Hidden Object Detection for Computer Vision Based Test Automation System

RAJIBUL ANAM, MD. SYEFUL ISLAM, MOHAMMAD OBAIDUL HAQUE
Samsung R&D Institute Bangladesh
BANGLADESH
rajibul.an@samsung.com, syeful.islam@samsung.com, obaidul.h@samsung.com

Abstract: In Software Quality Assurance, computer vision based automation tools are used to test the window based application and window contains many type of objects like button, box, list, etc. Automation tool detect window objects by comparing images. Most of the objects are visible in the screen but some are not visible to the screen at the first time, proper interaction with the window application hidden objects get visible to the screen like drop-down list item, editor text object, list box item and slider. With the vision based automation systems these hidden objects cannot be searched directly. In this paper proposes some methods which use image and shortcut key to interact with the testing software to search the hidden objects. These methods will enhance the automation tools to access the window application hidden objects faster.

Key-Words: - test case automation; software quality assurance; vison based; window application

1 Introduction

Software Quality Assurance (QA) is one of the critical areas of software development process life cycle. After co-work with developer and designers, QA ensures the correctness of the operation by testing the software application through different type of test cases [1]. Many methods have been used to test the software and among them Black Box and White Box Testing are very commonly used. Black Box testing consists of specification and experience based testing, which checks the entire software operation [1-2]. White Box testing follows the structure based testing, which checks the software process flow [1-2]. QA testing actions or steps are executed by mouse and key events, after the events, program flows and interfaces get change [2], which is the part of the QA testing. Manual testing operates by human, it executes series of steps and check for the specific output which has chances of error [1],[3]. But Automation System executes series of steps according to the code instruction, which executes test steps faster than human and less error [3]. Moreover, automation system has been used for Black Box testing because it follows specific test steps and expects for target results. Most of the testing application needs to be tested with the predefined Test Cases (TC). These predefined TCs can be automated, so the testing will be faster and human dependency will be reduced.

Graphical User Interface (GUI) QA testing purpose many type of automation system have been used such as Pesto [4], DEVSimPy [5], Watir [1], Selenium [7], Sikuli [6],[8]. These systems either use the vision based or screen objects position pointing technique to track the screen objects. Script re-usability and smooth execution are essential for the automation system [6]. Automation system executes QA testing steps/actions easily by tracing image and objects position. GUI applications has hidden or not visible objects like text objects in the editor, drop-down list object, multi tab scroll object and slider. These types of hidden objects can be searched easily by manual QA testing. But vision based systems firstly; trace objects (image) and secondly, execute action events on the screen object position. Vision based system uses only image based object detection method, which uses only mouse events to interact with the GUI testing application therefore, complex steps like search hidden object from a list box or slider scrolling takes time and sometimes failed to trace the target hidden object. Considering these difficulties, focuses on how to access hidden objects accurately and enhance re-usability.

This paper proposes some methods that will enhance vision based automation tools to discover hidden objects (GUI elements) from the GUI based applications by using key and mouse events. Vision based system uses only mouse events therefore, include the key events (shortcut key additional feature) to trace the hidden object easily and accurately. The propose methods use shortcut key

and click (image object to click on the GUI application) for action events, then interact with the visible objects through few steps, afterwards the hidden object gets visible on the screen. The proposed methods will enhance the vision based automation systems to search the hidden objects faster.

This paper is arranged as follows. Section 2 provides brief review of other automation tools for GUI testing. Section 3 describes about the proposed solution. Section 4 details of the usability study and finally conclusion is on section 5.

2 Related Works

Software QA testing automation systems has been used to reduce human effort. Automation systems trace the target objects by screen object position, image matching and source name. Automation actions/steps are executes by click, drag-and-drop and keyboard events. Tools like Sikuli, Robot and Pesto uses image tracing, object source name and object screen position to access the object. These automation systems developed either on vision based or screen objects position detection based methods. Below discuss details of these two types of automation systems.

2.1 Vision Based Automation

Sikuli is an open source GUI vision based automation system, which searches the target object using screenshot [8-10]. The IDE permits users to take a screenshot of the target object (GUI elements) such as button, icon, dialog box and run time detects the object to direct the mouse and keyboard events [11-13]. Figure 1 illustrates the Sikuli Framework, where built-in modules are available like find, click and key events [14]. There are more modules available which cannot be used from IDE directly. It has the Application Programming Interface (API) for testing and developing the library. It is a platform independent framework.

Robot Framework is a generic testing automation system to test Acceptance Test-Driven Development (ATDD) [15]. ATDD is a process where developers and testers discuss the demands required by the customers to come with the acceptance test before development. The acceptance test provides the functional importance of the software [15].

