APSIPA Transactions on Signal and Information Processing, 2022, 11, e20

Original Paper

Enhanced Automatic Areas of Interest (AOI) Bounding Boxes Estimation Algorithm for Dynamic Eye-Tracking Stimuli

Ezekiel Adriel D. Lagmay and Maria Mercedes T. Rodrigo*

Ateneo de Manila University, Quezon City, Philippines

ABSTRACT

In eye-tracking research, an area of interest (AOI) is defined as any object in the visual stimuli which is/are focused on by the viewer, defined with bounding boxes of any shape. If a study makes use of a small number of static visual stimuli, then researchers may define AOIs manually. However, if the stimuli is dynamic, then manual AOI definition is not efficient or scalable. This paper presents the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm which automatically estimates the AOI bounding boxes in dynamic stimuli using simple image segmentation techniques. This algorithm is an improvement on the Automatic AOI Bounding Boxes Estimation Algorithm. It uses a faster version of the SLIC algorithm which utilizes the AVX2 SIMD (Single Instruction, Multiple Data) parallelization paradigm, and replaces the second K-Means Image Segmentation procedure at the end of the pre-

*Corresponding author: Ezekiel Adriel D. Lagmay, ezekiel.lagmay@obf.ateneo.edu. The authors would like to thank the Ateneo Laboratory for the Learning Sciences (ALLS) Eye-Tracking Team for providing the necessary data sets for testing and analysis. The authors would also like to thank Proceso L. Fernandez, Ph.D. and Christian Wolf for their guidance and assistance in the evaluation of the the end results of the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm.

Received 25 June 2021; Revised 23 November 2021
ISSN 2048-7703; DOI 10.1561/116.00000026

vious version of the algorithm with Region Adjacency Graph (RAG) Thresholding. The evaluation of the overall results of the new algorithm shows that the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm is superior to its predecessor both in terms of accuracy (recall and precision) and efficiency (benchmarking).

Keywords: Eye-tracking, image segmentation, dynamic stimuli, areas of interest (AOI), bounding boxes

1 Introduction

1.1 Context of the Study

Eye-tracking is a method of data capture and analysis in which a person's eye movements (or gazes) are recorded as the person looks at a given visual stimuli [9]. These eye movements provide researchers with an objective indicator of where the person focuses his or her attention [9]. Because a stimulus may be composed of many parts with varying levels of importance, an important step in analyzing eye gaze is to divide the stimulus into areas of interest (AOIs). Defining these AOIs is often a manual process, one that is difficult to scale as the number of stimuli increases. To automate the process of defining AOIs, researchers have employed an area of computer vision known as object detection or "...detecting instances of semantically meaningful objects of certain classes, such as humans, buildings, or cars in digital images and videos" [14]. Object Detection has already been used in surveillance, transportation systems, driver assistance, smart rooms, and visual data summarizing, but the main challenge to achieving accurate object detection is the handling of complex data found in images and videos (such as pose, lighting, and clutter) [14].

The research presented in this paper is a subset of a larger research project on novice and expert programmer debugging skills [33, 36]. The goal of the project was to determine how novices and experts differed in the way they read and traced through buggy code in order to find errors. Participants were given 12 buggy programs, a text editor, and a compiler. As they attempted to debug these programs, their eye movements and all screen activity were recorded. Defining AOIs in a context such as this is challenging because the stimulus is dynamic and therefore AOIs are always changing in shape, form, or even location. Manual definition of AOIs, an approach used when visual stimuli is static, was not a feasible approach. It was therefore necessary to develop algorithms to automatically detect AOIs in order to make the process more efficient and scalable. This paper presents the algorithm that was developed for this purpose, and can be generalized to other cases in which dynamic AOIs comprised mainly of text.

Common approaches to addressing AOI definition in dynamic stimuli usually involve using sophisticated software/algorithms (such as deep learning packages like TensorFlow) which may require the use of specialized hardware (such as NVIDIA GPUs) for optimal results (examples include [4, 5, 25]). However, not all computers or operating systems support those kinds of hardware. For instance, Apple's macOS do not officially support NVIDIA CUDA or OpenCL, and even their computers ship with AMD Radeon GPUs instead of NVIDIA GPUs [7, 40]. Another alternative would be to use cloud services (such as Google Cloud, Amazon AWS, and Microsoft Azure; see [39] for an example), but the main caveat with this is that they either require a paid subscription or impose limitations on the usage of their features for free subscription tier users [3, 11, 20]. There are numerous image segmentation techniques that achieve similar outcomes, and if combined in the proper sequence, these image segmentation algorithms will enable the identification of AOIs in dynamic stimuli without the hardware and software compatibility caveats of deep learning approaches and as well without compromising efficiency and accuracy. In addition, they are readily available using free and open-source implementations in contrast to those available on popular cloud services. Lastly, this paper contributes a method of layering or chaining existing individual image segmentation algorithms, an approach we have not yet seen in the literature.

This paper discusses the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm, an improved version of the Automatic AOI Bounding Boxes Estimation Algorithm introduced by Lagmay [16] and Lagmay and Rodrigo [17]. This algorithm uses a faster version of the SLIC algorithm which utilizes the AVX2 SIMD (Single Instruction, Multiple Data) parallelization paradigm [15], and replaces the second K-Means Image Segmentation procedure at the end of the previous version of the algorithm with Region Adjacency Graph (RAG) Thresholding. The source code, data, resources, and results shown here are all available in the paper's official GitHub Repository: https://github.com/KielLagmay/EnhancedAutomaticAOIBoundingBoxesEstimationAlgorithm.

