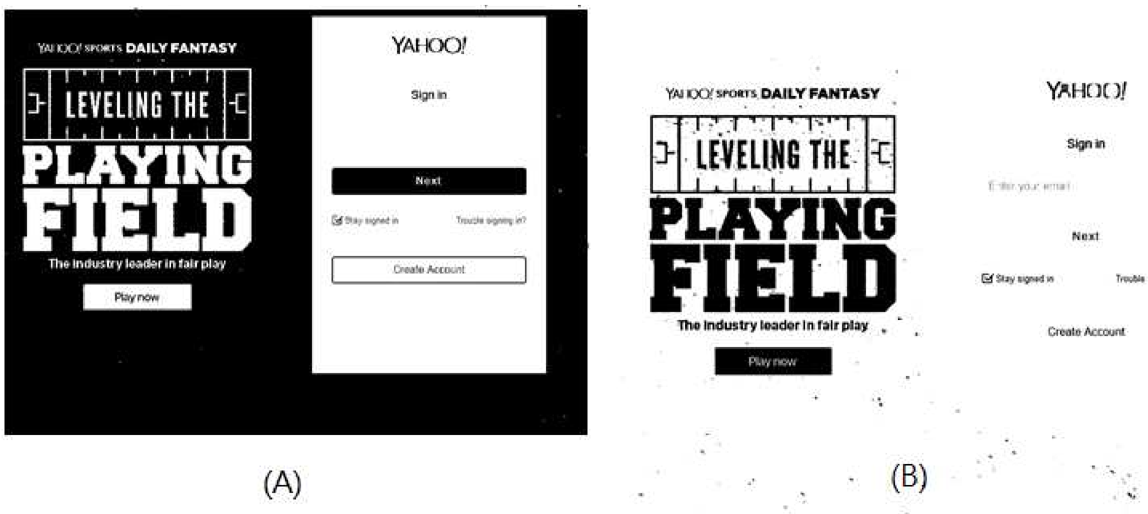


Figure 14. (A) The binarization using Otsu algorithm. (B) the binarization using text binarization approach.

4.3.2. Text binarization with training

Even though the recognition rate has increased significantly, 80% recognition rate can be unusable in practical applications. We further investigated and studied the characters which are not recognized correctly. The pixels of these characters are damaged by text binarization. Some of these damages are shown in Figure 15. So, there is much room to improve in the text binarization algorithms. The FBCITextBin uses Canny edge detection to find the contour of a character but it fails to produce stable and good results in many cases due to the anti-aliasing effects. These failures unfortunately cannot be completely overcome by training in the neural net engine. In Table 2, training could not raise the recognition rate further more because some of the characters are already poorly recognizable after text binarization.

Overall, these research results show that Tesseract OCR for screenshots in GUI automation can be improved significantly for practical usage in some domains. However, general OCR that are capable of dealing with arbitrary input remains to be difficult. Even Google Vision OCR may fail on some obvious places. For example, Google Vision OCR cannot recognize the world "FIELD" in test image (E), while our method can. OCR will always be the Achilles heel for several computer vision problems.

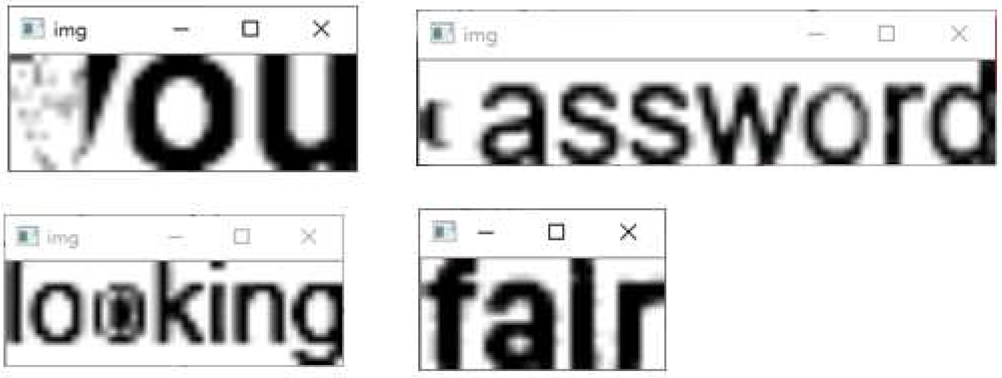


Figure 15. The damaged character pixels.

5. Conclusions

In this paper, we describe two major approaches of GUI automation techniques. While platform dependent approaches are favored by more commercial products, computer vision approaches have their unique merits such as cross platform capability and no need to keep up with platform's technology advances. Computer vision approaches are feasible on application domains like compatibility testing, games, etc., whereas platform dependent approaches are not. However, computer vision approaches introduce more difficulties to the GUI automation problems which require more computer vision research or technology breakthrough before having an edge over platform dependent approaches.

As pointed out in the paper, among the computer vision problems, OCR is probably the most critical, essential, and difficult problem to deal with. Other computer vision problems are engineering problems and are comparatively solvable by ordinary image analysis techniques. We believe commercial services such as Google vision OCR may become better and better by big data but there are problems as always. For example, Google Vision OCR cannot recognize the world "FIELD" in test image (E), while our method can. Page layout analysis, preprocessing, and segmentation remain to play a key role in OCR. A powerful recognition engine such as deep neural nets can be implicitly compromised by the processing quality in previous stages.