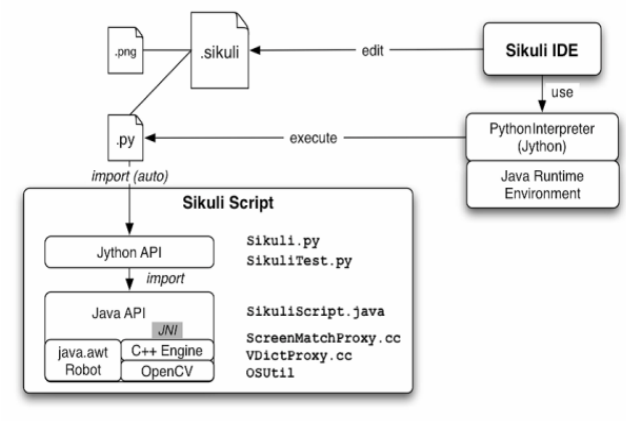

Figure 1. Sikuli Framework

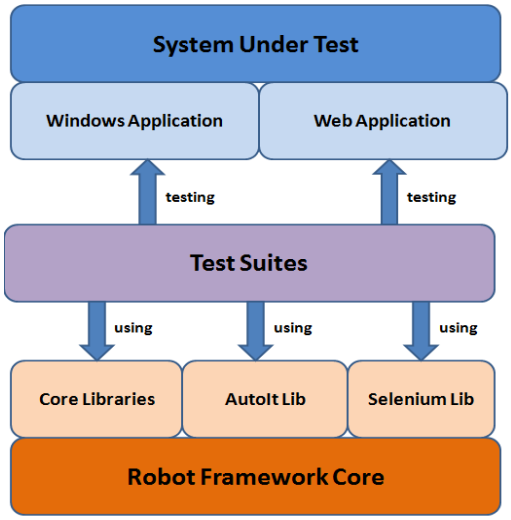

Figure 2. Robot Framework

The aim of the acceptance tests is to justify the requirements by providing examples for each test. The examples can be tested to prove compliance. The script language is written using plain English natural commands called Keywords [9]. Keywords are common like methods in programming language. Natural command keywords make the tests more readable and easy to understand even for non-coders. This framework script writing is extended to Python (can run also on both Ironpython and Jython) or java. The developers can use the existing syntax to create the script or can create own syntax. Robot framework uses for GUI testing and system resource management, but only java based software can be tested. It generates auto report of the testing as html and text format. It has the API for testing and developing the library.

2.2 Screen Objects Position Detection Based Automation System

The TestComplete [16], TestPlant [17] and Squish [18] are recoding based framework. These systems

record user interaction events according to the screen object position and replay time these systems execute the events according to the recorded sequence. Replay time these systems execute the TC very fast and generate a report of the TC. These systems use their own language to generate script and developers can edit the auto generated scripts. Recording time these systems track the user events position like mouse click position, drag-and-drop positions and keyboard input information. If the testing software font, color gets change then the recorded script can be replay unless the application layout has changes. Moreover recording and replay (execution) time the screen size (resolution) should be same because all the action events execute according to the objects position. If the app window or pop-up window position gets change on the screen then replay time generate errors. The application layout should be same at recording and replay time.

3 Proposed Solution

Various types of automation systems are available and most of it uses vision based algorithm to search the GUI target objects. These systems take the screenshot of the window first; then select the target object from the screenshot and interact with the application by mouse or key events. These systems are used to search the GUI window target objects like toolbar button, menu item, icon and dialog box [14]. Moreover, searching arbitrary depends on the screenshot and the target object image. If the target objects image do not matches with the screenshot image, then automation system could not search the target object in the screen. In this case system searches for an object which is not visible (hidden object) in the screen or the object is not available in the testing software. But the hidden object needs to be search because it is available in the testing software and part of the TC. The automation system would not be able to search the target object until the target object gets visible on the screen. Current approaches required entering an image as query to search the target object. If searching for a hidden object in the window screen and could not trace the target object then it will generate an error, which is a limitation of the vision based automation system.

The proposed method searches hidden objects like item in the editor, drop-down list object, multi tab scroll object and slider positions. In addition shortcut key events has introduced instead of screens object image to trace the target object. The Editor Scroll-bar Object Selection method uses to search the hidden object from a scroll-bar affiliated object. The Drop-down List Object Selection method is applicable to search hidden object from the drop-down list. The Multi Tab List Object Selection method is valid to search the hidden object from the multi tab list box and Slider Positon Selection method is applicable to search and puts the slider position according to the code instruction. Below sections discuss details of the proposed methods which uses mouse and key (shortcut key) events.

3.1 Editor Scroll-bar Object Selection (ESOS)

QA testing time automation system needs to check the text editor or webpage interface and font decoration objects. It becomes very hard to find text object in the editor which contains a long page and the target text object stand at the end of the page, at this scenario the stroller get enabled. But hidden objects do not appear on the screen and could be visible unless the system searches for the hidden objects [14]. Figure 3 shows the editor screen with scroll-bar object, where a text editor is opened. The automation system needs to search the figure 4 hidden target object (search in the screen) in the (figure 3) text editor. In this scenario the automation system needs to scroll down the scroll-bar using the mouse [13]. There is no specific method for the scroll-bar to scroll down at specific point. To solve this problem, proposed the ESOS method where the scroll-bar will be scroll down until it reaches the target object.

