Wireframe to HTML and CSS using Computer Vision

Matthew Garrett
Computer Science and Software Engineering
University of Canterbury
Christchurch, New Zealand
mga114@uclive.ac.nz

Richard Green
Computer Science and Software Engineering
University of Canterbury
Christchurch, New Zealand
richard.green@canterbury.ac.nz

***Abstract*—Rapid prototyping is vital when creating user interfaces, however the development time for converting a single wireframe design to code is high. This paper proposes a method to automatically generate HTML and CSS for a webpage based on a wireframe design using traditional computer vision techniques. This paper looks into previous solutions, and uses character recognition and extraction with Tesseract, along with the border following algorithm for contour detection to identify elements in a wireframe design. These elements are then converted into an HTML file with Tailwind CSS, which can be run as a webpage. The solution manages to accurately recreate 87.3% of objects in a wireframe design, and 73.3% of text. 2.5% of objects were not present in the output HTML, compared to 3.3% of all text. The remaining text issues were largely issues with individual character detection, and because the output is expected to be changed from the wireframe look, this is not a large problem for this method. This is much better than prior research in this space, however this method requires heavy limitations on the wireframe design for the input compared to previous methods, and fails to generalise for all types of wireframe inputs.**

***Index Terms*—Computer Vision, Website Generation, Wireframe, Tesseract OCR**

I. Introduction

This paper proposes a method of generating HTML and CSS code for a website based on an image of the website's wireframe design. This allows designers with no prior website design knowledge to generate a website layout from their chosen design. Optical character recognition techniques are used to extract and remove text information from a design image, before an adapted border following algorithm detects non-text shape information. The information gathered from the image is then processed into an HTML output with Tailwind CSS, which can be run as a website to display the converted input image elements as HTML components. HTML and CSS are used together to structure and style web pages, with the content being controlled by HTML, and the styles handled by CSS. Most large scale companies use UI component libraries for web development, and converting a wireframe design to HTML has been identified as one of the biggest costs for front end web development [5]. This method allows for rapid iteration of website layout design changes, from the wireframe design to website code. The proposed method also gives the user freedom of choice for design software and requires no HTML or CSS knowledge. This paper will analyse the accuracy, limitations, and efficiency of using this method compared to manually coding the design into HTML.

II. Background

A. Related Work

Previous studies have been performed into generating front-end code from an image. Studies into converting screenshots of mobile apps into element objects outlined the importance of performing text recognition and removal as a prior step to object recognition [3] [8] [15]. These studies analysed deep learning techniques, as well as pure computer vision techniques for application object recognition, and discovered improved detection when using the Faster-RCNN architecture compared to computer vision techniques. Faster-RCNN, and other deep learning models, were found in another study to be less effective at generic website object detection compared to computer vision techniques [15]. Another model, UIED, was found to better detect objects when trained on 90,000 inputs, and achieved a 46% higher F1-score for accuracy [15]. The improved accuracy of deep learning techniques were found to be negligible when accounting for a noiseless ideal image, such as a direct screenshot of a requested size from a design tool [8]. Additionally, the main benefit of using deep learning techniques for website element generation is to cover cases over all colours, shapes, and styles [3]. Style detection and generation does not apply to wireframe designs, and an implementation of a hybrid approach to a similar problem found no change in the positioning accuracy of objects using computer vision when compared to deep learning [6].

B. Text Detection

Text recognition in images is done using OCR (optical character recognition) software. Tesseract OCR is open-source and is a widely used solution that can detect and read text information from images. Tesseract detects potential points of characters from concave vertices of a proposed outline, and attempts to join the vertices together to form a character [12]. Tesseract can accept new training data, however a large amount of the data used for the initial training of Tesseract is printed fonts, rather than written text [12].

Website design tools such as Figma are the primary input generators for this method, and as such the initial training

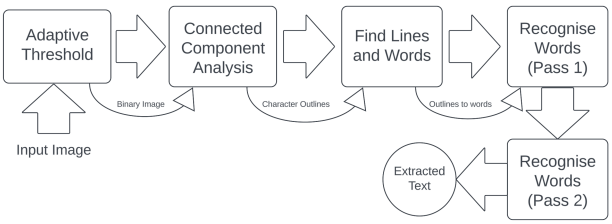

Fig. 1. Tesseract OCR Flow [2]

of Tesseract has previously been found to be acceptable for screenshot based text recognition [3]. Because Tesseract is a model trained on input data, image processing techniques are recommended to achieve a better result. A study performed into the accuracy of text detection with Tesseract found it possible to achieve a 92% word accuracy after performing image processing [11]. The process flow for Tesseract OCR can be seen in figure 1 [2].

