Artificial Intelligence in Automated System for Web-Interfaces Visual Testing

Kateryna Ivanova[0000-0003-0072-2591], Galyna Kondratenko[0000-0002-8446-5096], Ievgen Sidenko[0000-0001-6496-2469], Yuriy Kondratenko[0000-0001-7736-883X]

Intelligent Information Systems Department, Petro Mohyla Black Sea National University, 68th Desantnykiv Str., 10, Mykolaiv, 54003, Ukraine,
ivnat97@gmail.com, halyna.kondratenko@chmnu.edu.ua, ievgen.sidenko@chmnu.edu.ua, yuriy.kondratenko@chmnu.edu.ua

Abstract. In this paper, the authors consider an artificial intelligence technique of providing visual testing, and also the developed system that is integrated into functional automated test suites. Thus carried out monitoring and analyzing of visual changes in the graphical interface of the application under test. A proposed tool is supposed to resolve the existing problems of the traditional snapshot visual testing. Graphical user interface (GUI) testing is a very important testing step for quality control of software applications. The GUI is the central node in the test application, from where all functions are accessed. Thus, it is difficult to thoroughly test programs through their graphical interface, especially because they are designed to work with humans, not machines. In addition, they are inherently non-static interfaces, prone to constant changes caused by functionality upgrades, improved usability, changing requirements or changed contexts. This complicates the development and maintenance of test cases without resorting to time-consuming and costly manual testing. A proposed automated system for web-interfaces visual testing uses computer vision technology as an artificial intelligence technique for visual comparison. A comparative analysis is carried out with the developed interface for testing (in particular, a web page) and the expected mockup with the location of visual elements on the page for example, an interface from the customer). When designing an automated system for web-interfaces visual testing, the programming languages Python, JavaScript, library TensorFlow, testing framework Cypress, and database MySQL were used.

Keywords: automated testing, visual testing, artificial intelligence, snapshot, digital image, AI algorithms, computer vision, graphical interface, visual element, pixel comparator.

1 Introduction

Automated visual testing is a quality assurance process designed to automatically check the visual display of the interface. Currently, there are two main areas of automated visual testing: verification of CSS element style names and comparison of screenshots taken during automated functional tests [1].

The biggest problem with automated visual testing is that people and machines perceive pixels differently. Two images of the user interface may appear completely identical to a person, but differ at the pixel level. Anti-aliasing algorithms for resizing images, different video cards create differences in pixels. Thus, the testing program, which should obtain the exact match of points between two figures, may be filled with pixel differences, that is, a big amount of false defects.

The subject of the study is methods of analysis and comparison of expected and actual images of the interface. The relevance of this work is determined by the fact that due to the rapid development of automated visual testing, there is a need to develop a system that will allow more accurate analysis of the interface and reduce the number of false results.

The purpose of the paper is to investigate artificial intelligence tools and technologies for analyzing, processing and comparing GUI images captured in the process of performing automated regression function tests, and creating a system that will allow tracking web application interface errors, generating a report, and editing basic (expected) GUI images.

In this paper the authors proposed to create a system that will be integrated into functional automated test suites, while performing the function of monitoring and analyzing visual changes in the graphical interface of web applications.

2 Related Works and Problem Statement

One of the traditional methods of automated visual testing is to check the CSS styles of the corresponding HTML page element. These checks are performed during the execution of functional tests based on various frameworks, such as Selenium Webdriver, Cypress, WebdriverIO, or Appium. The driver manages the web application as the user would, and checks whether the application is working properly [2].

For the item on a page you can perform the following checks (Fig. 1):

1. Element .todo-list must have a specific color represented in the sixteen-year-old form #d6d6d6;
2. Element .todo-list should have the text design crossed out-line-through.