Figure 5 line 1-2 searches the *mainobject* (the style.css object) and take focus on the style.css object. Line 3 puts the cursor at the beginning of the editor. Line 4-8 searches for the *targetobject* (figure 4), if it does not find *targetobject*, then goes to the next line until it reaches to the *targetobject* (figure 4). If the *targetobject* found then select (click) the hidden target object. The scroll-bar cannot be used directly (can access it but cannot scroll it as requires) and with this method automation system can search the hidden object from the screen without scrolling the scroll-bar. Figure 6 shows where the hidden object gets visible and hidden target object is found by using this method.

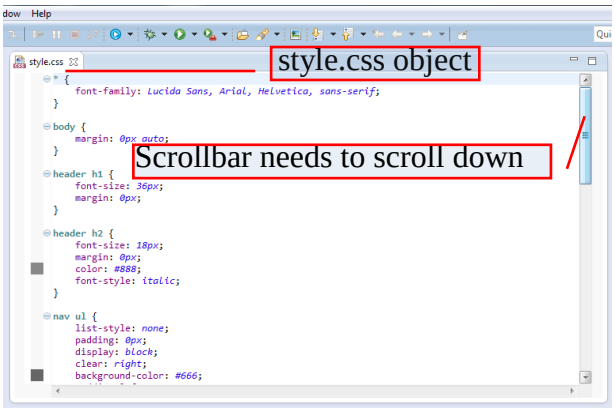

Figure 3. Screenshots of a Scrolling Object

```
color: #888;
margin-top: 24px;
```

Figure 4. Hidden Target Object

Input:
mainobject is an image or text object to focus on the object;
targetobject is an image object;
Output:
targetobject get selected;
Variables:
screenimage is the desktop screen capture image;
onelinedown is an keyboard value to move down the cursor next line;

ScrollbarObjectSelection(*mainobject*, *targetobject***)**
1. **If** *mainobject* matched with *screenimage* **Then**
2. Click on the *mainobject*;
3. Put cursor to the beginning of the editor;
4. **While** until found the *targerobject*
5. Move the cursor *onelinedown*;
6. **If** *targetobject* matched with the *screenimage* **Then**
7. Click the *targetobject* in the screen;
8. Break;
9. **Else** cannot found the *targetobject*;
10. **End**

Figure 5. Editor Scrollbar Object Selection Method

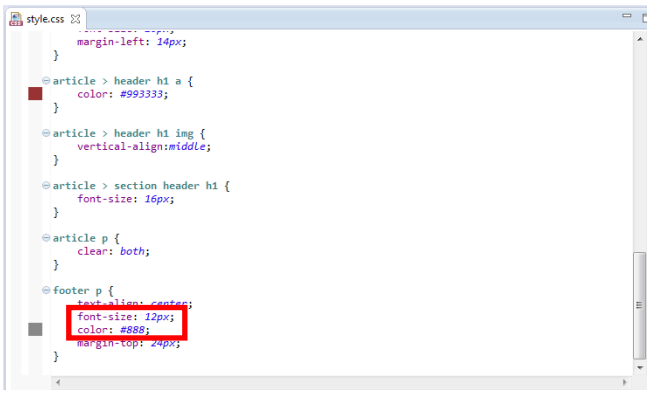

Figure 6. After Scrolling the Screen

3.2 Drop-down List Object Selection (DLOS)

Drop-down list objects are used in the window and web based applications. There is some drop-down list which contains text with images. Most of the drop-down list contains long item lists which need to check and test for the QA. If drop-down item list is long, then most of the items will not be visible on the screen [14] which creates hidden object in the list. But drop-down list hidden object items cannot be access properly by the automation system because the scroll-bar appears dynamically and needs to scroll it to get the hidden object. The proposed DLOS method will enhance the automation system to search the hidden objects from the drop-down list.

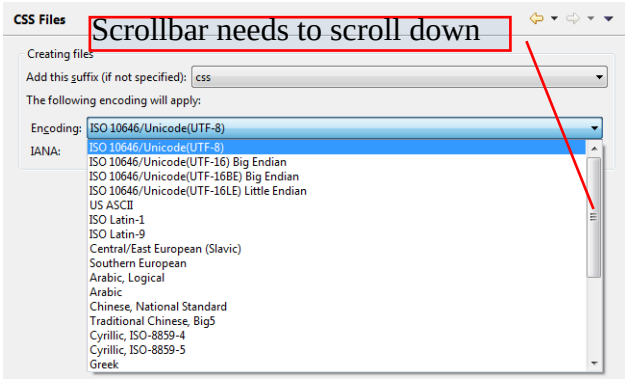

Figure 7. Screenshot of Dropdown List

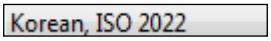

Figure 8. Hidden Target Object

Input:
shortcutkey is keyboard value to take focus of the object;
targetobject is an image object;
Output:
targetobject get selected;
Variables:
screenimage is the desktop screen capture image;
onelinedown is an keyboard value to move down the cursor next line;

DropdownListObjectSelection(*shortcutkey*, *targetobject***)**
1. **If** *shortcutkey* works to select the object **Then**
2. **While** until the *targetobject*
3. Move the cursor *onelinedown*;
4. **If** *targetobject* matched with *screenimage* **Then**
5. Click the *targetobject* in the screen;
6. Break;
7. **Else** cannot found the *targetobject*;
8. **End**

Figure 9. Dropdown List Object Selection Method

Figure 7 shows the drop-down list with long item lists which contains hidden object. Figure 8 shows the hidden target object which needs to search from the drop-down list. Figure 9 shows the DLOS algorithm where line 1 uses *shortcutkey* to select the drop-down list object. Line 2-6 searches for the *targetobject* from the list, if not found *targetobject* then goes to the next list item until it reaches to the *targetobject*. Figure 10 shows the *targetobject*

matched with *screenimage*. This method searches the entire hidden objects from the list and checks for the target object.

