

國立臺灣大學電機資訊學院電機工程學系

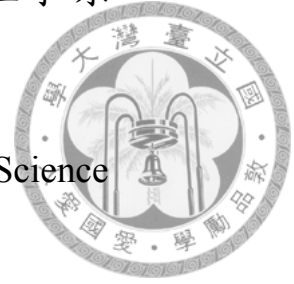
碩士論文

Department of Electrical Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis



基於視覺之使用者界面分割演算法

Vision Based User Interface Segmentation Algorithm

陳奕安

Yi-An Chen

指導教授：王勝德博士

Advisor: Sheng-De Wang, Ph.D.

中華民國 107 年 7 月

July, 2018





## 誌謝

論文得以順利完成，最先要感謝的是指導教授王勝德教授，在學術思想與邏輯上給了我相當大的啟發，沒有您悉心的指導，就沒有這篇論文的誕生；同時也要感謝口試委員雷欽隆教授、王鈺強教授、曾俊元教授精闢的見解，使論文更趨完善，在此敬申謝意。

在這段既艱辛又滿足的學術道路上，要感謝各位學長姊、同儕與學弟妹與摯友們：同實驗室的哲賢、冠廷，一同修課的方為、恩宇、彥鈞，同在研究所奮鬥、準備出國唸書與正在國外奮鬥的韻竹、佳蓓、恩禾、佳軒、子睿、秉民...等大學相識的夥伴們。因為有您們的鼎力相助，這篇論文才能完成。

最後，感謝我的父母與妹妹，是我在遭遇挫折時的避風港，您們的一路相挺與無私付出，使我沒有後顧之憂，得以專心求學，順利取得學位。

碩士生涯告一段落，隨之而來的是更大、更艱鉅的挑戰。請容許我貪心的期望，未來的日子中，希望還能有您們的支持與鼓勵，有您們在，我就能更堅定的走下去！





## 摘要

圖像分割廣泛的被應用於將視覺表現不同的資訊區隔出來。然而，在使用者界面相關的研究領域，不同使用環境發展出不同的演算法。本篇論文中，我們提出一個統合的、基於視覺的使用者界面分割演算法，能利用使用者界面的截圖即能計算其架構資訊。此演算法首先利用邊緣偵測以及一些定義好的邏輯偵測出使用者界面上的基本元素：方塊、線段、以及圖形輪廓。接著，我們定義一個計算兩元素距離的函數以及一個閾值選擇演算法來進行階層式分群。我們將此演算法運行在網頁界面以及手機應用程式上來評估其性能，並分析評估過程中演算法常見的缺失。

關鍵字： 影像分割、使用者界面、人機界面、文件分析、跨平台整合





# Abstract

Segmentation is used broadly to differentiate the presentation of different kinds of information. However, methods on different user interface environments tend to develop their own algorithms for this process. In this paper, we propose a unified vision-based segmentation algorithm, called UISeg, that only uses screenshots to estimate structural information of the user interface. The algorithm first leverages edge detection and a set of heuristics to recognize discrete elements such as boxes, lines, and contours. Then, we define a pairwise distance function and a threshold selection algorithm for the hierarchical clustering process. We evaluate the performance of UISeg with screenshots of web pages and mobile applications. Also, we analyze common failure cases among them.

**Keywords:** segmentation, user interface, human-computer interaction, document analysis, cross-platform integration







# Contents

誌謝	iii
摘要	v
<b>Abstract</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Related Work</b>	<b>5</b>
2.1 Web Applications . . . . .	5
2.2 Mobile Applications . . . . .	6
2.3 Desktop Applications . . . . .	6
<b>3 Methodology</b>	<b>9</b>
3.1 Text Detection . . . . .	9
3.2 Contour Detection with Computer Vision Techniques . . . . .	11
3.3 Distance Model . . . . .	13
3.4 Hierarchical Clustering . . . . .	16
<b>4 Evaluation</b>	<b>21</b>
4.1 Performance Evaluation . . . . .	22

4.2	Failure Analysis . . . . .	23
4.2.1	Floating elements . . . . .	23
4.2.2	wrapping rows . . . . .	24
4.2.3	Invisible and highlight separators . . . . .	24
4.3	Conclusion . . . . .	24
<b>Bibliography</b>		<b>27</b>





# List of Figures

1.1	Overview of the <b>UISeg</b> algorithm. The three steps in <b>UISeg</b> : (1) text detection (2) contour detection (3) clustering. . . . .	3
3.1	Illustration of critical sections and separator effect. Left: The critical section and base distance of a and b. Right: Line 1 has separator effect greater than Line 2. . . . .	17
3.2	<b>UISeg</b> results on web, mobile, and desktop environments . . . . .	18
4.1	Failure cases. Top: floating elements. Middle: wrapping rows. Bottom: invisible separators. . . . .	26





# List of Tables

3.1	Detection Criteria . . . . .	13
4.1	Segmentation and Clustering Performance Metrics . . . . .	22





# Chapter 1

## Introduction

Desktop, mobile, and web platforms are the three most common user interface environments we face with in our daily life. With the advent of new technologies, lines between these environments are getting blurred. From developers' point of view, it would be easier to develop programs on one framework for different devices. React Native, for example, is one of the many attempts to bringing modern web techniques to mobile devices. Electron allows the development of cross-platform desktop applications with JavaScript, HTML, and CSS. On the other hand, application users turn to expect that the experience of individual services can be shared across platforms. This can be observed in the design of many well-known Internet services like Facebook, Spotify, Uber, etc. These phenomena suggest that there is a unifying method to model user interfaces in different platforms, and it shouldn't be based on environment-dependent features.