An example of such checks using the JavaScript programming language and the Cypress test framework is shown below:

$$\begin{aligned} &\text{cy.get}(\text{'.completed'}).\text{should}(\text{'have.css'},\ \text{'text}-\text{decoration'},\ \text{'line}-\text{through'}) \\ &\text{cy.get}(\text{'.completed'}).\text{should}(\text{'have.css'},\ \text{'color'},\ \text{'rgb}(217,217,217)\text{'}) \end{aligned}.$$

When using usual checkpoints in program tools of functional testing such as Selenium Webdriver, Cypress, WebdriverIO, or Appium, you must to view the main visual components: visible, top coordinates, height, width, background color. This means that you will need the next amount of approvals:

$$20\ \textit{visual elements} \cdot 5\ \textit{statements per element} = 105\ \textit{lines of code}.$$

It will be extremely difficult to find all visual errors, even using all the code suggested above. For example, you can't access a visual element if it's hidden under another one.

Fig. 1. Sample styles of a particular visual element

Thus, this method of verification has certain disadvantages: significant increase in the code of test methods; complexity of supporting test code, since changing style names can cause a large number of failed tests; this method will not detect a valid visual defect, such as an element offset on the page [3].

Screenshot testing. This method is implemented as follows. First-generation automated visual testing uses snapshot testing technology. When testing snapshots, the

screen bitmap is trapped at different points in the test run, and its pixels are compared with the base bitmap.

The algorithms for testing images are very simple: we iterate over a pair of pixels and then check whether the sixteen-year-old color code matches. If the color codes differ, a visual error report is generated [4-6].

There are a large amount of open source and commercial program tools for snapshot testing available because they can be created easily enough. Unlike manual testing, program tools for visual testing of images, you can rapidly identify the differences of pixels. And this is a step forward, because the computer can identify visual differences easily and accurately. Some of these program testing tools called "pixel-perfect testing" (Fig. 2).

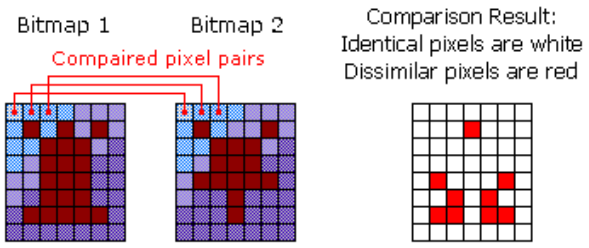

Fig. 2. Pixel image comparison process

Another problem is that these tests are often slow compared to lighter unit tests that don't require a full browser (Fig. 3).

Summing up, comparing images of the expected result (ER) and the actual result (AR) as follows has the following disadvantages [5, 7-11]:

- inability to search for visual changes at different image resolutions;
- significant slowdown in the execution time of automated test suites;
- it is not possible to search for visual changes separately for a specific component of the graphical interface;
- high probability of false alarms.

Fig. 3. Example of a false positive when comparing screenshots

3 Methods of Image Segmentation and Processing

When selection a method to process GUI screenshot, we should consider the following: perform font smoothing; different situations in which styles and elements are displayed by different browsers; ability to compare images with different resolutions.

The process of image segmentation is based on the information available in the image itself. This can be information about the color of the pixel, differences in colors and shades, or the texture of the image [9].

K-means algorithm. The K-means algorithm was proposed in 1979, but it is still relevant today. The idea behind it is quite simple, and the algorithm works quite effectively. However, the optimality of solutions obtained using the K-means algorithm is not guaranteed. In addition, the disadvantage of the algorithm is the need to know the number of clusters in advance [10, 12-17].

Formally, the solution to the clustering problem is to "mark" each of the existing objects-assign it a number of a certain class. The K-means algorithm assumes that objects are divided into classes in such a way that differences ("distances") between objects of the same class are minimized and differences between objects of different classes are maximized.

The current algorithm is a base of almost all fuzzy clustering algorithms, and its detail analysis will help us better understand the principles embedded in more complex algorithms.

In General, the algorithm is an iterative procedure [18-20]:

Step 1. Initialize the initial partition matrix U randomly and select the accuracy δ that will be used to complete the algorithm, set the iteration number $l = 0$.