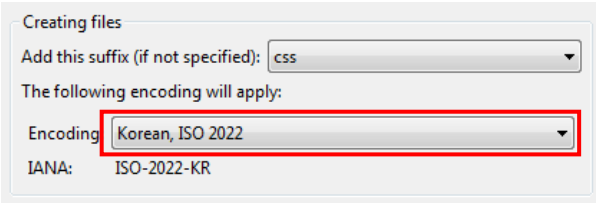

Figure 10. Screenshot of the Dropdown List with Target Object

3.3 Multi Tab List Object Selection (MTLOS)

QA testing time needs to interact with GUI window multi tab objects [13]. A multi tab window contains more than one list box objects with scroll-bar features. List box object contain many hidden objects, automation system needs to interact with the hidden objects to complete the TC. Figure 11 shows an example of the multi tab objects which contains three tab objects. Figure 11 (1) shows the first tab object (sample), figure 11 (2) shows the second tab object (Web App) with list box, figure 11 (3) shows the another tab object with hidden (item list) and figure 11 (4) scroll-bar enabled for scrolling.

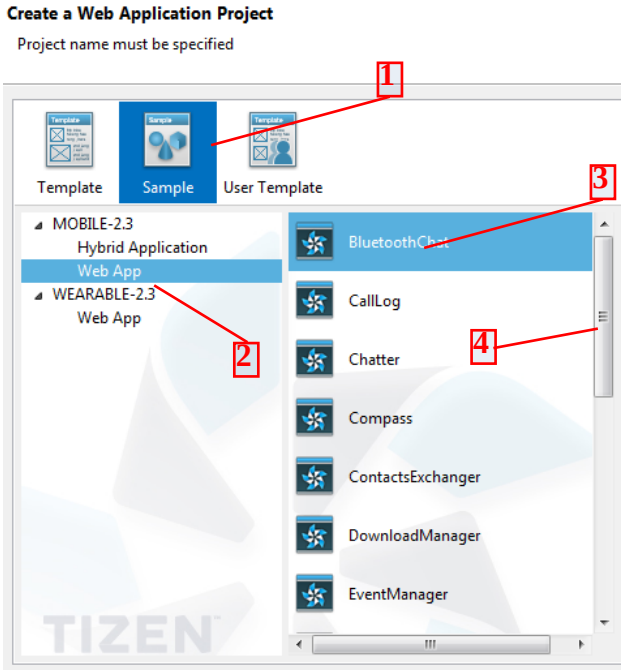

Figure 11. Multi Tab List Object Screenshot

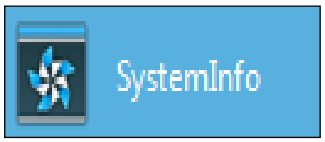

Figure 12. Hidden Target Object

To search the target object (figure 12) from figure 11, firstly needs to select the (1) sample object, secondly select the (2) web app object from the list box and finally searches for the target object from the (3) next list box.

The proposed MTLOS method is able to search target object from multi tab list-box hidden object. Figure 13 shows the MTLOS method, where line 1-2 selects the mainobject (figure 11, object 1), line 3-4 selects the next tab object and line 5-9 searches (figure 11, object 2) for the firstkeyinfo image object until it found. Line 10-11 selects next tab object (figure 11, object 3), line 12-16 search for the tergetobject (figure 12) until it found. Figure 14 shows the target object found using this method.

Input:
mainobject is an image object;
firsttab is a keyboard tab value;
firstkeyinfo is an image object;
secondtab is a keyboard tab value;
targetobject is an image object;
Output:
targetobject get selected;
Variables:
screenimage is the desktop screen capture image;
onelinedown is an action variable to move down the cursor next line;

Multitabobjectselection(*mainobject, firsttab, firstkeyinfo, secondtab, targetobject***)**

```
1.  If mainobject matched with screenimage Then
2.    Click on the mainobject;
3.    If firsttab is true Then
4.      Press tabkey;
5.      While until firstkeyinfo
6.        Move the cursor onelinedown;
7.        If firstkeyinfo matched with screenimage Then
8.          Click the firstkeyinfo in the screen;
9.          Break;
10.   If secondtab is true Then
11.     Press tabkey;
12.     While until targetobject
13.       Move the cursor onelinedown;
14.       If targetobject matched with screenimage Then
15.         Click the targetobject in the screen;
16.         Break;
17.   Else cannot found the targetobject;
18.   End
```