C. Object Detection

Typical object detection algorithms for generating GUI elements from an input image focus on using deep learning models such as Faster-RCNN [17]. Input images are manually trained along with the corresponding expected UI element code to generate a page and layout [19]. Typical minimum dataset sizes required for training the model were found to be 600 distinct versions of GUI elements, and this would need to be done for each expected input method to the algorithm [14]. Another study into the effectiveness of deep learning based solutions for GUI object detection inspected the border following algorithm and found more reliable results compared to deep learning models [3].

The border following algorithm finds pixels and their borders, and connects to form a contour of the shape [16]. Borders are considered similar pixels, that satisfy the below equation with threshold values n, m [16].

$$X_p - m \leq X \leq X_p + m, Y_p - n \leq Y_p \leq Y_p + n \quad (1)$$

D. Previous Limitations

When performing website image recognition, it is vital to extract the text information before running the object detector [3] [15]. Tesseract also performs best without further training for black text on white background [11], and as such the input image should have processing applied. Tesseract also struggles with detecting nested text, such as text inside of an object with high similarity to a character in the training set [10], and further image processing steps were found to increase the reliability and accuracy of Tesseract without resorting to deep learning models [10].

Prior deep learning techniques for object detection require a large amount of data and processing to accurately train, and the increase in accuracy was not found to be significant for specific cases, such as object detection in wireframe designs [8]. As mentioned, current deep learning approaches for GUI

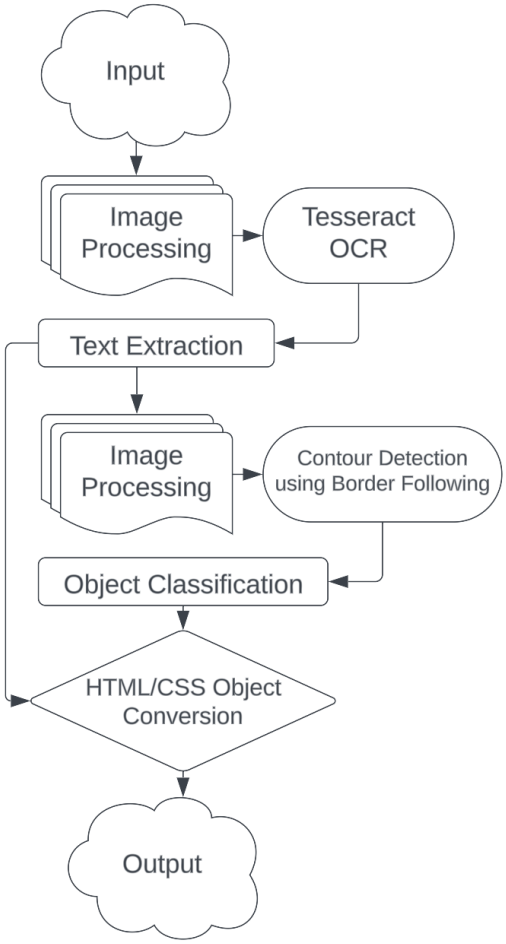

Fig. 2. Method workflow proposal

generation focus on the style, however for a wireframe design object positioning accuracy is much more relevant. Wireframe designs typically consist of simple shapes of solid colours [20], which can easily be handled by classical computer vision shape detection algorithms [3]. Additionally, the input image for object detection is required to match the standards set by the training data [14].

III. Proposed Method

The proposed method takes an input of a screenshot image of a website wireframe design, performing text extraction followed by object detection, and converting the resulting data into an HTML file that includes relevant CSS classes based on detected object positioning and sizing. The workflow for the overall method, including intermediary steps can be seen on figure 2.

A. Technology Used

The program was developed and tested using Python 3.12.2 in a MacOS Ventura and Windows 10 environment. The python implementation of OpenCV (opencv-python 4.9.0.80) was used for image processing and non-text related computer

vision steps, while Pytesseract 0.3.10 was used for character and text recognition. The border following algorithm used is implemented by the OpenCV function findContours [9]. Wireframe images generated for the input of this algorithm were created using Figma, and screenshot using built-in tools to convert the wireframe design to an image usable for the algorithm. React and Tailwind were also used to assist in displaying the generated HTML/CSS.

B. Text Detection and Extraction

Tesseract OCR performs best on a noiseless black and white image [2]. As the input images are expected to be wireframe design screenshots, both of these requirements can be satisfied in a single gray scale conversion step [15]. Performing additional image processing steps were found to be ineffective to improving the accuracy of character recognition.