This paper introduces **UISeg**, a purely vision-based segmentation algorithms that receives a user interface screenshot and generates a segment tree with all nodes being visually and semantically coherent. The segment tree can be seen as the underlying structure of the screenshot image, and the coherence constraint means it is close to the mental model

humans unconsciously create when seeing the user interface. Also, **UISeg** treats all user interfaces the same. That is to say, the algorithm doesn't make assumption about characteristics that distinguish user interfaces, including design patterns, displayed languages, UI toolkits, operation systems, execution contexts (like browser engines). We validate the performance of **UISeg** by evaluating the algorithm with web page and mobile application screenshots. In both environments, **UISeg** achieves about 90% accuracy on segmentation and 75% on clustering.

The potential impact of **UISeg** is that it can be used as the backbone of systems that extract information from user interfaces. For example, studies on user interface test automation apply different techniques for mobile applications and web applications to interact with testing targets because they use features that are only available in one environment. With **UISeg**, UI testing process can be divided into a segmentation stage and an exploration stage, and we can utilize the outputs of **UISeg** to examine the user interface in a more generalized, human-centric way.

To summarize, we make the following contributions in this paper:

1. A modified segmentation method based on prior work. More features about the layout of the user interface are detected without using platform-dependent information.
2. A distance model that uses these features to calculate pairwise distance between regions. This model can be seen as an attempt to quantify the strength of linkage between regions in human minds. Also, Segment trees can be generated by using the model with a simple clustering and threshold selection algorithm.
3. The implementation of algorithms and the labeling tool are publicly available<sup>1</sup> to support further studies.

---

<sup>1</sup><https://gitlab.com/uiseg/uiseg>



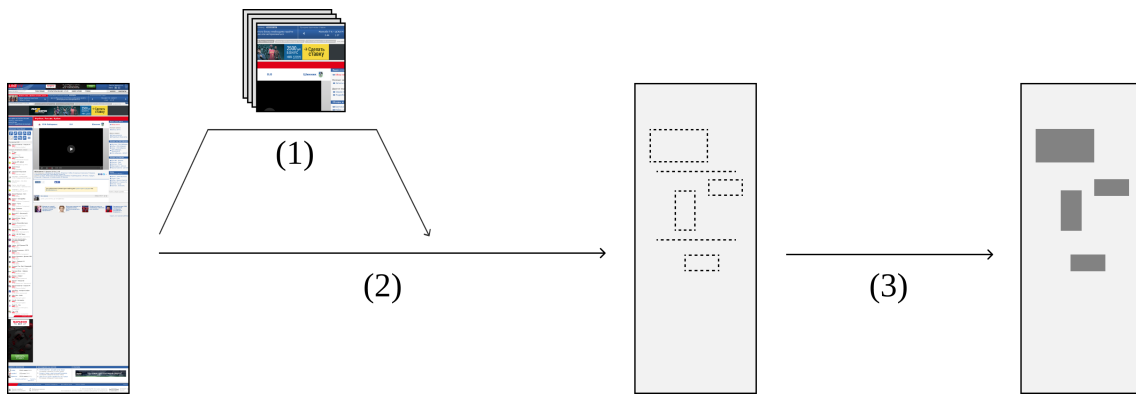


Figure 1.1: Overview of the **UISeg** algorithm. The three steps in **UISeg**: (1) text detection (2) contour detection (3) clustering.





## Chapter 2

### Related Work

Different problems require segmentation to differentiate useful information from the background. In this section, we discuss the existing works on three most commonly used UI environments. These works focus on different topics but all of them have segmentation involved, implicitly or explicitly.

#### 2.1 Web Applications

The Internet can be seen as a big decentralized, unnormalized data source. Data mining applications utilize web page segmentation methods to detect repeated structures and retrieve main contents. Several studies [4, 6, 7, 15] suggest that edge detection is useful to determine separability between content blocks. Zeleny et al. [24] defined a similarity function between content elements to perform clustering. It is worth mentioning that these vision-based algorithms are often not considered best approaches in the industry because structure-based algorithms [11, 12, 22] or hybrid approaches [2, 18, 21] tend to run faster than vision-based alternatives.

## 2.2 Mobile Applications

In recent years, there has been growing interest in transforming a UI screenshot into computer code. REMAUI [19] contains a set of heuristics to reverse-engineer Android user interfaces from screenshots. A series of recent studies [1, 5, 14] has indicated that machine learning can be useful to prototype software GUIs on Android system. Segmentation is utilized in these works implicitly to search for regions of interest (RoIs).



## 2.3 Desktop Applications

As the first and most familiar platform people working on, desktop is the most studied environment among the three. Besides the topics mentioned above, many researches focus on supporting interactions by analyzing the visual patterns of GUI, referred to as *direct pixel access*. The first work to explore this idea was examined in the 90's [16]. After that, novel tasks have been proposed in conjunction with studies in human-computer interaction fields. Dixon et al. implements various advanced interaction behaviors to validate their results on reverse engineering UI structure [9, 10]. Sikuli project [23] offers a set of visual scripting APIs for automating GUI interactions. These works provide their own knowledge about how to gain understanding of GUI on their targeted platforms, and segmentation is the key step in it.