Figure 13. Multi Tab List Object Selection Method

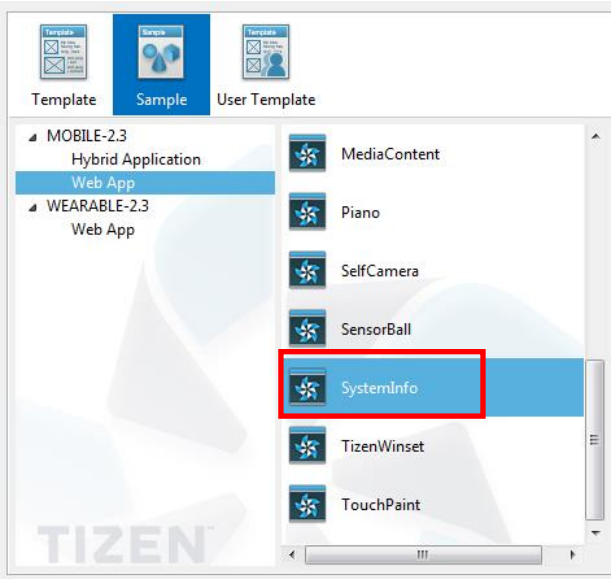

Figure 14. Screenshot of Multi Tab List Object with Target Object

3.4 Slider Position Selection (SPS)

Slider is a GUI window application object which has no onscreen values (from where to drag and drop) like scroll-bar. QA purpose slider needs to access, change slider positions and checks the expected result. But for the QA testing purpose automation tool needs to access and change the value of the slider [13] which takes time and sometimes automation system failed to change the slider positions as requires. Figure 15 shows the slider where it is at the Error mode and testing purpose needs to set as Debug mode (figure 16). While TC execution, automation system cannot put the slider as required position easily, sometimes it starts scrolling on the left and sometimes on the right side. As a result it takes additional time to reach to the goal. The proposed SPS method can overcome this problem and put the slider position according to the code instruction easily.

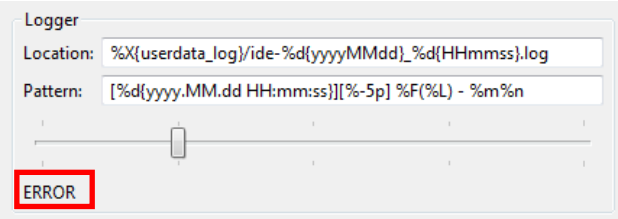

Figure 15. Screenshots of Slider Object

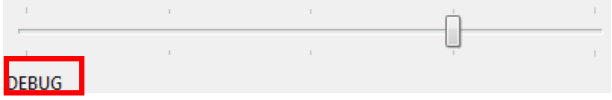

Figure 16. Target Object

Input:
mainobject is an image object;
keyinfo is a keyboard key move value;
targetobject is an image object;
Output:
targetobject get selected;
Variables:
screenimage is the desktop screen capture image;
onelinedown is an action variable to move down the cursor next line;

scrollslider(*mainobject, keyinfo, targetobject***)**

```
1.  If mainobject matched with screenimage Then
2.    Click on the mainobject;
3.    Scroll slider to lowest value;
4.    While until targetobject
5.      Press keyinfo;
6.      If targetobject matched with screenimage Then
7.        targetobject object found;
8.        Break;
9.  Else cannot found the targetobject;
10. End
```

Figure 17. Slider Positon Selection Method

Figure 18. Slider Position with Minimum Value

Figure 17 illustrates the SPS methods, where line 1-2 take focus on the *sliderobject*, line 3 takes the slider lowest sliding position (figure 18). Line 4-8 increases the slider positions according to the *keyinfo* value until it reaches to the *targetobject*. With this method the slider hidden objects value can be search faster.

3.5 Framework Comparisons

The Sikuli [9] is a vision based automation framework, it uses image to detect the object and after that use the action events to interact with the testing software. The TestComplete [16], TestPlant [17] and Squish [18] are screen objects position detection based framework, all the mouse and key events are applied on specific point of window screen object position. If the object position gets change (screen resolution) or mismatched then select or interact with different objects and generate error. Table 1 shows the comparison criteria of automation frameworks.

Screen objects position detection based framework can access the hidden objects, if the screen window objects position remains fixed on the second run time. But TC execution time it is very hard to confirm the window objects position. Developer record the test, execution time if the window object appears at different position, then developer needs record the steps again which is

redundant. Table 2 illustrates the frameworks, if the screen window object position or the resolution gets change, then cannot interact with the onscreen window objects to search the hidden objects. But same time the proposed methods are able to interact with the changed screen (position or resolution) window objects. The proposed methods uses key and (image objects) click events to interact with the window objects to search the target hidden objects.

Table 1. Comparison of Automation Framework Criteria

	Sikuli	TestComplete	TestPlant	Squish
Open Source	Yes	No	No	No
App code required	Yes	Yes	Yes	Yes
Platform Independent	Yes	No	Yes	Yes
Hidden Object Identification	No	No	Yes	Yes
Image Based	Yes	No	No	No
Screen Position Dependent	No	Yes	Yes	Yes
Test Recording	No	Yes	Yes	Yes

Table 2. Comparison of Framework With Screen Object Position

Criteria		Sikuli	TestComplete	TestPlant	Squish	Propose Methods
Changed Screen Position	**Hidden Object Identification**	No	No	No	No	Yes
	Slider Interaction	No	No	No	No	Yes

3.6 Complexity Comparisons

The time complexity depends on flow of the algorithm [19]. If the algorithm uses nested operation then the complexity gets higher. Below table 3 shows the comparison of the Proposed Algorithm (PA) and existing Vision Based Algorithm (VA), where O denotes as growth of a function and n is number of steps. It is clear that VA and PA time complexity are almost same. There is no significant difference between PA (ESOS, DLOS, MTLOS and SPS) and VA. But there are differences on the execution time because of the dependency (wait for the object, interaction methods).