To begin, adaptive thresholding is performed on the image to gather sets of correlated pixels. These can be calculated from equation 2, where weights ω are the probabilities of pixels separated by threshold value t with variance σ^2 [1].

$$\sigma_b^2(t) = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2 \tag{2}$$

where

$$\mu_0(t) = \sum_{i=0}^{t-1} \frac{p(i)}{\omega_0(t)} \tag{3}$$

$$\mu_1(t) = \sum_{i=t}^{L-1} \frac{p(i)}{\omega_1(t)} \tag{4}$$

These blobs are then sorted based on candidate chop points, which are concave vertices of an approximation of a character outline [2]. To convert the detected elements into characters, distance to predicted prototype lines from each blob is computed by using the weighted distance d_f with an angle to the prototype line θ [1].

$$d_f = d^2 + \omega\theta^2 \tag{5}$$

$$E_f = \frac{1}{1 + kd_f^2} \tag{6}$$

$$d_{\text{final}} = 1 - \frac{\sum_f E_f + \sum_p E_p}{N_f + \sum_p L_p} \tag{7}$$

This distance is then passed into a KNN learning algorithm for character classification, which is then used as a second pass over the page to catch any missed chracters [1]. While this algorithm can detect full words or sentences, for this use case it was found to over detect the bounding box, or incorrectly identify wireframe elements as sections of a word. The effect of this can be observed in figure 3.

This over-detection of data is primarily an issue due to the next steps of the proposed algorithm, where all text elements must be removed before performing object detection [3]. If the bounding box of detected text elements are over estimated, there is the potential for sections or entire wireframe elements to be removed from the image. One important caveat from

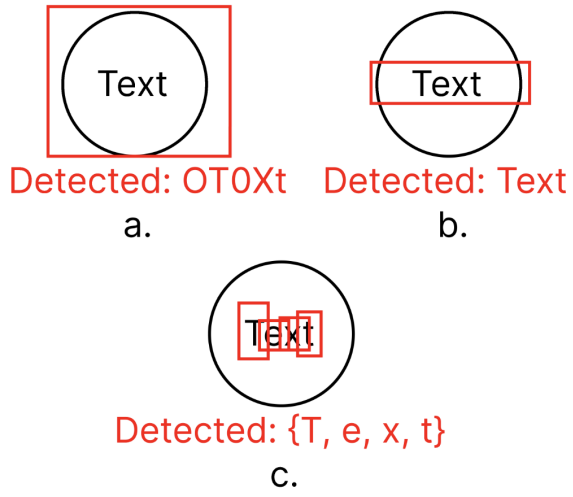

Fig. 3. Tesseract detecting wireframe elements as characters (a), Tesseract over detecting bounding box (b), Tesseract character recognition only (c)

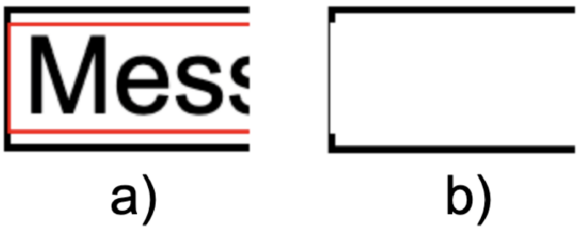

Fig. 4. Oversized text bounding box (a) eroding wireframe element section (b)

using this method is the requirement for further processing to convert the characters back into their original strings of text. A benefit of this method is that wireframe designs do not typically include multi-line strings of text [18], and therefore this algorithm treats strings as single lined. The characters can then be trivially sorted into their corresponding strings, and set the bounding box of each string to minimally match the bounding box of the overall text. Using this method was found to increase the accuracy of object detection in the next step by removing characters from the image that would otherwise be classified as contours.

C. Object Detection and Classification

Deep learning based solutions, such as Faster-RCNN, were not used for this solution. The requirements for input image standardisation meant that the input images must be constructed in a specific way and tightly coupled with the wireframe design tool [14].

Border following is a computer vision based algorithm for detecting shapes in an image [13]. This technique works by identifying potential borders in a processed image, and following the discovered contour to identify a shape. Image elements

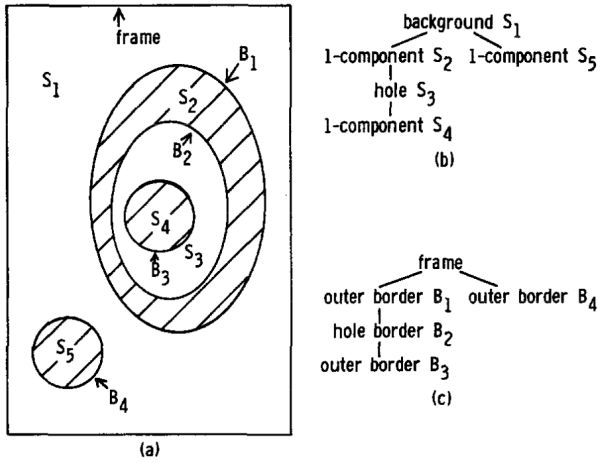

Fig. 5. Border following surroundness among connected components (b) and among borders (c). [13]