Step 2. To determine the cluster centers:

$$c_l^{(i)} = \frac{\sum_{j=1}^{d} u_{ij} m_j}{\sum_{j=1}^{d} u_{ij}}, 1 \le i \le c \,, \tag{1}$$

where $c_l^{(i)}$ is the cluster centers, d is the dimension of the object, c is the amount of clusters, l is the amount of the current iteration, u is a matrix of partitioning m object, m is the object, which is investigated.

Step 3. To update the matrix splitting:

$$u_{ij}^{(l)} = \begin{cases} 1, \ if \ \ d\left(m_j, c_i\right) = \min\limits_{l \le k \le c} d\left(m_j, c_k\right) \\ 0, \qquad in \ other \ cases \end{cases}, \tag{2}$$

where $u_{ij}^{(l)}$ is an element of the partition matrix, $d\left(m_j, c_i\right)$ is the selected metric, c is the cluster, m is the object, which is investigated.

Step 4. Verify the constraint $\left\|U^{(l)} - U^{(l-1)}\right\| < \delta$, where U is the partition matrix, and δ is the selected precision. If the condition is met, end the process, if not, go to step 2 with the iteration number $l = l + 1$.

There is also an alternative version of this algorithm:

Step 1. Randomly select cluster centers from input data elements and select the accuracy δ that will be used to complete the algorithm, set the iteration number $l = 0$.

Step 2. Update the split matrix (1).

Step 3. To determine the cluster centers (2).

Step 4. Check how much the cluster centers have shifted. If they have shifted less than the accuracy of δ – complete the process, if not-go to step 2 with the iteration number $l = l + 1$.

There is also a set of restrictions: $u_{ij}^{(l)} \{0,1\}; \sum_{j=1}^{c} u_{ij} = 1; 0 < \sum_{j=1}^{d} u_{ij} < d$, which determines that each data vector can belong to only one cluster and not to the others. Each cluster must have at least one data item and no more than the total number of items.

The main disadvantage of this algorithm due to the discreteness of the elements of the partition matrix is the large size of the spatial partition. To eliminate this disadvantage is to represent the elements of the partition matrix with numbers from the range from 0 to 1. That is, the membership of a data item to a cluster is determined by the membership function, in this way the data item can be a member of few clusters with different degrees of ownership [16]. Other disadvantages and limitations of the K-means algorithm are: you need to know the number of clusters in advance; the algorithm is very sensitive to the choice of initial centers of clusters; the classical variant implements a random choice of clusters, which is very often a source of error; does not cope with the task when the object belongs to different clusters equally or does not belong to any one.

Neural networks and fuzzy logic are also used for image segmentation [21-25].

Mean Shift method. It is not necessary to determine the amount of clusters in advance. This is defined based on the source data. The direction to the centroid of the nearest cluster is defined by where most of the nearest points are located.

Mean Shift is a great iterative method. The average offset was proposed by Fukunaga and Hostetler and expanded for use in other areas, such as computer vision. The method considers functional space as an empirical function of probability density. If the input data is a set of points, then mean shift method considers them as a sample from the basic probability density function. If there are dense regions in the functional space, they belong to the form of the probability density function. Clusters can be identified associated with this form using the mean shift method [17-20].

Mean Shift groups objects with similar attributes. Pixels with similar features are combined into a single segment, and the output is an image with uniform areas.

For example, you can select pixel coordinates (x, y) and RGB pixel components as coordinates in the feature space. When you draw pixels in the feature space, you can notice a thickening in certain places [21].

To make it easier to describe the thickening of points, a density function is introduced:

$$f\left(\vec{x}\right)=\frac{1}{Nh^{d}}\sum_{i=1}^{N}K\left(\frac{\vec{x}-\vec{x_{i}}}{h}\right), \tag{3}$$

where $\vec{x}$ is the feature vector of the i-th pixel, d is the amount of features, N is the amount of pixels, h is a parameter responsible for smoothness.

The maxima of the function are located at the points where the image points are condensed in the feature space. Pixels that belong to the same local maximum are combined into a single segment. It turns out that to find which of the centers of condensation the pixel belongs to, you need to step along the gradient to find the nearest local maximum (Fig. 4) [21, 26].