While these works gain huge success in their fields, none of them provides an platform-independent solution, and part of the reason of that is because the segmentation process requires features only available in some platforms. Our work comes from the idea described in [19] that by detecting texts and elements jointly on user interfaces, one can effectively analyze user interfaces in a way very similar to how humans perceive them.

We also get a good insight from Zeleny *et al.*'s approach [24], which shows that a carefully designed pairwise similarity function is crucial for good segmentation results. We discuss the implementation details in the next section.







## Chapter 3

# Methodology

Our segmentation algorithm **UISeg** contains three main processing steps. The first step is a powerful deep learning-based text detection module that detects texts from a screenshot image in the paragraph level. Then, computer vision algorithms are used, together with the results of text detection, to create a valid set of contours that covers all elements on the screenshot image. Finally, the bounding box of these contours is fed into a clustering algorithm and a possible hierarchical segmentation of the user interface is generated.

### 3.1 Text Detection

Text detection on the paragraph level distinguishes texts from images and generate bounding boxes for each paragraph. Different from optical character recognition (OCR) techniques, text detection doesn't recognize every word in the input image. Although there have been many studies on OCR and existing OCR engines have been proven to be powerful and accurate in many cases, we argue that our text detection approach is more suitable than existing solid, open-sourced OCR engines like Tesseract [20] for our work because:

- we don't need character/word prediction nor their localization information. It is hard

to transform the output of the OCR engine to fit our need (paragraph localization information).

- OCR engines may not be language-independent. For example, Tesseract requires each language having one specific trained model to perform better, and the preparation can be time-consuming and not applicable if we don't have any assumption about the language shown on the input image.
- as discussed in [19] and [13], OCR would have poor results if text is arranged freely and combined with images, which is typically how user interfaces are designed. Our preliminary tests show that OCR tools fail to have comparable performance on the desktop screen, indicating that using these OCR tools will reduce stability of our algorithm. This is the most important reason why we develop our own model.

Based on the discussion above, we leverage the off-the-shelf object detection model Faster R-CNN [17] for our work. The implementation is modified from an online, open-sourced version<sup>1</sup> rather than the official one because it can be integrated into our codebase more easily. This model takes a 512x512, normalized RGB image as input and return  $(X_{\min}, Y_{\min}, X_{\max}, Y_{\max}, confidence)$  tuples for each paragraph it finds.

To train the model, we collect 3867 images from Alexa's top one million website list using browser automation technology. For each website, only one screenshot is captured either from the main page or a page selected by randomly clicking a link on the main page. This is to avoid any potential design bias between the main page and other pages. Paragraph localization labels are also collected when visiting the website. Then, these images are cropped into required dimension and fed into the model for training after standard data augmentation steps like random horizontal flipping, random scaling, and normalization.

---

<sup>1</sup><https://github.com/chenyuntc/simple-faster-rcnn-pytorch/>



We find out that using deep learning models not only avoids the drawbacks mentioned above, but has the following advantages:

- *Time efficiency.* While inference speed varies by the actual implementation, the running time of deep learning models is generally smaller than that of OCR engines.

This becomes more prominent when the input comes from a long, scrollable web page, which can be commonly seen in modern websites.

- *Generalization.* The model proves to be very robust, especially on the aspect of language and user interface environment. Although the training data come from website screenshots and most of display languages in the dataset is English, it performs well in other cases. Moreover, the accuracy can be improved by collecting more data from different sources without modifying the model.

The output tuples with confidence score greater than 0.7 are put into the computer vision process.

## 3.2 Contour Detection with Computer Vision Techniques

User interface screenshots differ from natural images in that they tend to have distinct block regions. These blocks may not have lines surrounding them to distinguish contents from the background, but the interesting thing is that humans can perceive these blocks instinctively and assign them semantically coherent meanings unconsciously.

Based on the above observation, we first apply edge detection algorithm to convert the original 3-channel screenshot into a binary single-channel edge map. Canny [3] is selected because of its great S/N ratio among various edge detection algorithms.

The following definitions describe roles we are going to mention in this step.



**Definition 1.** A **region**  $r$  on the feature map can be specified by a 4-tuple  $(r_{x1}, r_{y1}, r_{x2}, r_{y2})$ .

We say a region **covers** another one if the latter is inside the former.

**Definition 2.** A **cluster** is a set of non-overlapped regions  $\{r_1, r_2, \dots, r_n\}$  that may have semantically coherent meanings. It can also be seen as a region specified as  $(c_{x1}, c_{y1}, c_{x2}, c_{y2})$ ,

where

$$\begin{cases} c_{x1} = \min_r r_{x1} \\ c_{y1} = \min_r r_{y1} \\ c_{x2} = \max_r r_{x2} \\ c_{y2} = \max_r r_{y2} \end{cases}$$

**Definition 3.** A **box** is a special kind of region that covers a set of non-overlapped regions.

In the process of clustering, these non-overlapped regions will be merged into a cluster, and the box will be discarded.

**Definition 4.** A **line**  $l$  is specified using 4-tuple  $(l_s, l_e, l_a, \text{axis})$ ,  $\text{axis} \in \{H, V\}$ , indicating the start, end, anchor, the type of axis of the line. A line cannot overlap with any region but it can exist in a box.