that have a pixel path between them are considered connected components, which can be classified by the algorithm into distinct objects [16], and are identified using equation 1. The border following algorithm then follows the first detected pixel for each object in a clockwise or counter-clockwise direction to detect the border of each shape [13]. While following the border, the pixels determined to be the border of the shape are saved as the shape contour [16], which is the output of the algorithm. The choice of direction is often decided by the placement or occurrence of surrounding pixels belonging to the detected object.

The steps for this algorithm can be seen below [7], which defines a border represented by pixels $P_0 ... P_{n-2}$.

- Find starting pixel P_0 of a region border, and previous move direction *dir*.
- If *dir* = 0 the border is 4-connectivity, else *dir* = 7 the border is detected in 8-connectivity.
- Begin the search of the neighbourhood.
  - (*dir* + 3) *mod* 4
  - If *dir* even: (*dir* + 7) *mod* 8
  - If *dir* odd: (*dir* + 6) *mod* 8

  The first similar pixel found is a new boundary element P_n. Update *dir*.
- Repeat algorithm until $P_n = P_1$ and $P_{n-1} = P_0$.

The detected shape and border elements are further processed to ensure accurate and distinct border detection [13]. The border following algorithm is sensitive to noise in the input image, and blurring and thresholding the image has been found to improve the accuracy of contour detection [4].

Further processing is then required to remove misclassified identified objects. Shapes with a small total area are immediately discarded, as these are usually unwanted artifacts from prior image processing steps. Additionally, the number of sides a detected contour has is analysed, and only shapes that are boxes or circles are saved. This is to simplify the HTML generation, and can be done due to the majority of shapes

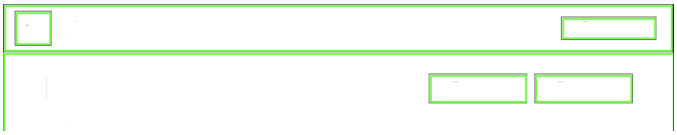

Fig. 6. Outlines of detected objects in wireframe design after text removal.

on any given wireframe design being classified as a rectangle or circle [20]. Another important processing step is to remove all duplicate shapes detected by the algorithm. Because the border following algorithm detects contours, shape outlines in the wireframe design can cause the algorithm to detect the outside and inside of the border as separate shapes [4].

An important side note is the process for identifying the effectiveness of the overall solution. This paper analyses the effectiveness of the solution using F1 scores to compare against previous literature. The F1 score is the weighted average of the precision and the recall of the solution [15]. This involves looking at true positives (TP), false positives (FP), and false negatives (FN).

$$F1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{8}$$

D. HTML/CSS Generation

The final step is to generate an HTML/CSS webpage based on the detected objects in the image. For the purposes of this algorithm, text and shapes are treated the same. All detected objects are sorted into an array based on their size. This means that the first element of this array is the root of the HTML tree, which is the overall window size. The algorithm then steps through the array in order and adds elements to the tree based on the smallest sized parent that can fit the object based on positioning. One important note in this step is to include a threshold value for sizing and positioning, as slight differences in the size and position of detected objects that were detected in prior steps may lead to invalid object placement in the tree.

Listing 1. Object Hierarchy Construction

```
objects = detected_objects.sort()
global_parent = objects[0]

for each object in objects[1:]:
    parent =
    global_parent.find_parent(object)
    parent.add_child(object)

global_parent.sort_by_y()
```

The generated tree can then easily be parsed to HTML based on the classification of the object. For this implementation, all sizing and positioning is fixed based on the detected size in the input image. This gives the benefit of replicating the wireframe design, but fails to be a responsive design and adapt to different screen sizes.