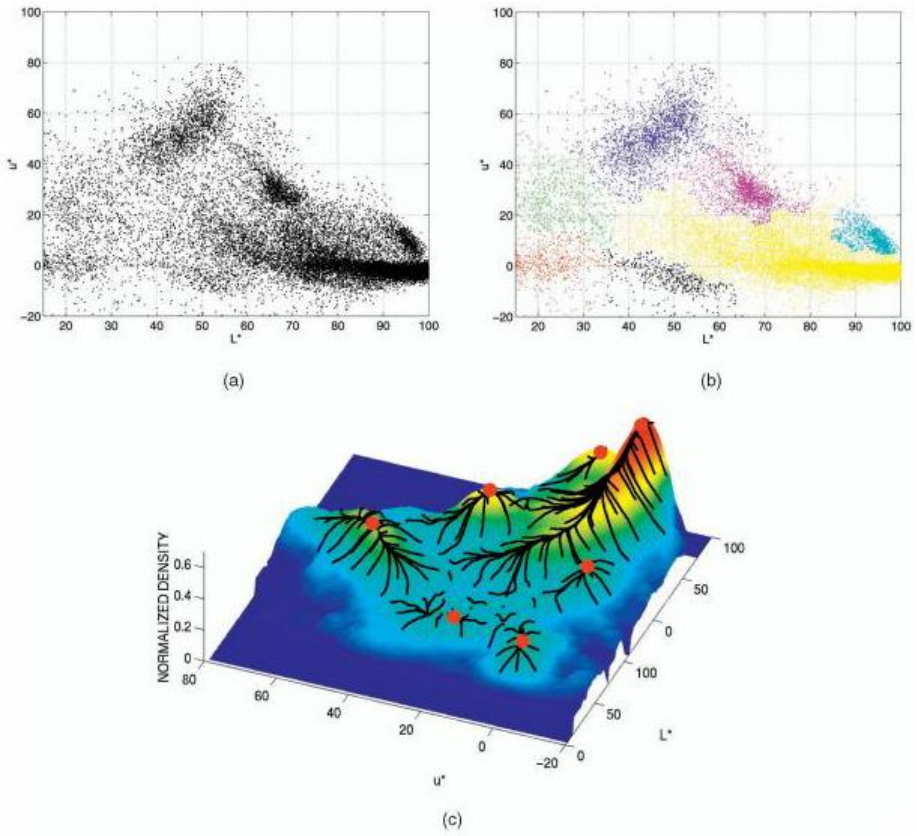

Fig. 4. Pixels on the image: (a) pixels in a two-dimensional feature space, (b) pixels that arrive at the same local maximum are colored the same color, and (c) density function, the maxima correspond to the places of the highest concentration of points

When you select point coordinates and color intensities as attributes, pixels with similar colors and located close to each other will be combined into a single segment.

Accordingly, if you choose another feature vector, then the Union of the pixels in the segments will already be on it. For example, if you remove coordinates from features, then the sky and the lake will be considered one segment, since the pixels of these objects in the feature space would fall within the same local maximum [19-21].

4 Practical Implementation

In the process of designing the system, the task was to create a system that will be integrated into functional automated test suites, while performing the function of monitoring and analyzing visual changes in graphical interfaces of web applications.

The system will work in the following order:

1. Loading basic images determines the expected appearance of the web application at each step of the test;
2. Capture a screenshot of the GUI;
3. Analysis and comparison of images of the expected result and the actual result based on the selected algorithm;
4. Generating a report on test results;
5. The decision-maker should review the detected changes and ignore them or report an error;
6. The base images or expected result should be updated according to the ignored changes.

Thus, clustering algorithms for image segmentation in practice should be considered in detail. For example, consider the main page (Fig. 5) of a web-application for booking air tickets of Ukrainian International Airlines (UIA).