1.2 Research Objectives

The research objectives are as follows:

- To detect and keep track of AOIs automatically for each group of scenes which takes into consideration the presence of text.

- To develop an AOI bounding boxes estimation algorithm using simple image segmentation techniques instead of deep learning algorithms.

- To introduce new Image Segmentation techniques which would help improve the accuracy and efficiency of the previous Automatic AOI Bounding Boxes Estimation Algorithm.

1.3 Scope and Limitations

This study made use of video output data (in AVI format) from an eye-tracking experiment that attempted to determine how novice and expert programmers differed in the way they traced through code. We used data from participants from a single university in the Philippines (to be discussed further in Section 3).

2 Review of Related Literature

The literature review in this section focuses on publications concerning bounding boxes. Most of the literature that we have found in relation to dynamic stimuli eye-tracking/AOI are concerned with visualizations but not automatic determination of bounding boxes. Several methods exist to address the problem of automatic detection of areas/objects of interest. Some of the methods that these studies have utilized include Draw-and-Track (wherein the AOI(s) are first manually defined by the user and then is/are continuously tracked by the algorithm) [19], Active Video Methodology [14], and Image Segmentation [6, 8]. The following sections describe each of these techniques in detail.

2.1 Draw-and-Track AOI Detection

Liu *et al.* [19] tracked a user-defined AOI in a neurosurgical video by using a rectangle to define AOI borders, and the edge information of the surgery instrument is used to adjust the tracked position. The overall procedure makes use of Mean Shift Analysis for data cluster and object tracking, wherein the instrument is represented in a featured space by its estimated Probability Density Function [19]. Whenever the instrument changes orientation, the frame's edge information is used to update the tracking position [19]. Lastly, the Tangent Line Problem is used for smoothly "morphing" the AOI [19]. The final product of AOI detection with tracking is given in Fig. 6 in [19].

2.2 Active Video Methodology

The main limitation of the Draw-and-Track AOI Detection method is that AOI detection is done manually at the beginning of the algorithm (only the AOI tracking and update portions are automatic), and hence will not be efficient for more objects [19]. Huang and Li [14] attempted to solve this issue by looking into the interesting properties of Active Video which loosely resembles human eye movements in that it follows the Object(s) of Interest (OOIs) by means of panning, tracking (or dolly), or revolving (or tilting), which is very reminiscent of saccades [14]. These important properties thus help eliminate the need for manually defining the initial AOIs by providing actual cues for video segmentation and enable OOI tracking using the object-centered property manifested by Active Video [14]. The results are shown in Fig. 7 in [14], with the colored area in each grayscale video frame being the OOI.

This approach is not appropriate for desktop environments which are not active videos but rather passive videos: backgrounds are static and multiple objects appear, disappear, and occlude [14]. Also, in both [14] and [19], the sample videos used are of only one "scene" type.

2.3 Image Segmentation Algorithms

One essential component of automatically extracting and detecting AOIs from images and videos is image/video segmentation. This involves grouping the pixels of an image or video frame into different clusters according to characteristics such as color and texture. This essentially simplifies the image by showing the relevant objects shown in it and their respective areas and boundaries. One such algorithm, based on the K-Means Clustering Algorithm, is described by Dhanachandra *et al.* [8]. The results are shown in Figs. 3 and 5 in [8], where each region is indicated with a different grayscale color.

An alternative image segmentation algorithm is based on the DP Clustering Algorithm which "...Determine[s] the cluster numbers and cluster centers based on the decision graph, on the representation of the input image in color feature space" [6]. This is based on the notion that "cluster centers are often surrounded by points which has lower density and have a relatively large distance from these points with higher density" [6]. The procedure first involves converting the image into feature spaces using color channel feature, and then DP Algorithm is applied on the resulting representations, with Euclidean Distance used to measure distances between all input data, and Gaussian Kernel as the basic measurement of a point's density [6]. The results are shown in Figs. 6 and 7 in [6], where each color represents a specific region in the image.

2.4 Synthesis

Of the three techniques – Draw-and-Track, Active Video Methodology, and Image Segmentation – the last item has the most potential for the purposes of this study, since it is very efficient and generalizable to different scenes. Image Segmentation is automatic and considers important and essential objects in a frame (such as taskbars, titlebars, etc.), and most importantly, also considers lines of text.

3 Methodology

In this section, the video files and their frames that are used in evaluating the algorithm are first described. This is then followed by the discussion of the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm itself, starting with Histogram Equalization which can define AOI Bounding Boxes for lines of text, followed by K-Means and SLIC Image Segmentation techniques which enable the detection of Graphical User Interface (GUI) elements on a more general level. Image Sharpening, which allows the detection of AOIs for lines of text over a dark background, is discussed next, followed by Bilateral Filtering and RAG Thresholding which help smoothen the segmented outputs. Lastly, the methods for evaluating the algorithm, Overlapping Rectangles Evaluation Algorithm and Benchmarking, are discussed.

3.1 Dataset Description and Collection Methods

This study will be using the video output component of the Individual Dynamic Data collected from 9 participants from University A. The eye-trackers used in the study were made by Gazepoint, with a sample rate of 60 Hz and an accuracy of 0.5–1 degree [33]. The screen resolution of the monitors used in the experiment was 1024×768, and, by default, the problems were shown in full screen [33]. There were a total of 12 debugging problems shown to the participants, with the first three having a single bug while all others have three bugs [33]. For each problem, the participants were asked to read the given problem (in PDF form) and walk through its corresponding Java program code (which was shown in the Notepad++ application) [33]. The participants were free to make changes to the original program (adding code, deleting code, etc.) and they have access to a Java compiler via CMD. A typical experiment set-up used is shown in Figure 1 [33].