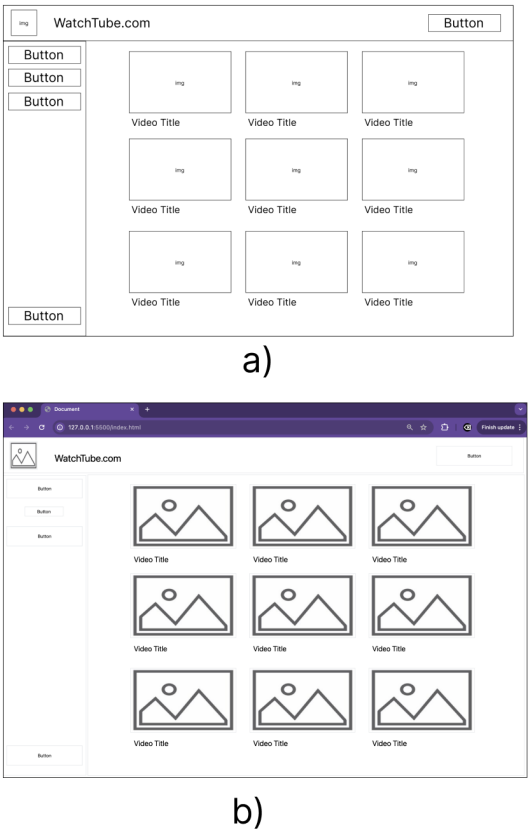

Fig. 7. Screenshot of generated HTML page (b) from input image demo7.png (a).

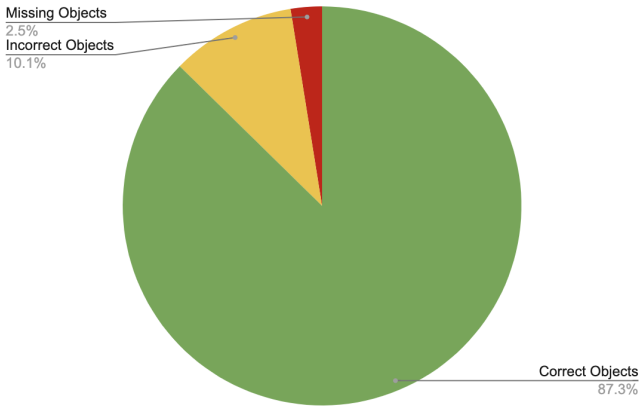

Fig. 8. Object detection and classification accuracy.

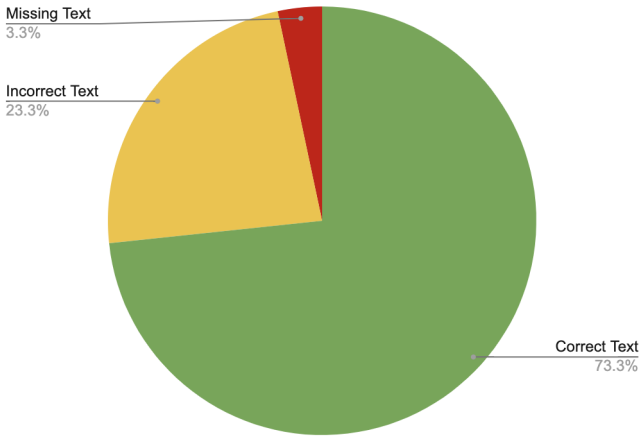

Fig. 9. Text detection and classification accuracy.

IV. Results

This method was tested using 8 input wireframe images generated on Figma, taking on average just under 500 milliseconds for an input image to generate HTML. The accuracy of this approach was calculated from the accuracy of the object detection and text detection. The full tabulated results can be found in appendix I. An input and output for this method can be observed in Figure 7, and the accuracy was recorded from the output based on the input image.

For this report, images, buttons, and boxes are considered to be objects for these results. An important note here is that object classification requires text classification accuracy, as image and button components are identified by the algorithm as ”img” and ”button” respectively. This method only misclassified an image or button component 2.5% of the time due to the text detection, which matches the average error rate for text detection from GUI elements [15].

10.1% of all objects were incorrectly classified by the algorithm, which were elements generated with improper sizing or duplication. Of this result, 75% of these were due to incorrect sizing, with the remaining being incorrectly duplicated. This result gives us an F1 score for the object detection of 0.9324, which is drastically higher than prior research that only managed to get an F1 score of 0.524 when using UIED [15].

Text detection had a higher error rate with just 73.3% being correctly detected, which is below the expected from the background literature that managed to achieve 91.67% accuracy [11]. In 3.3% of these cases the text was completely missed by the algorithm, and the remaining issues were mostly misidentified words, with just under 15% being an issue with font sizing. While there is a larger number of misidentified strings, for the expected wireframe output of this method, this does not have a huge effect. Wireframe designs typically only contain placeholder text [8], so any incorrectly detected strings may be immediately changed after HTML generation.

V. Limitations

The calculated F1 score of 0.9324 can not be directly compared to the implementations in the background literature. In most of these cases, their deep learning approach was to tackle GUI element generation from an existing design, which is drastically different to the wireframe input images accepted by this approach. This method fails to extend itself

beyond the use-case of wireframe image inputs, and further research in this space should focus on deep learning solutions to address this. This method also requires strict rules for wireframe design. Image and button elements must be labelled as such in the wireframe creation step, and placeholder icons, images, or other designs in the image will cause this approach to rapidly break down.