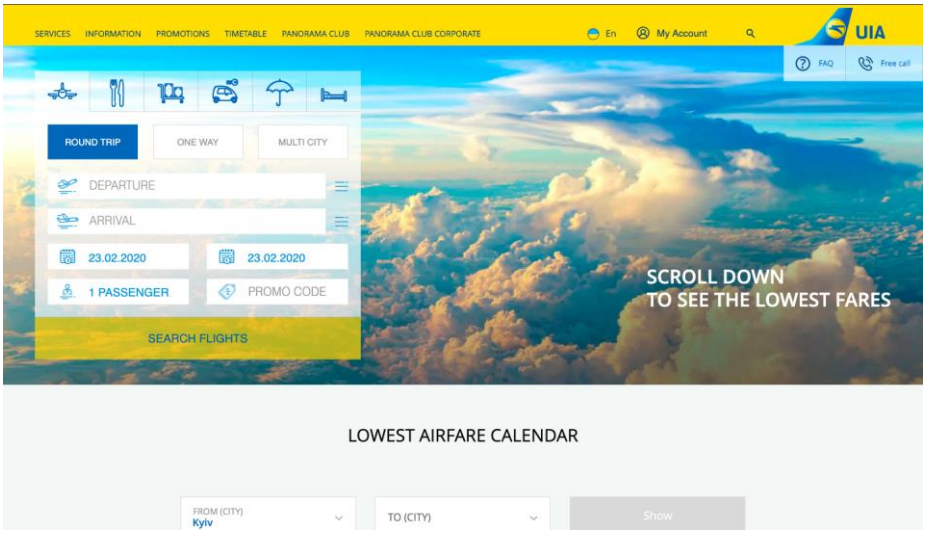

Fig. 5. Input image

Let's try to apply the K-means algorithm (Fig. 6) and the Mean Shift method (Fig. 7) to segment the input image (Fig. 5) with 4 clusters.