These roles exist because we believe that this is how humans perceive user interface. Edge detection finds the boundaries of objects within the image and the resulting edges can be connected into contours, which have great possibilities to serve their purposes on the user interface. Based on what they serve for, we can categorize them into 1. regions that could be images or texts in the original screenshot, 2. boxes that can always be seen as a whole and separate their contents from the outside, 3. lines that represents dividers. The detailed criteria to detect these elements are shown in Table 3.1.

<b>Contour</b>	contour not detected as box; width > 5px & height > 10px
<b>Box</b>	contour area/bounding box area > 0.8; sum of children contour area/contour area > 0.65
<b>Horizontal Line</b>	morphological transformations; not covered by texts; length > 150px
<b>Vertical Line</b>	morphological transformations; not covered by texts; length > 100px

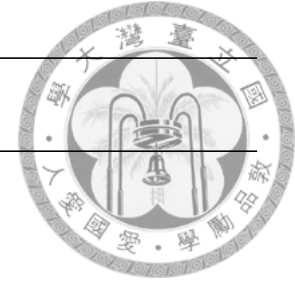


Table 3.1: Detection Criteria

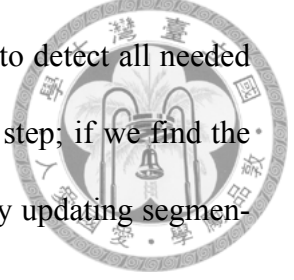
### 3.3 Distance Model

The regions detected in the last step are considered ‘content’ elements coming in a spatially flat structure, and the distance model define pairwise distance between any two of these elements plus clusters of them. It consists of two metrics: the base distance that is calculated by directly using the dimension information of regions, and the amplification factor that magifies the base distance. We believe that this mechanism resembles how humans perceive user interface. To put it more precisely, we observe that the distance model should have the following characteristics:

1.  $\text{distance}(a, b) > 0, \forall a, b$
2.  $\text{distance}(a, b) = \text{distance}(b, a)$
3.  $\text{distance}(a, b') > \text{distance}(a, b)$  if there are lines between  $a$  and  $b$  but there is not between  $a$  and  $b'$ .
4.  $\text{distance}(a, b) > \text{distance}(a, c)$  if  $a, b, c$  are arraged spatially in the order  $a \leftrightarrow c \leftrightarrow b$ .

Another important observation about distance calculation is that, we can ignore the raw pixel information of the original user interface, as opposed to how other vision-based

segmentation algorithms work, and only consider dimension information about the target elements in the distance calculation process. In other words, we try to detect all needed features in the segmentation step and process them in the clustering step; if we find the overall accuracy unsatisfactory, we can either detect more features by updating segmentation algorithm, or fine-tune the clustering method.



Features we get in the clustering process are boxes, horizontal/vertical lines, and regions storing in their own planes, which is defined below:

**Definition 5.** A **plane** is a collection of regions or lines that can be accessed in a spatial way. Two regions don't overlap with each other if they are in the different planes.

Among these features, the region plane is the core one in the proposed distance model and is used to calculate the base distance metric.

**Definition 6.** Let  $a = (a_{x1}, a_{y1}, a_{x2}, a_{y2})$ ,  $b = (b_{x1}, b_{y1}, b_{x2}, b_{y2})$  be two regions in the same plane with center  $(a_{xc}, a_{yc})$  and  $(b_{xc}, b_{yc})$ , respectively. We define **base distance** between  $a$  and  $b$  to be

$$bd = \max\left(\frac{w}{RI}, h\right), \quad (3.1)$$

where

$$\begin{cases} w = \max\left(|a_{xc} - b_{xc}| - \frac{(a_{x2}-a_{x1})+(b_{x2}-b_{x1})}{2}, 0\right) \\ h = \max\left(|a_{yc} - b_{yc}| - \frac{(a_{y2}-a_{y1})+(b_{y2}-b_{y1})}{2}, 0\right), \end{cases}$$

$RI$  is the **row inclication value** representing the tendency to merge regions in the same row rather than in the same column.

The proper row inclication value varies by user interface environments. We find that 3 is a proper value for a general use case.

Then, we introduce size difference and separator effect that compose amplification factor. The size difference take into consideration that we want to group lists of elements together, and they often comes in similar sizes.



**Definition 7.** The **size difference** of region  $a$  and  $b$  is

$$A_{\text{size}} = \frac{2}{1 + s_r}, \quad (3.2)$$

where

$$s_r = \min\left(\frac{\text{Area}(a)}{\text{Area}(b)}, \frac{\text{Area}(b)}{\text{Area}(a)}\right).$$

This function is chosen as it is a convex downward function passing through points  $(0, 2)$  and  $(1, 1)$ .

As for separator effect, we first define critical region between any two regions; then, lines within the critical region will make merging the two regions more difficult.

**Definition 8.** Using the notaion discussed above, we define **critical region**  $Q_{ab}$  to be the largest region within  $G_{ab} - a - b$ , where  $G_{ab} = (g_{x1}, g_{y1}, g_{x2}, g_{y2})$ ,

$$\begin{cases} g_{x1} = \min(a_{xc}, b_{xc}) \\ g_{y1} = \min(a_{yc}, b_{yc}) \\ g_{x2} = \max(a_{xc}, b_{xc}) \\ g_{y2} = \max(a_{yc}, b_{yc}) \end{cases}$$

In Figure 3.1 we show an example with two regions and the critical region of them.