3.2 Enhanced Automatic AOI Bounding Boxes Estimation Algorithm

3.2.1 Histogram Equalization for Estimating AOI Bounding Boxes for Lines of Text

As mentioned earlier, Dhanachandra *et al.* [8] recommends the usage of Partial Contrast Stretching (or simply Contrast Stretching) to preprocess the images prior to segmentation to improve image quality and contrast. However, Tables 1 and 2 show that Histogram Equalization gives better outcomes than Contrast Stretching for this matter, despite the tendency of the former to yield unnatural looking images [29]. This is because Histogram Equalization approximates a rectangle area for each line of text, and hence, it could be used to define the AOIs particularly for text areas [16, 17].

Histogram Equalization is a “computer image processing technique used to improve contrast in images by spreading out the most frequent intensity values

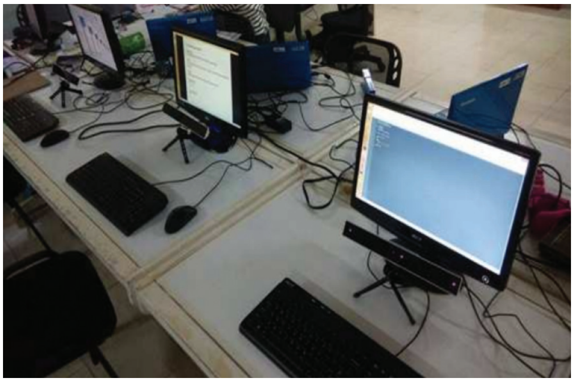
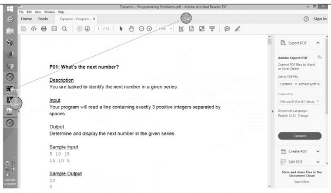
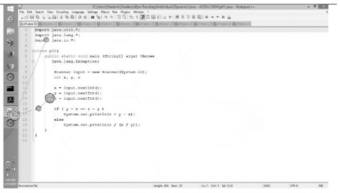
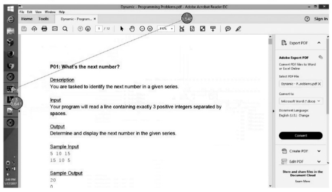
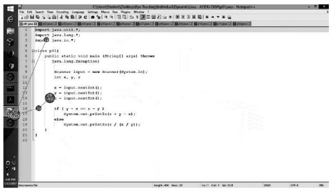
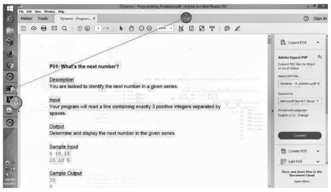
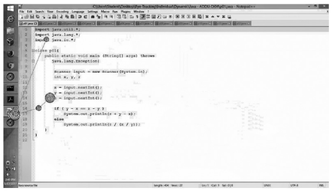
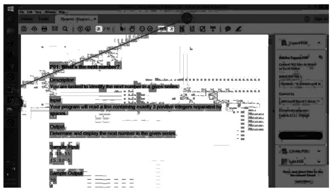
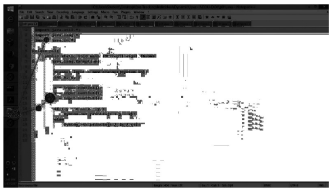


Figure 1: Eye-tracking experiment set-up used by ALLS [33].

Table 1: Histogram equalization results on selected frames of the stimuli, compared to similar techniques (part 1) [16, 17, 29].

	PDF File	Source Code
Original		
Contrast Stretching		
Adaptive Equalization		
Histogram Equalization		

Dynamic - Programming Problems.pdf - Adobe Acrobat Reader DC

P01: What's the next number?

Description
You are tasked to identify the next number in a given series.

Input
Your program will read a line containing exactly 3 positive integers separated by spaces.

Output
Determine and display the next number in the given series.

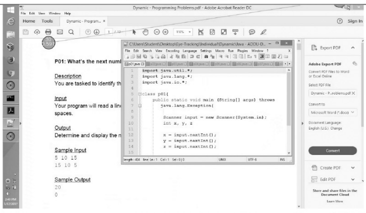
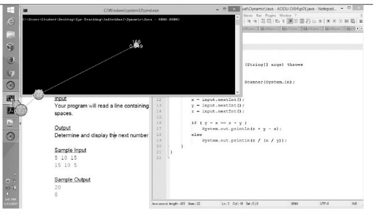
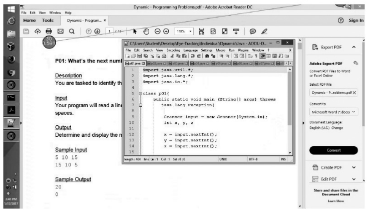
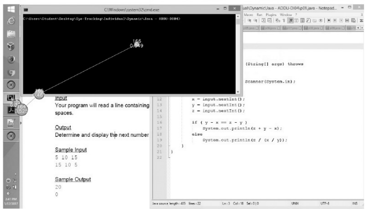
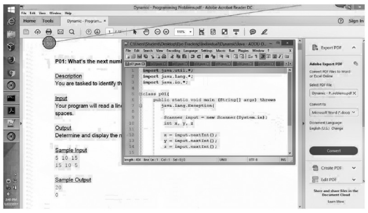
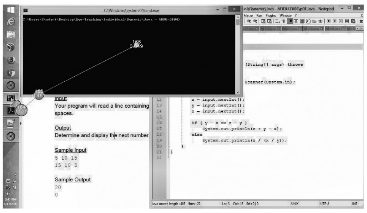
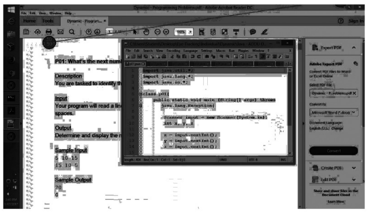
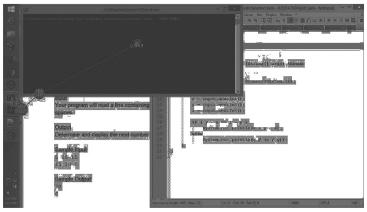
Sample Input
```
5 10 15
15 10 5
```