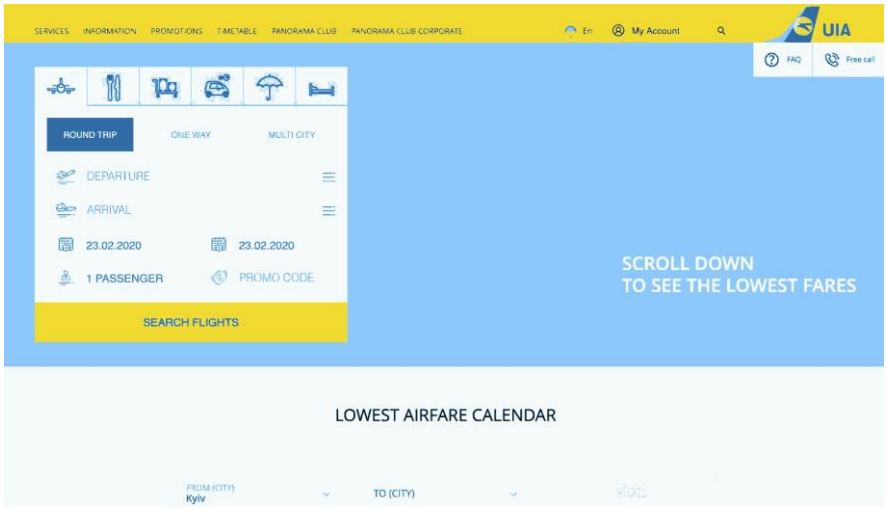

Fig. 6. The result of image segmentation by the K-means algorithm

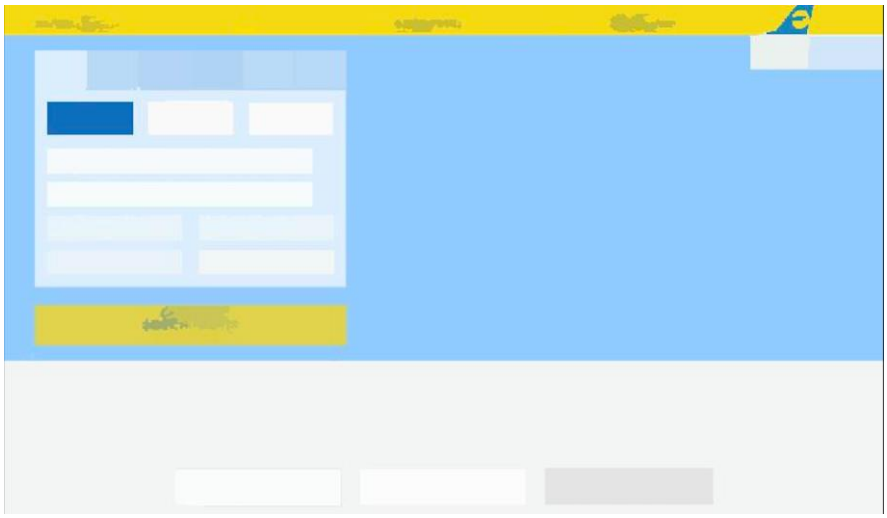

Fig. 7. The result of image segmentation by the Mean Shift method

As a result (Fig. 6), you can see that after processing the image, we lost a significant number of elements and all the text on the page was not smoothed out. In addition, if fewer clusters are used, a significant number of colors are lost, which will cause the element color to be accidentally changed by the tester. Thus, the results of using dif-

ferent image segmentation methods (K-means (Fig. 6) and Mean Shift (Fig. 7)) show that the Mean Shift method is better for the current task, since we get clearly highlighted visual elements on the page, no text on the screenshot and the same results at different image resolution.

At the stage of database design, the appropriate structure of tables was identified and implemented (Fig. 8). When visual testing is complete, the user can log in, view the report from the appropriate test, and either reject or accept the error (Fig. 9).