**Definition 9.** We define **separator effect value** to be

$$A_{\text{sep}} = \prod_{l \in L \cap Q_{ab}} \frac{|l|}{\max(\text{dim}(a, b), |l|)}, \quad (3.3)$$



where  $|l|$  denotes the length of line  $l$ , and

$$\text{dim}(a, b) = \begin{cases} \min(\text{width}(a), \text{width}(b)) & , \text{ if Axis}(l) \text{ is } W \\ \min(\text{height}(a), \text{height}(b)) & , \text{ otherwise.} \end{cases}$$

Finally, the pairwise distance between any two regions is defined below.

**Definition 10.** The **pairwise distance** of any two regions  $a$  and  $b$  is defined as

$$\text{distance}(a, b) = \begin{cases} bd^{A_{\text{sep}} \times A_{\text{size}}} & , a, b \text{ are in the same box} \\ \infty & , \text{ otherwise.} \end{cases} \quad (3.4)$$

The use of exponential function in our distance model may seem unintuitive at first glance, but actually it is required for better clustering performance, which we will discuss later.

### 3.4 Hierarchical Clustering

The goal of hierarchical clustering is to create a segment tree from the elements detected in the segmentation step, with each node in the segment tree a visually and semantically coherent segment. We find that by using the distance model defined in the previous section, clustering can be done iteratively with a simple threshold selection method.

By iterating from the smallest box to the biggest box, we can ensure that there is not

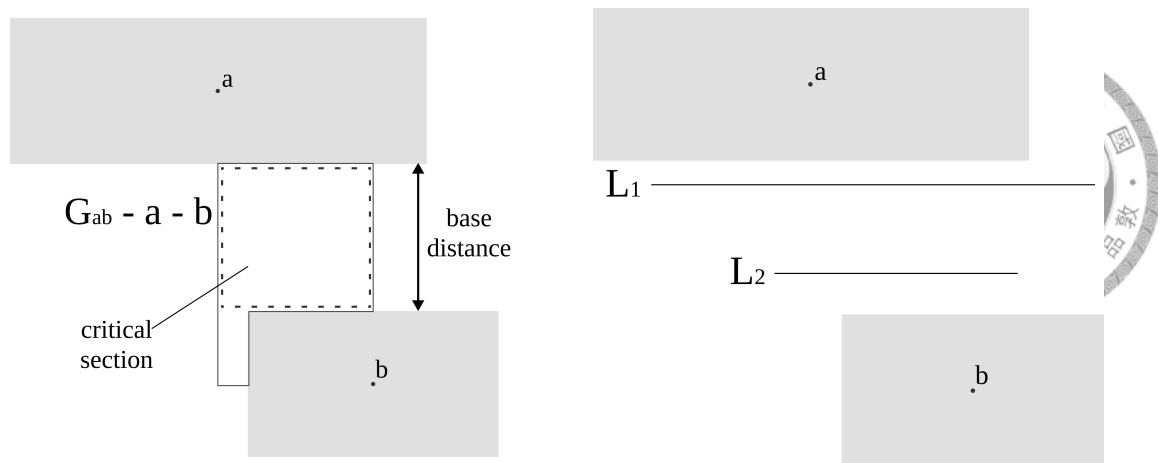


Figure 3.1: Illustration of critical sections and separator effect. Left: The critical section and base distance of a and b. Right: Line 1 has separator effect greater than Line 2.

any box within candidates in each iteration. Then, all boxes will be converted to clusters by the end of the algorithm.

The last thing about the clustering algorithm is the heuristic of determining a good threshold in Line 5 of the algorithm. There is a trade-off about the selection of threshold: if the threshold is too small, we only merge two regions in each iteration, and the created cluster might not be a valid coherent segment; on the other hand, if the threshold is too large, the final segment tree would contain too few nodes, and the algorithm would become useless.

The good news is that, because of the exponential function used to calculate pairwise distance, the distribution of distance has a upward-sloping, concave upward curve, and small distance tends to be grouped together. Therefore, selecting a threshold becomes easy: we sort the distance sequence and compute its discrete-time derivative. The first index at which the derivative increases is used to find the threshold in the original sequence.