Table 3. Complexity of the Algorithms

Time Complexity of VA	Time Complexity PA
O(n) [14]	*O(n)* ESOS
O(n) [14]	*O(n)* DLOS
O(n) [14]	*O(n)* MTLOS
O(n) [14]	*O(n)* SPS

4 Usability Study

GUI Automation system executes action according to the instructions (code). Basically two types of event occur in the GUI automation, one is key event and another is mouse event. Automation system runs the code; execute commands which interact with GUI testing system. To generate mouse or key events, screen objects position detection based systems record the user actions and automatic generates code for automation system. And image based systems do not have this facility, developer needs to write code.

4.1 Case Study Design

This section describes the experimental results obtained by the VA and PA with four predefined Test Cases. The empirical study presented in this paper is conducted in real time context. This paper proposed four methods which uses vision based methods and shortcut key to access the object, which is a combination of image and key events. Moreover these methods will enhance the automation system to get the target object faster. To support this claim carried out a case study to test the hypothesis below.

H1: Using shortcut key (key event) and vision based screen object detection (click) to search the target object reduces automation systems interaction events than using only vision based object detection (click).

H2: Combination of vision based screen object detection and shortcut key can trace the hidden object faster than using only vision based screen object detection.

This study was designed to test the VA and PA performances. To execute the automation used Intel Core i7 (3.4GHz) processor, 4GB ram, Windows 7 OS and display resolution (1920×1080). QA testing purpose selects the Tizen IDE application [20]. Table 4 shows details of TCs, which is created to test the Tizen IDE for QA purpose. Each TC was executed thirty times randomly. TC-1 executes the ESOS, TC-2 executes the DLOS, TC-3 executes the MTLOS and TC-4 executes the SPS algorithm. There are two dependent variables in this study: number of events (interaction) and task completion

time. This information cannot observe directly and therefore, can only be measured after completion of all the data.

Table 4. Test Cases

Test Case	Steps	Search the Target Object	Proposed Method Used
TC-1	Click on style.css file from the file browser	footer p { font-size: 12px;}	ESOS
TC-2	Click Top-up menu Window->Preferences; New window Double Click Web->Click CSS Files->Encoding List	Korean, ISO 2022	DLOS
TC-3	Click Top-up menu File->new->Tizen Web Project; New window Click Sample->Mobile->Web App->TizenWinset	TizenWinset	MTLOS
TC-4	Click Top-up menu Window->Preferences; New window click Tizen SDK->Click Logging->Slider	Select Slider to DEBUG	SPS

4.2 Results

Figure 19 shows the number of mouse and key events has been used to execute the TCs. SA used 52 clicks; ESOS used 2 clicks and 2 key events to execute TC-1. SA used 30 clicks; DLOS used 1 click and 8 key events to execute TC-2. To Executes TC-3, SA used 18 clicks and MTLOS used 2 clicks and 8 key events. SA used 13 clicks; SPS used 1 click and 7 key events to execute TC-4. The interaction events shows that the proposed algorithms used less interaction events compared to only vision based systems. Figure 20 shows the total number of events (click and key) to execute the PA and VA. PA used six click events and 25 key events, same time VA used 113 click events to execute all the TCs. Table 5 illustrates the Mann-Whitney U test analysis results, where Z = 2.411 and p = 0.0163, which is statistically significant PA used less interaction events than VA. From this result can conclude that combination of mouse and key events required less interaction to execute the TCs which supports the H1 hypothesis.

Figure 21 shows TC-1 completion time of VA and ESOS, VA has two outlier values, VA took 30.74 seconds and ESOS took 1.91 seconds to complete the tasks, where N = 60, $\bar{x}$ of VA is 45.50, ESOS is 15.50, Z = 6.663 and p = 0.000002, which is statistically significant and the result shows that ESOS takes less time that VA.