This method also fails to handle a combination of designs for one screen, and can't implement actions, behaviours, or responsive actions for a webpage. Additionally, the text detection accuracy falls short of other solutions, with only 73.3% true positives, compared to an expected 91.56% in background literature [11].

This method was only tested using computer generated wireframe designs, however physical wireframe designs, such as those made on whiteboards or paper, are also very common [6]. It is unclear the effectiveness of this solution on these other types of wireframe designs.

VI. Conclusion

This paper proposed a method to generate HTML/CSS webpages based on pictures of wireframe designs. Overall, this method managed to achieve 87.3% accuracy with object detection, and 73.3% accuracy for text. Using this method could be an easy way for designers to rapidly prototype different designs and view them in their expected web environment. This method is more accurate at detecting position and sizing than prior methods, with a higher F1 score of 0.9324 compared to 0.524 in a UIED deep learning approach [15], however this method rapidly breaks down when more complicated wireframe designs are used as the input.

A. Future Research

The object detection and classification section of this method is decoupled from the frontend GUI generation. This means that changing the frontend framework, or making the CSS responsive would be relatively simple.

Extending this method to support multiple input images of different states of a page that converge to a single output page could allow for a powerful tool that would be essential to all web developers. Future research could also include adding a middle deep learning step, that could identify and remove logos, images, icons, or other elements that aren't supported by this method, before processing the image with this method as before. This could allow for a much more robust tool that isn't limited in the way the wireframes are constructed.

Future research into plugin construction could improve the usability of this method. Turning this method into a plugin for software based design tools, such as Figma, would allow the user to skip the image generation step and generate HTML straight from their created designs.

References

[1] S Akhil. An overview of tesseract ocr engine. In *A seminar report. Department of Computer Science and Engineering National Institute of Technology, Calicut Monsoon*, 2016.

[2] Sahil Badla. Improving the efficiency of tesseract ocr engine. 2014.

[3] Jieshan Chen, Mulong Xie, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, and Guoqiang Li. Object detection for graphical user interface: old fashioned or deep learning or a combination? In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 1202–1214, New York, NY, USA, 2020. Association for Computing Machinery.

[4] Pengwen Dai, Sanyi Zhang, Hua Zhang, and Xiaochun Cao. Progressive contour regression for arbitrary-shape scene text detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7393–7402, 2021.

[5] Thomas Derleth. Corporate component and service libraries—a concept for creating, maintaining and managing a company-specific user interface component library in the field of frontend web development 12. 2018.

[6] Sidong Feng, Minmin Jiang, Tingting Zhou, Yankun Zhen, and Chunyang Chen. Auto-icon+: An automated end-to-end code generation tool for icon designs in ui development. *ACM Trans. Interact. Intell. Syst.*, 12(4), nov 2022.

[7] T.D. Haig and Y. Attikiouzel. An improved algorithm for border following of binary images. In *1989 European Conference on Circuit Theory and Design*, pages 118–122, 1989.

[8] Thisaranie Kaluarachchi and Manjusri Wickramasinghe. A systematic literature review on automatic website generation. *Journal of Computer Languages*, 75:101202, 2023.

[9] opencv. opencv. https://github.com/opencv/opencv/blob/master/modules/imgproc/src/contours.cpp#L1655, 2024.

[10] Everistus Zeluwa Orji, Ali Haydar, İbrahim Erşan, and Othmar Othmar Mwambe. Advancing ocr accuracy in image-to-latex conversion—a critical and creative exploration. *Applied Sciences*, 13(22):12503, 2023.

[11] AS Revathi and Nishi A Modi. Comparative analysis of text extraction from color images using tesseract and opencv. In *2021 8th International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 931–936. IEEE, 2021.

[12] Ray Smith. An overview of the tesseract ocr engine. In *Ninth international conference on document analysis and recognition (ICDAR 2007)*, volume 2, pages 629–633. IEEE, 2007.

[13] Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.

[14] Thomas D. White, Gordon Fraser, and Guy J. Brown. Improving random gui testing with image-based widget detection. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2019, page 307–317, New York, NY, USA, 2019. Association for Computing Machinery.

[15] Mulong Xie, Sidong Feng, Zhenchang Xing, Jieshan Chen, and Chunyang Chen. Uied: a hybrid tool for gui element detection. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1655–1659, 2020.

[16] Mitsuru Yamada and Kazuo Hasuike. Document image processing based on enhanced border following algorithm. In *[1990] Proceedings. 10th International Conference on Pattern Recognition*, volume 2, pages 231–236. IEEE, 1990.