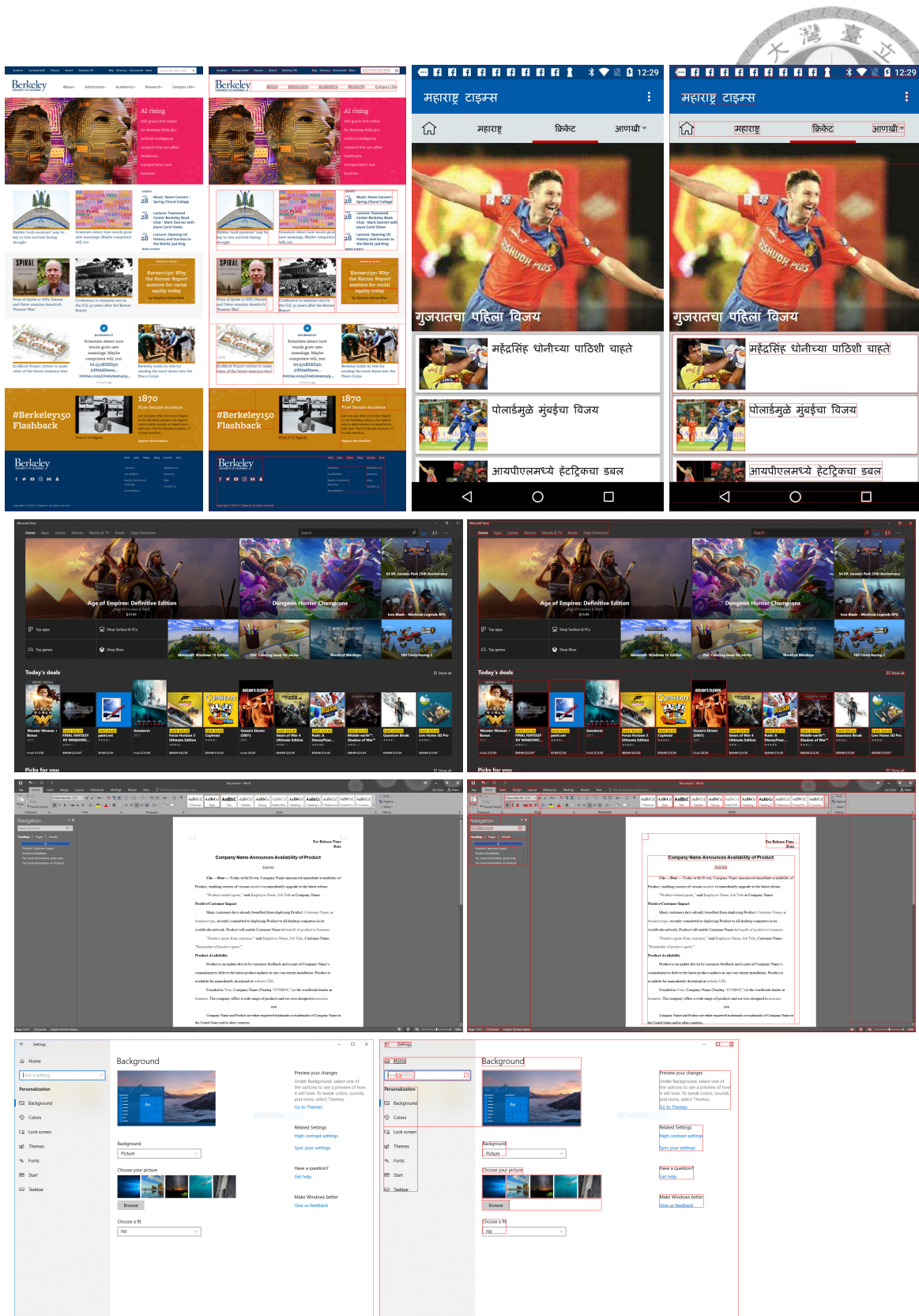


Figure 3.2: UISeg results on web, mobile, and desktop environments





---

**Algorithm 1 UISeg Algorithm**

---

**Input:** regions, boxes

**Output:** a segment tree of the user interface

- 1: Start from the smallest box to the biggest box
  - 2: **if** there is only one region covered by the box **then**
  - 3:   Remove the box. Go to Line 1 to process the next box.
  - 4: **end if**
  - 5: Add all regions covered by the box into a new plane. Select a clustering threshold.
  - 6: **while** there are some region pairs with pairwise distance  $<$  threshold **do**
  - 7:   Remove them from the plane.
  - 8:   Merge connected components into clusters and add these clusters into the plane.
  - 9:   Update pairwise distance information.
  - 10:   Select a new clustering threshold.
  - 11: **end while**
  - 12: Remove regions in the plane and merge them into a cluster.
  - 13: Remove the box. Go to Line 1 to process the next box.
-





## Chapter 4

# Evaluation

To examine the result of **UISeg** algorithm, we randomly select 100 web page screenshots (from previous mentioned Alexa's website list excluded those used to train the text detection model) and 100 mobile application screenshots from [8]. Since it would be too much efforts put to collect desktop UI with source code at large scale, we only evaluate them by showing examples. For web and mobile UI, We run **UISeg** algorithm on these images and mark 20% of nodes in the result segment trees with two groups of correctness labels. The first group of labels (*perfect*, *fair*, *bad*) measures how humans feel about the bounding box of a result segment by only looking at the segmented image; all surrounding pixels and child segments are not considered. The second group (*perfect cluster*, *missing children*, *additional children*, *bad cluster*) takes parent-children relationship into account and represents the overall performance of **UISeg**. Note that we don't change any hyperparameter in these two environments.

In the remaining section, we will first discuss the labeling result of **UISeg** algorithm on web and mobile app environments. Then, we analyze common failure cases observed during the labeling process and show examples of them.

Web					
avg. segments per page	perfect cluster	missing children	additional children	bad cluster	total
perfect	95.8 (79.0%)	9.3 (7.7%)	4.8 (4.0%)	0.7 (0.5%)	<b>110.7</b> (91.2%)
fair	1.9 (1.5%)	1.6 (1.4%)	1.3 (1.1%)	0.7 (0.6%)	5.5 (4.5%)
bad	0.2 (0.2%)	0.6 (0.5%)	0.2 (0.2%)	4.2 (3.4%)	5.2 (4.2%)
total	<b>97.9</b> (80.7%)	11.6 (9.5%)	6.3 (5.2%)	5.5 (4.5%)	121.3 (100.0%)
ground truth					164.8 (135.86%)
Mobile					
avg. segments per page	perfect cluster	missing children	additional children	bad cluster	total
perfect	35.0 (75.0%)	4.7 (9.9%)	1.4 (3.1%)	0.2 (0.5%)	<b>41.4</b> (88.6%)
fair	1.1 (2.2%)	0.6 (1.3%)	0.7 (1.4%)	0.2 (0.3%)	2.5 (5.2%)
bad	0.1 (0.1%)	0.1 (0.1%)	0.2 (0.4%)	2.6 (5.6%)	2.9 (6.2%)
total	<b>36.1</b> (77.3%)	5.3 (11.3%)	2.3 (4.9%)	3.0 (6.4%)	46.7 (100.0%)
ground truth					39.2 (83.9%)