Sample Output
```
20
0
```

C:\Users\Student\Desktop\Eye-Tracking\Individual\Dynamic\Java - ADDU-DI04\p01.java - Notepad++

```java
import java.util.*;
import java.lang.*;
import java.io.*;

class p01{
    public static void main (String[] args) throws
        java.lang.Exception{

        Scanner input = new Scanner(System.in);
        int x, y, z

        x = input.nextInt();
        y = input.nextInt();
        z = input.nextInt();

        if ( y - x == z - y )
            System.out.println(z + y - x);
        else
            System.out.println(z / (x / y));
    }
}
```

Table 2: Histogram equalization results on selected frames of the stimuli, compared to similar techniques (part 2) [16, 17, 29].

	Source Code + PDF	CMD + Source Code + PDF
Original		
Contrast Stretching		
Adaptive Equalization		
Histogram Equalization		

(or stretching out the intensity range of the image)" [32]. This is based from image transformation functions (intensity mappings) involving Probability Density Functions (PDFs) and Integrals, specifically the following equation [10, 35]:

$$s = T(r) = (L-1)\int_0^r p_r(w)dw \tag{1}$$

Where w is a dummy variable of integration [10, 35]. The right side of the equation is the Cumulative Distribution Function (CDF) of random variable r

[10, 35]. This is explained further by Gonzalez and Woods [10] and UCI [35]. In the case of discrete values, however, Probabilities (Histogram Values) and Summations are considered instead [10, 35]. The probability of occurrence of intensity level r_k in a digital image is approximated by [10, 35]:

$$p_r(r_k) = \frac{n_k}{MN}; \quad k = 0, 1, \ldots, L-1 \tag{2}$$

Where MN is the total number of pixels in the image, n_k is the number of pixels having intensity r_k, and L is the number of possible intensity levels in the image (usually 256) [10, 35]. A plot of $p_r(r_k)$ vs. r_k is known as a histogram [10, 35]. The discrete counterpart of the transformation of Equation (1) is given by [10, 35]:

$$\begin{aligned} s_k &= T(r_k) \\ &= (L-1)\sum_{j=0}^{k} p_r(r_j) \\ &= \frac{(L-1)}{MN}\sum_{j=0}^{k} n_j; \quad k = 0, 1, \ldots, L-1 \end{aligned} \tag{3}$$

In other words, each pixel in the input image with intensity r_k is mapped to a corresponding pixel with level s_k in the output image [10, 35]. Here, the transformation (mapping) function $T(r_k)$ is known as Histogram Equalization or Histogram Linearization Transformation.

In Python, Histogram Equalization is available through Scikit-Image (*skimage*)'s *exposure* module via the function *equalize_hist*() [28].

3.2.2 K-Means and SLIC Image Segmentation for Estimating AOI Bounding Boxes for Non-Text (GUI) Elements

However, using Histogram Equalization for AOI approximation is still insufficient for two reasons. Firstly, it treats adjacent non-text elements (i.e., the GUI) as a single AOI. It does not recognize them as separate components (ex. Taskbars, Titlebars, Menu Items, etc.). In eye-tracking research, these items need to be recognized as separate components. To remedy this, Image Segmentation Algorithms (as discussed in [6, 8, 12]) can be used along with Histogram Equalization to help approximate AOI boundaries most especially for non-text elements. Specifically, K-Means Image Clustering and Simple Linear Iterative Clustering (SLIC) are used to further simplify the image and help determine AOI areas most especially for non-text elements [1, 8]. The K-Means Image Segmentation procedure is explained in detail in [8].

As with its predecessor, the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm uses the SLIC Image Segmentation Algorithm (summarized in Fig. 2 in [15] and sample results in Fig. 1 in [1]; original paper in [1]) [1, 16], but with optimizations to the following [15]:

1. CIELAB conversion and quantization – Convert RGB images into quantized CIELAB images represented with unsigned 8-bit integers. This procedure is parallelized on a per-pixel basis [15].

2. Cluster assignment – Use Manhattan Distance instead of Euclidean Distance as distance metric since a distance value is always represented with 16-bit unsigned integers instead of 32-bit floating point numbers, and because Manhattan Distance is more computationally efficient as only addition, subtraction, and conditional move are involved. To perform the Cluster Assignment Procedure using integer-only arithmetic (illustrated in Fig. 3 in [15]), a minimum distance value d_{ij} is maintained to its cluster during an iteration, and for each cluster k, d_{ij} is updated into distance to the centroid if it has less value. Lastly, the cluster labels are represented with unsigned 16-bit integers as well. This procedure is parallelized using a tiling scheme such that assignment of cluster k and cluster l can run in parallel if two regions of interest do not overlap. (illustrated in Fig. 5 in [15]) [15].

3. Centroid update – Each centroid moves to rounded integers of its center of mass in order to maintain the property that the elements of centroids and pixels are represented with integers. This step is parallelized by accumulating 5-dimensional pixel vectors for each cluster in each thread and gathering from threads [15].