In this paper, we improved the recognition rate from nearly 25% to nearly 85% (see test images (C)(E)) by introducing text binarization as a key preprocessing stage, which shows that we could be in a correct path. Currently, we are investigating and improving algorithms to raise the recognition rate to 98%. In our evaluation, the text binarization algorithm [24] are not stable and failed in many cases. According to our analysis, these cases are resolvable. In other words, we aim to build a page layout analysis and segmentation preprocessing stage specifically tailored for color desktop screenshots.

Acknowledgments

This research was partially supported by Ministry of Science and Technology, TAIWAN, under Grant No. 107-2218-E-008 -002. We would also like to show our gratitude to anonymous reviewers for their so-called insights and comments, which greatly help the final version of this manuscript.

Conflict of interest

All authors declare no conflicts of interest in this paper.

References

1. B. N. Nguyen and A. M. Memon, An observe-model-exercise* paradigm to test event-driven systems with undetermined input spaces, *IEEE T. Software Eng.*, **40** (2014), 216–234.
2. Selenium Projects, Seleniumhq, browser automation. Available from `http://www.seleniumhq.org/`, July 2019.
3. Hewlett-Packard Inc., Hp quicktest professional software. Available from `https://download.cnet.com/HP-QuickTest-Professional/3000-2383_4-10969380.html`.
4. Microsoft, Codedui test, microsoft ui automation (uia). Available from `https://en.wikipedia.org/wiki/Microsoft_UI_Automation`.
5. EOSS Group, Ranorex, test automation for everyone. Available from `https://www.ranorex.com/`.
6. Telerik, Test studio, automated testing made easy. Available from `http://www.telerik.com/teststudio`.
7. UIPath Ltd., UIPath , Accelerate Human Achievement . Available from `https://www.uipath.com`.
8. T. Yeh, T. Chang and R. C. Miller, Sikuli: using GUI screenshots for search and automation, in Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology, Victoria, BC, Canada, October 4-7, 2009 (A. D. Wilson and F. Guimbretière, eds.), *ACM*, (2009), 183–192 .
9. T-Plan, T-plan robot. Available from `http://www.t-plan.com/robot/`.
10. TestPlant, eggplant test automation tools. Available from `https://www.testplant.com/eggplant/testing-tools/`.
11. Y. Cheng, J. W. Kuo, B. Cheng, et al., platform-independent capture/replay test automation system," in 17th IEEE International Conference on High Performance Computing and Communications, HPCC 2015, 7th IEEE International Symposium on Cyberspace Safety and Security, CSS 2015, and 12th IEEE International Conference on Embedded Software and Systems, ICESS 2015, New York, NY, USA, *IEEE*, (2015), 1122–1127.
12. ABBYY Co. Ltd., ABBYY FindReader. Available from `https://www.abbyy.com/en-eu/`.
13. Google Inc., Cloud vision. Available from `https://cloud.google.com/vision/`.
14. HP labs, Tesseract-ocr. Available from `https://code.google.com/p/tesseract-ocr/`.
15. G. Neszaros, xUnit Test Patterns: Refactoring Test Code. Addison-Wesley, 2007.
16. IBM Inc., Rational robot. Available from `http://www-01.ibm.com/software/awdtools/tester/robot/index.html`.
17. H. Zhu, W. K. Chan, C. J. Budnik, et al., The 5th Workshop on Automation of Software Test, AST 2010, May 3-4, 2010, Cape Town, South Africa, *ACM*, (2010).

18. C. Hsueh, Y. Cheng and W. Pan, Intrusive test automation with failed test case clustering, in 18th Asia Pacific Software Engineering Conference, APSEC 2011, Ho Chi Minh, Vietnam, December 5-8, 2011 (T. D. Thu and K. R. P. H. Leung, eds.), *IEEE* Computer Society, (2011), 89–96.
19. A. Mesbah and M. R. Prasad, Automated cross-browser compatibility testing, in *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, (New York, NY, USA), *ACM*, (2011), 561–570.
20. Software Testing Help, Top 10 cross browser testing tools in 2019 (latest ranking). Available from `https://www.softwaretestinghelp.com/best-cross-browser-testing-tools-to-ease-your-browser-compatibility-testing-efforts/`.
21. R. Smith, An overview of the tesseract ocr engine, in Ninth International Conference on Document Analysis and Recognition (ICDAR 2007), **2** (2007), 629–633.
22. C. Song and V. Shmatikov, Fooling OCR systems with adversarial text images, *Computing Research Repository*, abs/1802.05385, 2018.
23. J. Zhang and J. Hu, Image segmentation based on 2d otsu method with histogram analysis, in 2008 International Conference on Computer Science and Software Engineering, **6** (2009), 105–108.
24. T. Kasar, J. Kumar and A. G. Ramakrishnan, Font and background color independent text binarization, in In Proceedings of 2nd International Workshop on Camera Based Document Analysis and Recognition, (2007), 3–9.