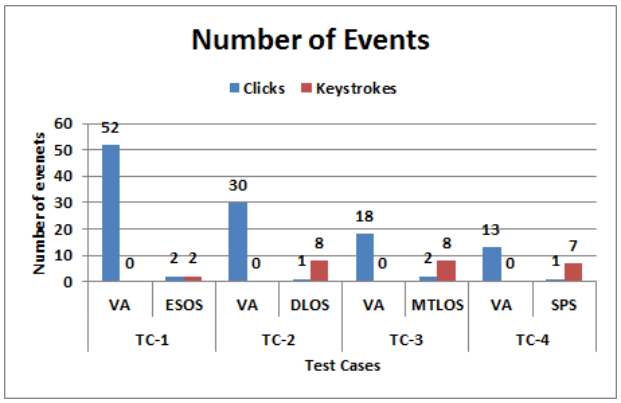

Figure 19. Number of Events for Each TC

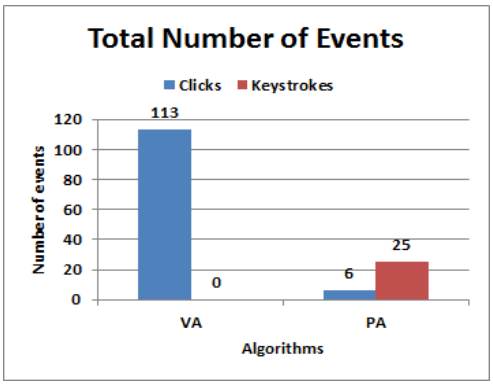

Figure 20. Total Number of Events

Table 5. Mann-Whitney U test analysis of number of events to finish tasks

Method	Events	*N*	*Mean*
VA	Click	4	6.5
	Key	4	2.5
PA	Click	4	2.5
	Key	4	6.5
Test Statics			
Z		2.411	
***p* value (2 tailed)**		0.0163	

Figure 22 illustrates the TC-2 completion time, VA has three outlier values, VA took 7.29 seconds and DLOS took 2.96 seconds, where N=60, $\bar{x}$ of SA is 45.50 and ESOS is 15.50, Z = 6.663 and p = 0.000002, which is statistically significant and the result shows that DLOS takes less time than VA.

Figure 23 illustrates the TC-3 completion time, VA has five outlier values and MTLOS has one outlier value. VA took 19.14 seconds and MTLOS took 13.29 seconds, where N = 60, $\bar{x}$ of SA is 45.50, ESOS is 15.50, Z = 6.654 and p = 0.0000002, which is statistically significant and the data shows that MTLOS takes less time than VA.

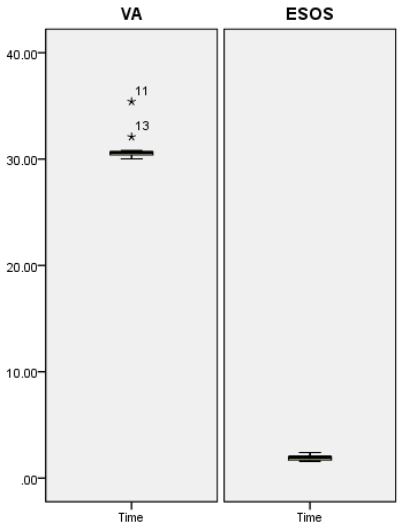

Figure 21. TC-1 Completion Time

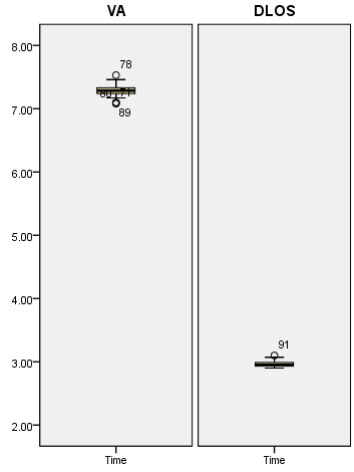

Figure 22. TC-2 Completion Time

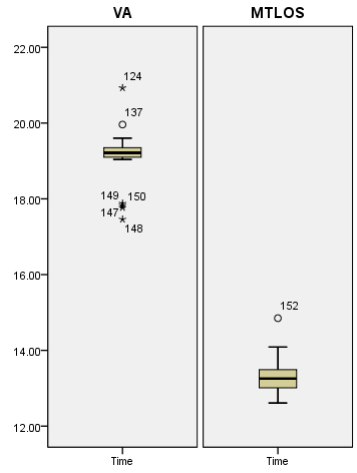

Figure 23. TC-3 Completion Time

Figure 24 shows the TC-4 completion time, VA has six and SPS has one outlier value. VA took 15.94 seconds and SPS took 9.85 seconds, where N = 60, $\bar{x}$ of SA is 43.50, ESOS is 17.50, Z = 5.767 and p = 0.000008, which is statistically significant and the results shows that SPS takes less time than VA.

Figure 25 shows the completion time of four TCs, VA took 18.27 seconds and PA took 7 seconds. Table 6 shows the average TCs completion time and table 7 shows the Mann-Whitney U test analysis results, where N = 240, $\bar{x}$ of SA is 164.50, PA is 76.50, Z = 9.819 and p = 0.0000001, which is statistically significant and the results shows that PA takes less time than VA. From this analysis can conclude that combination of mouse and key events, automation systems can trace the hidden objects faster which supports the H2 hypothesis.