4. Connectivity enforcement – Parallelized by utilizing the parallel CCL procedure by Gupta *et al.* [13] with some modifications in that adjacent pixels are treated as neighbors only if they have the same cluster number. Also, 4-connectivity (wherein only the north, east, south, and west pixels of each pixel are considered adjacent) was used instead of 8-connectivity for performance reasons [15].

This process can be optimized further using SIMD Vectorization and Row Subsampling (such as AVX2 for x86-64 systems or NEON for ARM systems), which reduces memory access during clustering [15]. Row Subsampling draws samples from a set of image rows (analogous to a long contiguous line of memory) and then only the rows from the sample are used for cluster assignment and centroid update, as opposed to Pixel Subsampling which treats each pixel in an image as an independent unit and then draws a proportion p of pixels from the image (see also Fig. 4 in [15]). Because of this, processing a row subsample results in less cache miss and that it could be easily implemented and optimized using SIMD unlike pixel subsamples [15]. Specifically, for each iteration i, rows whose index y satisfies $y \mod \mathit{Stride} \equiv i$ are sampled [15]. Centroid updates are then performed in the same way as the standard k-means clustering, but without weighted update or gradient step [15].

The overall FastSLIC algorithm is given in Algorithm 1 [15].

Algorithm 1: Overall FastSLIC Algorithm [15]

```
Input: $W \times H$ quantized CIELAB Image $I_{ij} = \{l_{ij}, a_{ij}, b_{ij}\}$, initial $K$
       centroids $C_k = \{x_k, y_k, l_k, a_k, b_k\}$, grid size $S$, compactness $m$,
       color quantization scale $n$, max iteration $N$, Subsampling
       stride $Stride$
Output: superpixel assignment $A_{ij}$
1 repeat
      // The assignment procedure is further described in
         Algorithm 2 by Kim [15]
2     Assign clusters and store the assignment into $A_{ij}$ with $Offset$ and
       $Stride$.
3     Update the centroids from assignment $A_{ij}$ in parallel.
4     $Offset \leftarrow (Offset + 1) \mod Stride$;
5 until iterated $N$ times;
6 Assign clusters and store into $A_{ij}$ with $Stride = 1$.
7 Enforce connectivity of $A_{ij}$.
```

3.2.3 Image Sharpening as a Helper to Histogram Equalization for Estimating AOI Bounding Boxes on Text Over Dark Backgrounds

Secondly, while Histogram Equalization works for text on a light background, it is not able to estimate AOIs for a line of text on a dark background (this would be detrimental most especially if a line of text happens to be on a command-line terminal). This can be solved by performing Image Sharpening on the video frame before performing Histogram Equalization on it. Image Sharpening and Blurring are both Image Filtering techniques which change the intensity of the noise in the picture by normalizing or averaging a block of pixels in an image to have low values difference [2]. In Image Blurring, each pixel value changes according to its neighboring pixels by taking the total average of all pixels [2]. The size of the pixels which the manipulation function is run on is called the kernel (usually 3×3) [2].

Image Sharpening uses a similar process, but the goal is to pop out details in the image by making each pixel value different from its neighbors [2]. The PIL Python Library has the Image, ImageFilter, and ImageFile libraries that can sharpen an image via the $filter(ImageFilter.SHARPEN)$ method that is called on the image object [26]. More specifically, $ImageFilter.SHARPEN$ implements a spatial filter using convolution to sharpen an image [26]. The convolution matrix used is $[[-2, -2, -2], [-2, 32, -2], [-2, -2, -2]]$ (as opposed to the usual matrix configuration of $[[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]]$) [2, 26].

3.2.4 Other Image Processing Techniques Utilized and Overall Algorithm

Lastly, in order to clearly define boundaries, procedures which help "smoothen" the image are carried out. In between the K-Means Image Segmentation and SLIC Image Segmentation procedures, an Image Smoothing procedure known as Bilateral Filtering is performed [21]. Bilateral Filtering uses a Gaussian filter "...in the space domain, but it also uses one more (multiplicative) Gaussian filter component which is a function of pixel intensity differences" [21]. The Gaussian function of space ensures that only pixels that are "spatial neighbors" are considered for filtering, while the Gaussian component applied in the intensity domain ensures that only those pixels with intensities similar to that of the central pixel are included to compute the blurred intensity value [21]. Thus, unlike other techniques, Bilateral Filtering is very effective at removing noise while at the same time preserving edges since "for pixels lying near edges, neighboring pixels placed on the other side of the edge, and therefore exhibiting large intensity variations when compared to the central pixel, will not be included for blurring" [21].

In addition, after the SLIC Image Segmentation step but before finally saving to a .png file, the video frame undergoes a process called RAG Thresholding which constructs an RAG from the image, defines edges as the difference in mean color, and then merges regions which have similar mean color [30].

The overall implementation of the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm in Python can be found in the function *estimateAOIBoundingBoxes* of the EstimateAOIBoundingBoxes_Standalone.py file in the official GitHub Repository, and the procedure is illustrated in Figure 2. Note that unlike in the original Automatic AOI Bounding Boxes Estimation Algorithm in [16], the temp image(s) are in the JPEG image format instead of the PNG format. Hence, the original PNG file is first converted into a JPEG file at the start of the algorithm, but the final output is still saved as a PNG file. In addition, the user is free to change the parameters for each of the steps in the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm by passing a dictionary of parameters to the *paramList* parameter.

3.3 Results Evaluation

The Enhanced Automatic AOI Bounding Boxes Estimation Algorithm was tested on the grayscale representation of 10 selected frames obtained from the video output data of three participants: Frame 267 from Participant 2, Frames 691 and 703 from Participant 3, and Frames 18, 125, 139, 170, 176, 225, and 803 from Participant 4. The results are shown in Tables 3 and 4.