[17] Jiaming Ye, Ke Chen, Xiaofei Xie, Lei Ma, Ruochen Huang, Yingfeng Chen, Yinxing Xue, and Jianjun Zhao. An empirical study of gui widget detection for industrial mobile games. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, page 1427–1437, New York, NY, USA, 2021. Association for Computing Machinery.

[18] Yudho Yudhanto, Wahyu Mangantasno Pryhatyanto, and Winita Sulandari. Designing and making ui/ux designs on the official website with the design thinking method. In *2022 1st International Conference on Smart Technology, Applied Informatics, and Engineering (APICS)*, pages 165–170. IEEE, 2022.

[19] Young-Sun Yun, Jinman Jung, Seongbae Eun, Sun-Sup So, and Junyoung Heo. Detection of gui elements on sketch images using object detector based on deep neural networks. In Seong Oun Hwang, Syh Yuan Tan, and Franklin Bien, editors, *Proceedings of the Sixth International Conference on Green and Human Information Technology*, pages 86–90, Singapore, 2019. Springer Singapore.

[20] Melissa Zhang. *Speeding Up the Prototyping of Low-Fidelity User Interface Wireframes*. PhD thesis, 2022.

	Total Objects	Correct Objects	Incorrect Objects	Missing Objects	Total Text	Correct Text	Incorrect Text	Missing Text
demo1.png	13	11	2	0	10	3	6	1
demo2.png	5	5	0	0	3	3	0	0
demo3.png	7	5	1	1	3	3	0	0
demo4.png	8	8	0	0	14	7	7	0
demo5.png	9	8	1	0	9	8	1	0
demo6.png	8	6	1	1	9	8	0	1
demo7.png	17	16	1	0	10	10	0	0
demo8.png	12	10	2	0	2	2	0	0

TABLE I

APPENDIX: OBJECT AND TEXT RECOGNITION ACCURACY RESULTS.