Table 4.1: Segmentation and Clustering Performance Metrics

## 4.1 Performance Evaluation

The result of the segment evaluation is shown in Table 4.1. Each cell in the table contains the number of occurrences marked with a pair of evaluation labels. It should be noted that the ground truth cell is calculated using source code but not necessarily reflect how humans feels about these screenshots. The result shows that **UISeg** has consistent performance in these environments, as (`perfect`, `perfect cluster`) and (`perfect`, `total`) reports about 75% and 90% of all detected segments, respectively. The clustering step performs slightly better in web environment (statistics in the `total` row). We conclude that this is because mobile applications tends to have less lines by design, and this affects the caculation of pairwise distance. Also, since the text detec-

tion model is only trained on web page screenshots, it performs less satisfying and worsen the performance on mobile environment.

There're some more numbers worth mentioning in this table. The (perfect, missing children) cell indicates that some clusters of good segmentation results contain less children than humans expect. This is the direct result of a rigid constraint of threshold selection. If we choose a looser threshold selection algorithm, the error would be reduced, but less segments would be detected. Another observation is that the number in (perfect, total) cell is greater than the ground truth in the mobile environment but not in the web environment. This implies that the algorithm captures a set of segments different from what UI programmers would expect, and that there's significant dissimilarity between implementation languages of user interface, which is why we design a purely vision-based, language-independent segmentation and clustering algorithm to have a better understanding about user interface.



## 4.2 Failure Analysis

We analyze the algorithm output labeled in the previous section to find out the cases in which **UISeg** fails to work. We show the examples in Figure 4.1 and list the common failure cases below:

### 4.2.1 Floating elements

**UISeg** assumes that input images come statically as a tree structure. But floating elements such as context menus or sticky buttons would break the assumption, so the distance model would be affected and clusters containing these elements would have a false bounding box.

## 4.2.2 wrapping rows

When there is no room for fixed-width elements to stay at the same row, they wrap: some items will jump to the second row and become closer to only a few items in the first row. Then, error might occur when **UISeg** clusters these items with wrapped elements before merging all elements as a whole. The algorithm fails in this case because it doesn't recognize repeating structure and relies on dimension statistics to infer repeating structure.



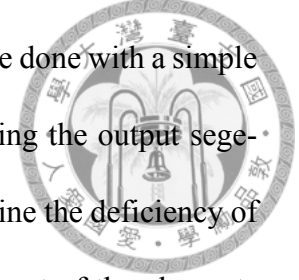
## 4.2.3 Invisible and highlight separators

When seeing a table, humans determine it to be row-major or column-major by perceiving repeated structure. Therefore, some designer would choose not to put separators between rows/columns for design considerations. This is not the case for **UISeg** because the algorithm only uses *RI*, the row indication value, to balance the importance between the roles of row and column. Moreover, separators are sometimes be put under headers to differentiate them from following contents. We call this kind of separators 'highlight separators'. In this case, however, **UISeg** would group headers with the contents above them. These two kinds of separators are not uncommon and account for a great amount of error, especially in the web environment.

## 4.3 Conclusion

In this paper, we have presented an approach for extracting UI structures in different environments. The environment-independent generalizability this approach provides is important because a new trend on user interface unification is observed by not only users but also designers and developers who create user interface. Compared with prior work, our

scheme emphasizes the roles of box and separator for accurately calculating pairwise distance between elements in the segmentation stage, so clustering can be done with a simple threshold selection method. We validated the performance by judging the output segmentation tree on the node level and conducting failure analysis to determine the deficiency of our algorithm. The results showed that our algorithm can capture most of the elements on the user interface, but still needs improvement on cases when capturing repeated structure is needed. We believe that this work offers possible solutions to integrate ideas from different environment domains together.





Help Me Choose

## Need more help?

If you still have questions about QuickStart, you'll find the answers in our Help and Support section.

[Help and Support](#)

© 2002-2018 T-MOBILE USA, INC.

[Instagram](#) [Facebook](#) [Twitter](#) [YouTube](#) [ESPAÑOL](#)

[ABOUT](#) [INVESTOR RELATIONS](#) [PRESS](#) [CAREERS](#) [DEUTSCHE TELEKOM](#)  
[PUERTO RICO](#)

[PRIVACY POLICY](#) [INTEREST-BASED ADS](#) [PRIVACY & SECURITY RESOURCES](#)  
[CONSUMER INFORMATION](#) [PUBLIC SAFETY/911](#) [TERMS & CONDITIONS](#)  
[TERMS OF USE](#) [ACCESSIBILITY](#) [OPEN INTERNET](#)