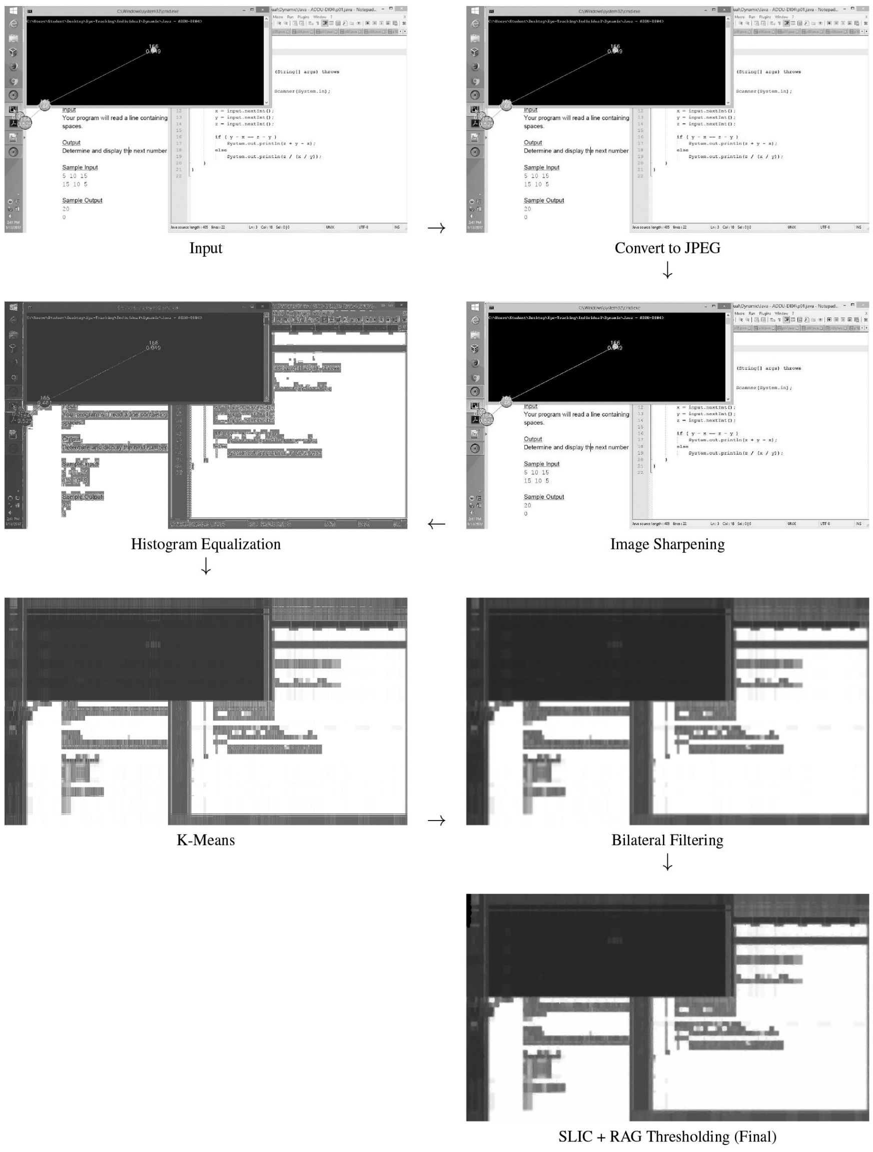


Figure 2: Illustration of the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm.

3.3.1 Overlapping Rectangles Evaluation Algorithm

To check the accuracy of the generated AOI bounding boxes to the ground truth, the Overlapping Rectangles Evaluation Algorithm (an improved version of Intersection Over Union (IOU), which is explained in detail in [27]) will be used, since simple IOU would not be trivial in the case of multiple images or a single image with multiple text rectangles [38]. More specifically, "the way the overlap information is accumulated during the calculation of the evaluation measures leaves room for ambiguity" [38]. Hence, a more accurate way of evaluating AOI rectangle bounding boxes must meet the following criteria [38]:

1. The evaluation measure should intuitively tell how many text rectangles have been detected correctly, and how many false alarms have been created (quantitative evaluation).

2. It should give an easy interpretation of the detection quality (qualitative evaluation).

3. It should support one-to-one matches, one-to-many matches, and many-to-one matches.

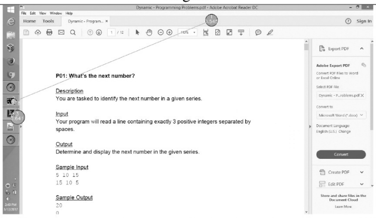
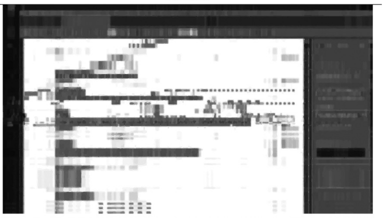
4. The measure must scale up to multiple images without losing its power and ease of interpretation.