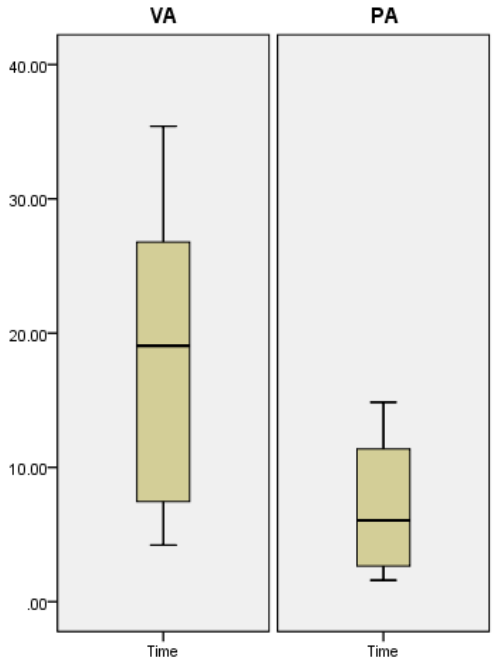

Figure 24. Total TC Completion Time

TABLE 6. Average execution time

	VA	Proposed Methods
Test Case	**Execution Time (sec)**	**Execution Time (sec)**
TC-1	30.74	1.91
TC-2	7.29	2.96
TC-3	19.14	13.29
TC-4	15.94	9.85
TOTAL (AVG)	**18.27**	**7**

TABLE 7. MANN-WHITNEY U TEST ANALYSYS OF COMPLETION TIME TO FINSH TASKS

Methods	***N***
VA	120
PA	120
Total	240
Test Statistics	
Z	9.819
***p* value (2 tailed)**	0.0000001

5 Conclusion

GUI automation tools enhance Test Case execution and reduce human efforts. Most of the Black Box Test Cases can be executed with this system; limitation of the technology vision based automation system in some cases take time and failed to find/search hidden objects and dynamic appearance of the objects. As a result all type of TCs cannot be executed using VA system. The proposed techniques have the unique features to identify hidden objects even the window objects screen position gets change. The proposed methods are implemented in real time automation application, which can discover the hidden objects smoothly and enhance the re-usability. The usability study results show that combination of key and mouse events in the VA system can find the hidden target object faster. These methods enhance the VA systems to find the target object faster which will help the QA testers to get the result quicker. Currently there is one limitation with these methods. It takes time to check the list box objects one by one to search the hidden target object. Future plan is to overcome these two limitations and works for complete introducing full testing framework for hidden object detection.

References

[1] S. Inderjeet, and T. Bindia, "Comparative Analysis of Open Source Automated Software Testing Tools: Selenium, Sikuli and Watir", International Journal of Information & Computation Technology, vol. 4, pp. 1507-1518, 2014.

[2] K. Dea-Kwang, and L. Lee-Sub, "Reverse Engineering from Exploratory Testing to Specification-based Testing", International Journal of Software Engineering and Its Applications, vol. 8(11), pp. 197-208, 2014.

[3] L. Maurizio, S. Andrea, R. Filippo, and T. Paolo, "Automated Generation of Visual Web Tests from DOM-based Web Tests", ACM/SIGAPP Symposium on Applied Computing, April, 2015.

[4] M. Leotta, A. Stocco, F. Ricca, P. Tonella, "PESTO: A Tool for Migrating DOM-Based to Visual Web Tests", ACM/SIGAPPSymposium on Applied Computing, April, 2015.

[5] L. Capocchi, J.F. Santucci, T. Ville, "Software Test Automation using DEVSimPy Environment", International Conference on Principles of Advanced Discrete Simulation, May, 2013.

[6] Borjesson, and F. Robert, "Automated System Testing using Visual GUI Testing Tools: A Comparative Study in Industry", "Borjesson2012visual", 2012.

[7] J. Hyunjun, L. Sukhoon, B. Doo-Kwon, "An Image Comparing-based GUI Software Testing Automation System", World Congress in Computer Science, Computer Engineering, and Applied Computing, 2012.

[8] K. Pragya, "Ameliorating the image matching algorithm of Sikuli using Artificial Neural Networks", International Journal of Computer Science & Communication, vol. 5, pp. 1-4, 2014.

[9] http://www.sikulix.com

[10] C. Tsung-Hsiang, "Using graphical representation of user interfaces as visual references", The 24th annual ACM symposium adjunct on User interface software and technology, pp. 27-30, 2011.

[11] S. L. M. Jeffrey, "User interface computation as a contextualized approach for introductory computing instruction", The 9th Annual International ACM Conference on International Computing Education Research, pp. 179-180, 2013.

[12] V. Andriychenko, L. Ying-dar , C. T. National, "Automatic Functionality and Stability Testing Through GUI of Handheld Devices", CiteSeerx, 2011.

[13] C. Tsung-Hsiang, Y. Tom, C. M. Robert, "GUI testing using computer vision", CHI 10th Conference on Human Factors in Computing Systems, pp. 1535-1544, 2010.

[14] Y. Tom, C. Tsung-Hsiang, C. M. Robert, "Sikuli: using GUI screenshots for search and automation", The 22nd annual ACM symposium on User interface software and technology, pp. 183-192, 2009.

[15] http://robotframework, Jan, 2015.

[16] http://smartbear.com/product/testcomplete, Jan, 2015.

[17] http://www.testplant.com, Jan, 2015.

[18] http://www.froglogic.com, Jan, 2015.

[19] O. S. Pietro, H. Jun, Y. Xin, "Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results", International Journal of Automation and Computing, vol. 4, pp. 218-293, 2007.

[20] https://www.tizen.org, Jan, 2015.