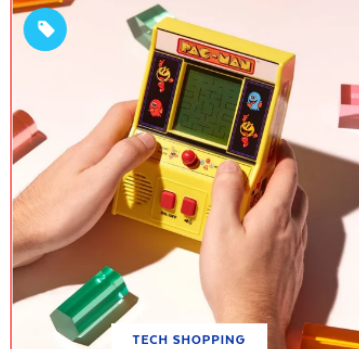
 <p>DIGITAL LIFE</p> <p>This 1 Accessory Makes Any iPhone Take the Best Photos Ever – No Upgrade Required</p> <p><a href="#">CHELSEA ADELAINE HASSLER</a> 6 Days Ago</p>	 <p>DIGITAL LIFE</p> <p>The Best iPhone X Cases of 2018</p> <p><a href="#">CHELSEA ADELAINE HASSLER</a> 6 Days Ago</p>	 <p>TECH SHOPPING</p> <p>35 Unique and Useful Gifts That He'll Love This Valentine's Day</p> <p><a href="#">KRISTA JONES</a> 2 Weeks Ago</p>
---	---	--

Figure 4.1: Failure cases. Top: floating elements. Middle: wrapping rows. Bottom: invisible separators.

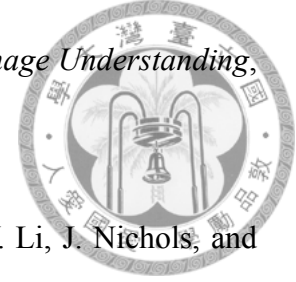




# Bibliography

- [1] T. Beltramelli. pix2code: Generating code from a graphical user interface screenshot. *CoRR*, abs/1705.07962, 2017.
- [2] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. Vips: a vision-based page segmentation algorithm. 2003.
- [3] J. Canny. A computational approach to edge detection. In *Readings in Computer Vision*, pages 184–203. Elsevier, 1987.
- [4] T.-H. Chang, T. Yeh, and R. Miller. Associating the visual representation of user interfaces with their internal structures and metadata. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 245–256. ACM, 2011.
- [5] C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu. From ui design image to gui skeleton: A neural machine translator to bootstrap mobile gui implementation. In *The 40th International Conference on Software Engineering, Gothenburg, Sweden*. ACM, 2018.
- [6] M. Corner, R. Mann, K. Moffatt, and R. Cohen. Towards an improved vision-based web page segmentation algorithm. In *Computer and Robot Vision (CRV), 2017 14th Conference on*, pages 345–352. IEEE, 2017.

- [7] M. Cormier, K. Moffatt, R. Cohen, and R. Mann. Purely vision-based segmentation of web pages for assistive technology. *Computer Vision and Image Understanding*, 148:46–66, 2016.
- [8] B. Deka, Z. Huang, C. Franzen, J. Hibschan, D. Afergan, Y. Li, J. Nichols, and R. Kumar. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual Symposium on User Interface Software and Technology*, UIST '17, 2017.
- [9] M. Dixon and J. Fogarty. Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1525–1534. ACM, 2010.
- [10] M. Dixon, D. Leventhal, and J. Fogarty. Content and hierarchy in pixel-based methods for reverse engineering interface structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 969–978. ACM, 2011.
- [11] D. Fernandes, E. S. de Moura, A. S. da Silva, B. Ribeiro-Neto, and E. Braga. A site oriented method for segmenting web pages. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 215–224. ACM, 2011.
- [12] G. Hattori, K. Hoashi, K. Matsumoto, and F. Sugaya. Robust web page segmentation for mobile terminal using content-distances and page layout information. In *Proceedings of the 16th international conference on World Wide Web*, pages 361–370. ACM, 2007.
- [13] D. Karatzas, F. Shafait, S. Uchida, M. Iwamura, L. G. i Bigorda, S. R. Mestre, J. Mas, D. F. Mota, J. A. Almazan, and L. P. De Las Heras. Icdar 2013 robust reading com-



petition. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 1484–1493. IEEE, 2013.



[14] K. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk. Machine learning-based prototyping of graphical user interfaces for mobile apps. *arXiv preprint arXiv:1802.02312*, 2018.

[15] A. Pnueli, R. Bergman, S. Schein, and O. Barkol. Web page layout via visual segmentation. *HP Laboratories*, 2009.

[16] R. L. Potter. *Pixel Data Access: Interprocess Communication in the User Interface for End-user Programming and Graphical Macros*. PhD thesis, College Park, MD, USA, 1999. AAI9926789.

[17] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[18] A. Sanoja and S. Gançarski. Block-o-matic: A web page segmentation framework. In *Multimedia Computing and Systems (ICMCS), 2014 International Conference on*, pages 595–600. IEEE, 2014.

[19] E. Shah and E. Tilevich. Reverse-engineering user interfaces to facilitate porting to and across mobile devices and platforms. In *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPEs'11, NEAT'11, & VMIL'11*, pages 255–260. ACM, 2011.

- [20] R. Smith. An overview of the tesseract ocr engine. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, pages 629–633. IEEE, 2007.
- [21] A. Spengler and P. Gallinari. Document structure meets page layout: loopy random fields for web news content extraction. In *Proceedings of the 10th ACM symposium on Document engineering*, pages 151–160. ACM, 2010.
- [22] C. S. Win and M. M. S. Thwin. Web page segmentation and informative content extraction for effective information retrieval. *IJCCER*, 2(2):35–45, 2014.
- [23] T. Yeh, T.-H. Chang, and R. C. Miller. Sikuli: using gui screenshots for search and automation. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 183–192. ACM, 2009.
- [24] J. Zeleny, R. Burget, and J. Zendulka. Box clustering segmentation: A new method for vision-based web page preprocessing. *Information Processing & Management*, 53(3):735–750, 2017.