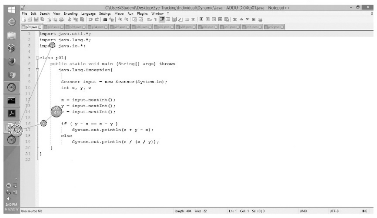
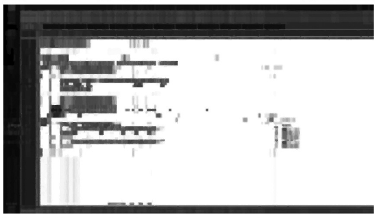
The first two goals in the criteria are contradicting yet related in that "the number of rectangles we consider as detected depends on the quality requirements which we impose for a single rectangle in order to be considered as dectected" [38]. Hence, two-dimensional plots are used to bridge the two goals: the y-axis is used to plot two measures on object counts (Recall R_{OB} and Precision P_{OB}) [38]. The formulas for R_{OB} and P_{OB} are given in Equation (4) [38]:

$$\begin{aligned} R_{OB} &= \frac{\text{No. of correctly detected rectangles}}{\text{No. of rectangles in the database}} \\ P_{OB} &= \frac{\text{No. of correctly detected rectangles}}{\text{Total no. of detected rectangles}} \end{aligned} \tag{4}$$

To compute the measures given in Equation (4), it is required to determine for each ground truth rectangle G_i whether it has been detected or not, and to determine for each detection result rectangle D_i whether its detection is correct or not [38]. In order to meet the third item in the criteria, the method in [18] for calculating the overlap matrices σ and τ was utilized [38]. The matrices are then analyzed in order to determine the correspondence between the two rectangle lists: a non-zero value in an element with indices (i, j) indicates

Table 3: Enhanced Automatic AOI Bounding Boxes Estimation Algorithm results for some frames (part 1).

Frame	Original	Result
Frame 18	 Dynamic - Programming Problems.pdf - Adobe Acrobat Reader DC **P01: What's the next number?** Description You are tasked to identify the next number in a given series. Input Your program will read a line containing exactly 3 positive integers separated by spaces. Output Determine and display the next number in the given series. Sample Input 5 10 15 15 10 5 Sample Output 20 0	

Frame 125	 C:\Users\Student\Desktop\Eye-Tracking\Individual\Dynamic\Java - ADDU-DI04\p01.java - Notepad++
	`import java.util.*;`
	`import java.lang.*;`
	`import java.io.*;`
	`class p01{`
	`public static void main (String[] args) throws java.lang.Exception{`
	`Scanner input = new Scanner(System.in);`
	`int x, y, z`
	`x = input.nextInt();`
	`y = input.nextInt();`
	`z = input.nextInt();`
	`if ( y - x == z - y )`
	`System.out.println(z + y - x);`
	`else`
	`System.out.println(z / (x / y));`
	`}`
	`}`
		

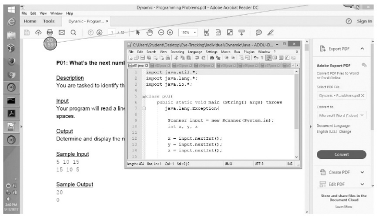
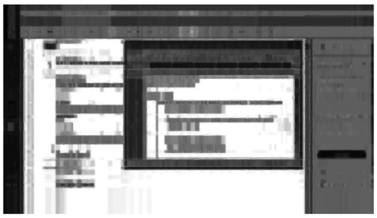
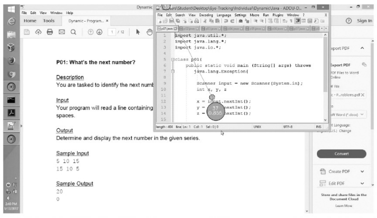
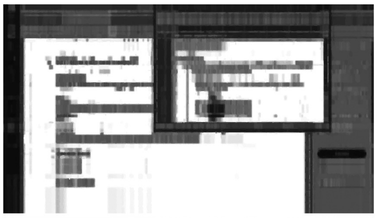
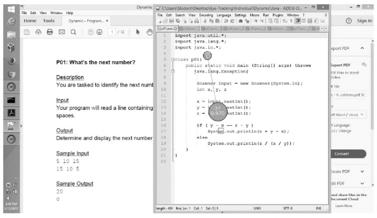
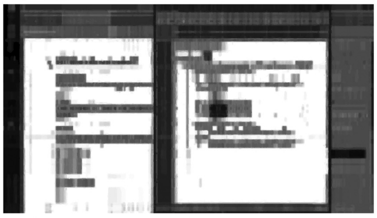
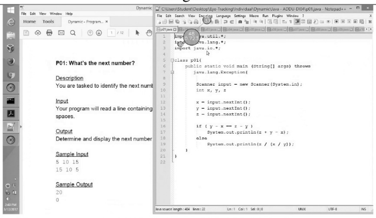
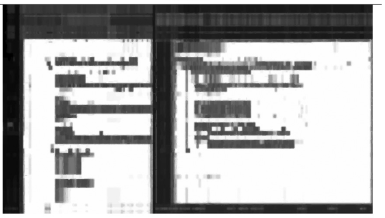
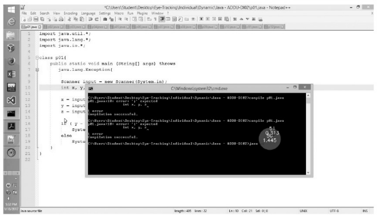
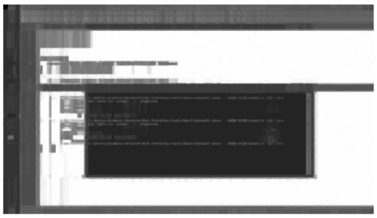
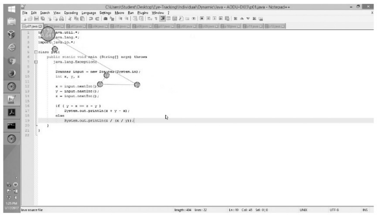
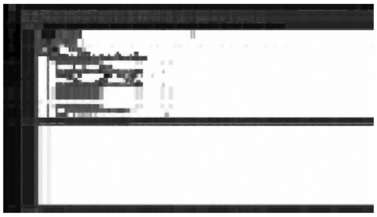
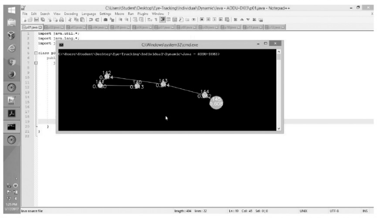
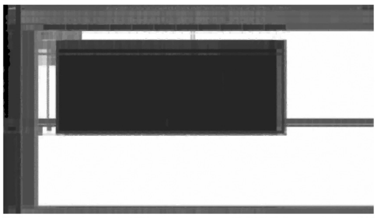
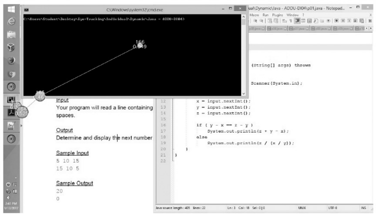
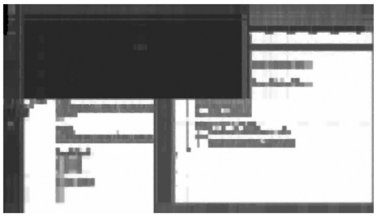
Frame 139		
Frame 170		
Frame 176		