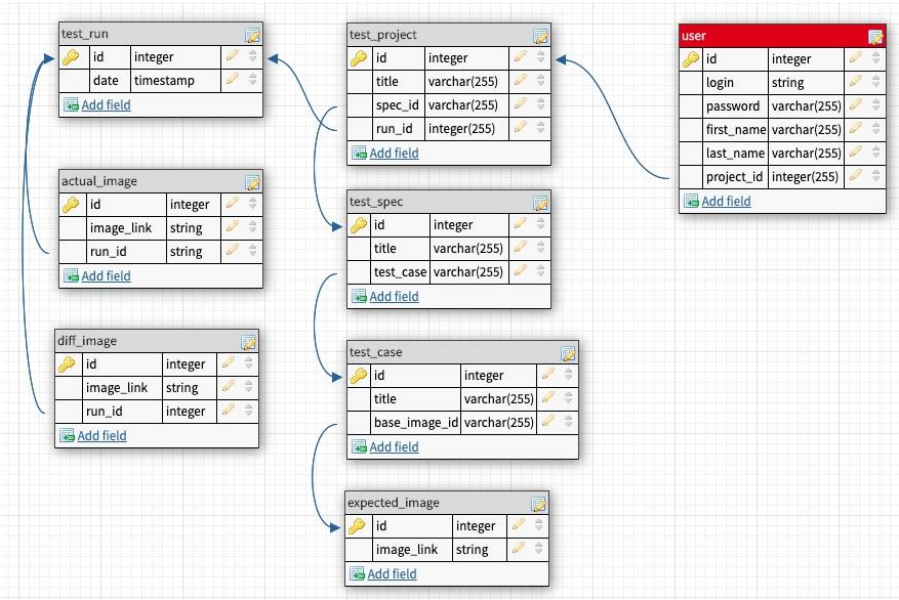

Fig. 8. Database structure for visual testing system

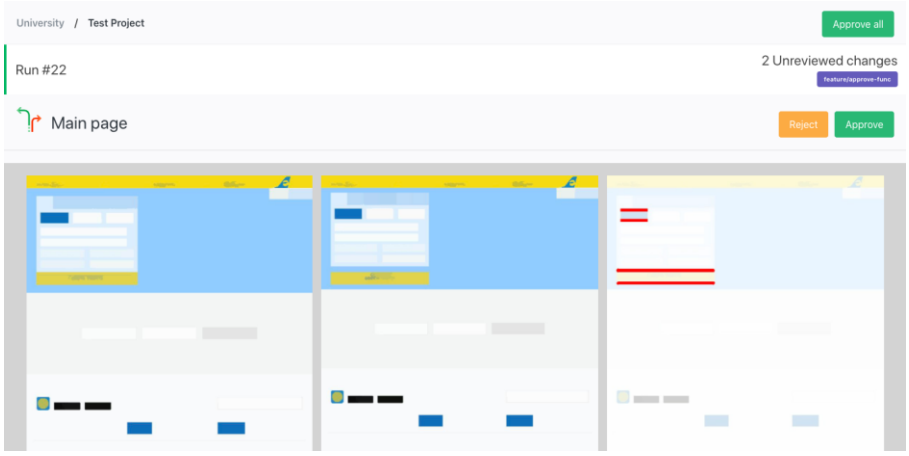

Fig. 9. A page for viewing the results of visual testing of the developed system

In addition, the module architecture for image analysis and comparison was designed. Based on the defined class diagram, a system was implemented. When designing an automated system for web-interfaces visual testing, the programming languages Python, JavaScript, library TensorFlow, testing framework Cypress, and database MySQL were used.

5 Discussion

The GUI testing is a very important testing step for quality control of software applications. Despite the fact that many automated testing tools and methods have been developed, they still do not solve all problems, such as tracking visual changes to the interface. The first part of the paper deals with the review of the theoretical foundations of automated visual testing, the review of existing methods and software. The next sub-task was to conduct a detailed analysis of the basic methods and approaches for automated visual testing, including methods for element style verification and pixel-by-pixel image comparison. The authors discuss clustering methods (K-Means and Mean Shift) for image segmentation. When selecting image processing methods, a detailed review and comparative analysis were conducted. The main result of the work is the creation of a system for automated visual testing. To solve this problem, the following tasks were performed: features and stages of web application development were considered; software for the development of the appropriate system was selected; image segmentation techniques were applied; system architecture was defined; a system for visual testing was developed and its performance was tested.

6 Conclusions

Due to the rapid development of automated visual testing, there is a need to develop a system that will allow you to perform a more accurate analysis of the interface and reduce the number of false positives.

This paper describes the study of artificial intelligence tools and technologies for analyzing, processing and comparing graphical interface images captured during automated regression functional tests, and to create a system that will allow users to track errors in the web application interface, generate a report, and edit the basic (expected) GUI image.

A proposed automated system for web-interfaces visual testing uses the Mean Shift method as an artificial intelligence technique for visual comparison. A comparative analysis is carried out with the developed interface for testing (in particular, a web page) and the expected mockup with the location of visual elements on the page for example, an interface from the customer). When designing an automated system for web-interfaces visual testing, the programming languages Python, JavaScript, library TensorFlow, testing framework Cypress, and database MySQL were used.

The result of this paper is a developed automated system for visual testing of web interfaces using artificial intelligence methods for image segmentation and more accurate analysis.