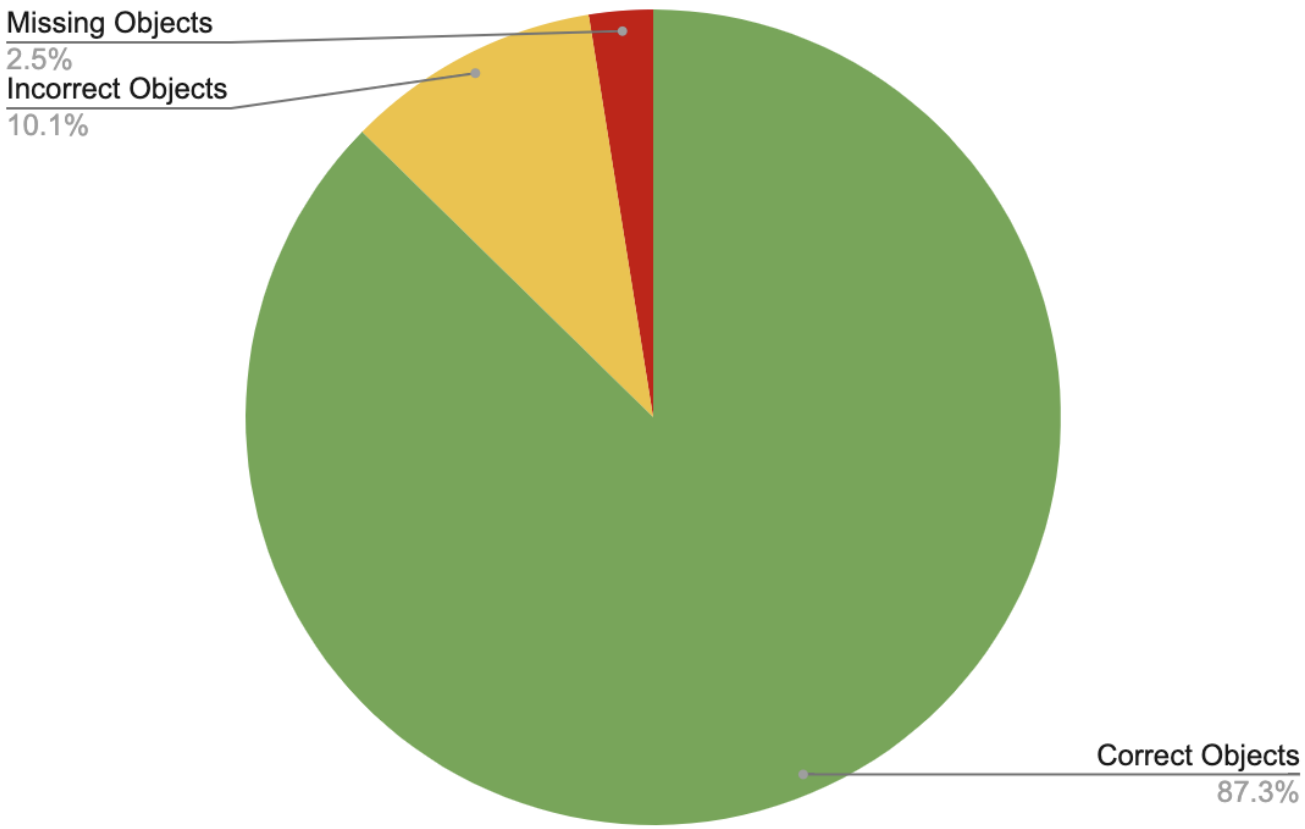

Fig. 10. Appendix: Object detection and classification accuracy.

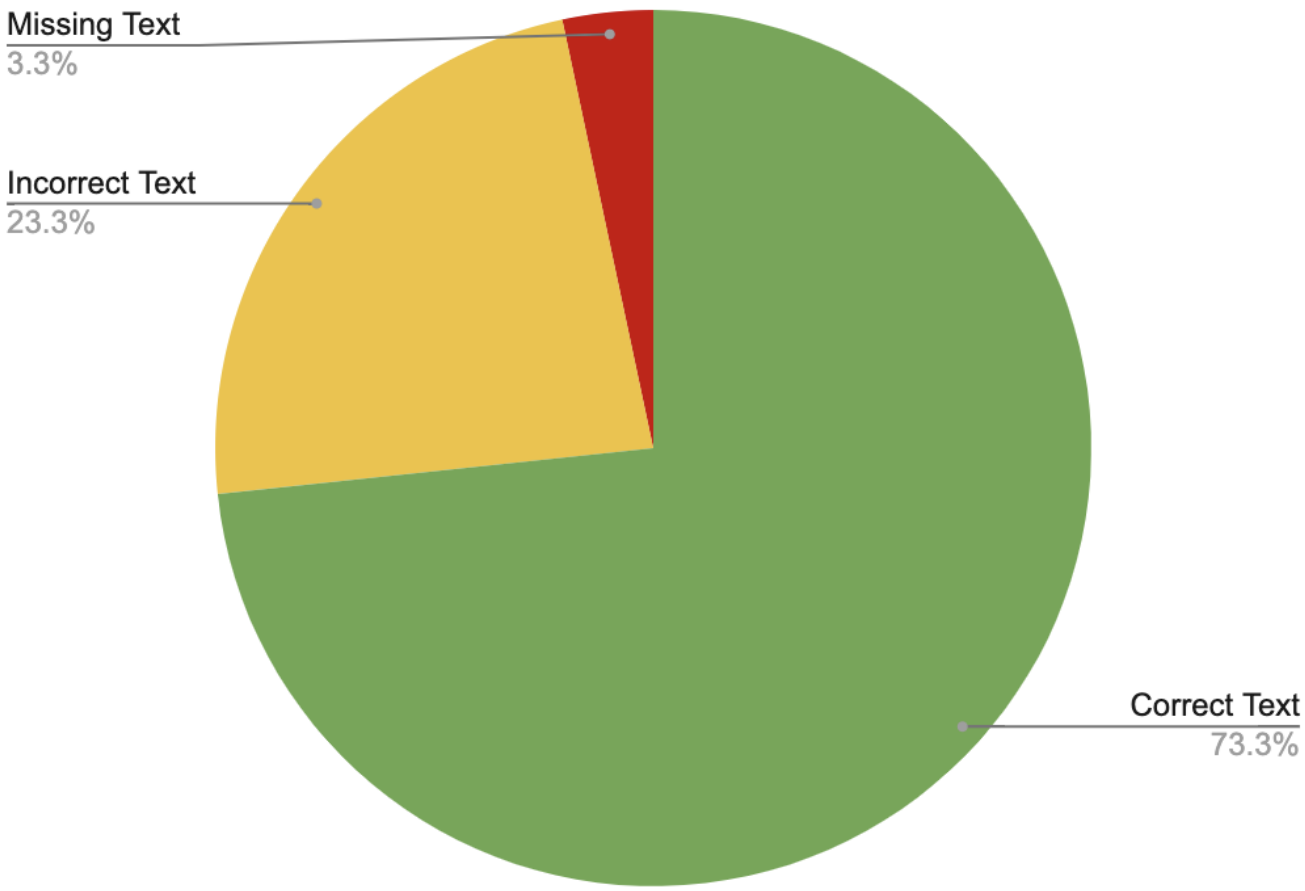

Fig. 11. Appendix: Text detection and classification accuracy.