References

1. Graves, T., Harrold, M., Kim, J., Porter, A., Rothermel, G.: An empirical study of regression test selection techniques. ACM Transactions on Software Engineering and Methodology (TOSEM) 10(2), 184-208 (2001).
2. Li, P., Huynh, T., Reformat, M., Miller, J.: A practical approach to testing GUI systems. Empirical Software Engineering 12(4), 331-357 (2007).
3. Lönnberg, J.: Visual testing of software. Master's thesis, Helsinki University of Technology (2003).
4. Myers, G.: The Art of Software Testing. John Wiley & Sons, N.Y. (2004)
5. Weyuker, E.: Axiomatizing software test data adequacy. IEEE Transactions on Software Engineering 1215(12), 1128-1138 (1986).
6. Schneider, G., Winters, J.: Applying Use Cases: A Practical Guide. Addison Wesley, Boston (1998).
7. Cohn, M.: Agile Estimating and Planning. Addison Wesley, Boston (2005).
8. Burnett, M., Ambler, A.: Interactive visual data abstraction in a declarative visual programming language. Journal of Visual Languages and Computing 5(1), 29-60 (1994).
9. Azem, A., Belli, F., Jack, O., Jedrzejowicz, P.: Testing and reliability of logic programs. The Fourth International Symposium on Software Reliability Engineering, 318-327 (1993).
10. Mikhov, D., Kondratenko, Y., Kondratenko, G., Sidenko, I.: Fuzzy logic approach to improving the digital images contrast. In: IEEE 2nd Ukraine Conference on Electrical and Computer Engineering, UKRCON, pp. 1183-1188, Lviv, Ukraine (2019).
11. Pomanysochka, Y., Kondratenko, Y., Kondratenko, G., Sidenko, I.: Soft computing techniques for noise filtration in the image recognition processes. In: IEEE 2nd Ukraine Conference on Electrical and Computer Engineering, UKRCON, pp. 1189-1195, Lviv, Ukraine (2019).
12. Mohanty, H., Mohanty, J., Balakrishnan, A.: Trends in Software Testing. Springer, Singapore (2017).
13. Robert, M., Linda, H., Shapiro, G.: Image segmentation techniques. [Online]. Available: https://www.sciencedirect.com/science/article/ pii/S0734189X85901537
14. Kaur, D., Kaur, Y.: Various Image Segmentation. Techniques: A Review 3(5), 809-814 (2014).
15. Hlaing, S., Khaing, A.: Weed and crop segmentation and classification using area thresholding. International Journal of Research in Engineering and Technology, 2321-7308 (2014).
16. Haug, S., Michaels, A.: Plant classification system for crop/weed discrimination without segmentation. IEEE Winter Conference (2014). DOI:10.1109/WACV.2014.6835733.
17. Lottes, P., Stachniss, C.: Semi-Supervised Online Visual Crop and Weed Classification in Precision Farming Exploiting Plant Arrangement. [Online]. Available: http://flourishproject.eu/fileadmin/user_upload/publications/lottes17iros.pdf.
18. Wang, Y., Chen, Y., Lu, P., Wang, H.: Sobel Heuristic Kernel for Aerial Semantic Segmentation. In: 25th IEEE International Conference on Image Processing (ICIP), Electronic ISSN: 2381-8549.
19. Fan, W.: Color image segmentation algorithm based on region growth. Computer Engineering 36(13), 25-34 (2010).
20. Angelina, S., Suresh, L., Veni, S.: Image segmentation based on genetic algorithm for region growth and region merging. In: International Conference on Computing, Electronics and Electrical Technologies (ICCEET), pp. 970-974 (2012).

21. Ugarriza, L., Saber, E., Vantaram, S. et al.: Automatic image segmentation by dynamic region growth and multiresolution merging. IEEE transactions on image processing 18(10), 2275-2288 (2009).
22. Pomanysochka, Y., Kondratenko, Y., Sidenko, I.: Noise filtration in the digital images using fuzzy sets and fuzzy logic. In: 15th International Conference on ICT in Education, Research, and Industrial Applications: PhD Symposium (ICTERI 2019: PhD Symposium), vol. 2403, pp. 63-72, Kherson, Ukraine (2019).
23. Kondratenko, Y., Gordienko, E.: Neural Networks for Adaptive Control System of Caterpillar Turn. In: Annals of DAAAM for 2011 & Proceeding of the 22th Int. DAAAM Symp. "Intelligent Manufacturing and Automation", 2011, pp. 0305-0306.
24. Kushneryk, P., Kondratenko, Y., Sidenko, I.: Intelligent dialogue system based on deep learning technology. In: 15th International Conference on ICT in Education, Research, and Industrial Applications: PhD Symposium (ICTERI 2019: PhD Symposium), vol. 2403, pp. 53-62, Kherson, Ukraine (2019). http://icteri.org/icteri-2020/PhDS/11110053.pdf.
25. Siriak, R., Skarga-Bandurova I., Boltov, Y.: Deep Convolutional Network with Long Short-Term Memory Layers for Dynamic Gesture Recognition. In: 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), pp. 158-162, Metz, France (2019). DOI: 10.1109/IDAACS.2019.8924381.
26. Celebi, M., Kingravi, H., Vela, P.: A comparative study of efficient initialization methods for the K-means clustering algorithm. Expert Systems with Applications 40(1), 200-210 (2013).