Table 4: Enhanced Automatic AOI Bounding Boxes Estimation Algorithm results for some frames (part 2).

Frame	Original	Result
Frame 225		
Frame 267		
Frame 691		
Frame 703		
Frame 803		

P01: What's the next number?

Description
You are tasked to identify the next numb

Input
Your program will read a line containing
spaces.

Output
Determine and display the next number

Sample Input
5 10 15
15 10 5

Sample Output
20
0

```
import java.util.*;
import java.lang.*;
import java.io.*;

class p01{
    public static void main (String[] args) throws
      java.lang.Exception{

       Scanner input = new Scanner(System.in);
       int x, y, z

       x = input.nextInt();
       y = input.nextInt();
       z = input.nextInt();

       if ( y - x == z - y )
          System.out.println(z + y - x);
       else
          System.out.println(z / (x / y));
    }
}
```

that ground truth rectangle G_i overlaps with result rectangle D_j, but the two rectangles are matched only if area recall and area precision are higher than the respective constraint [38]:

$$\begin{aligned} \sigma_{ij} &> t_r \\ \tau_{ij} &> t_p \end{aligned} \tag{5}$$

Where $t_r \in [0,1]$ and $t_p \in [0,1]$ are the constraints on area recall and area precision, respectively [38]. The matching criteria are then defined for each of the following cases (illustrated in Fig. 2 in [38]):

- **One-to-one matches**: One ground truth rectangle G_i matches with a result rectangle D_j if row i and column j of both matrices contain only one element satisfying Equation (5) [38].
- **One-to-many matches (splits)**: One ground truth rectangle G_i matches against a set S_o of result rectangles $D_j, j \in S_o$ if a sufficiently large proportion of the ground truth rectangle has been detected (i.e., $\sum_{j\in S_o} \sigma_{ij} \geq t_r$), and each contributing result rectangle overlaps enough with the ground truth rectangle to be considered a part of it (i.e., $\forall j \in S_o : \tau_{ij} \geq t_p$) [38].
- **Many-to-one matches (merges)**: One result rectangle D_j matches against a set S_m of ground truth rectangles if a sufficiently large portion of each ground truth rectangle is detected (i.e., $\forall i \in S_m : \sigma_{ij} \geq t_r$), and each ground truth rectangle has been detected with enough area precision (i.e., $\sum_{i\in S_m} \tau_{ij} \geq t_p$) [38].
- **Many-to-many matches (splits and merges)**: This is translated into several splits or a set of splits and one-to-one matches, since it is currently not supported (each ground truth rectangle in the matching set is either part of a split if it is matched against several detected rectangles, or it is part of a one-to-one match if it is matched against a single detected rectangle) [38].

R_{OB} and P_{OB} can now be defined as follows given the matching strategy [38]:

$$\begin{aligned} R_{OB}(G, D, t_r, t_p) &= \frac{\sum_i Match_G(G_i, D, t_r, t_p)}{|G|} \\ P_{OB}(G, D, t_r, t_p) &= \frac{\sum_j Match_D(D_j, G, t_r, t_p)}{|D|} \end{aligned} \tag{6}$$

Where $Match_G$ and $Match_D$ are piecewise functions which take into account the different types of matches and evaluate the quality of the match [38]:

$$\begin{aligned}
Match_G\left(G_i, D, t_r, t_p\right) &= \begin{cases} 1 & \text{if } G_i \text{ matches against a single detected rectangle} \\ 0 & \text{if } G_i \text{ does not match against any detected rectangle} \\ f_{sc}\left(k\right) & \text{if } G_i \text{ matches against several } (\rightarrow k) \text{ detected rectangles} \end{cases} \\
Match_D\left(D_j, G, t_r, t_p\right) &= \begin{cases} 1 & \text{if } D_j \text{ matches against a single detected rectangle} \\ 0 & \text{if } D_j \text{ does not match against any detected rectangle} \\ f_{sc}\left(k\right) & \text{if } D_j \text{ matches against several } (\rightarrow k) \text{ detected rectangles} \end{cases}
\end{aligned} \tag{7}$$

Where $f_{sc}\left(k\right)$ is a parameter function of the evaluation scheme which controls the penalty for splits or merges, set to a constant 0.8 by default [38].

In the case of N images, several lists of ground truth rectangles $G^k \in \overline{\mathbf{G}}, k = 1..N$ are compared with several lists of result rectangles $D^k \in \overline{\mathbf{D}}, k = 1..N$ [38]. Instead of accumulating results on multiple images by summing the recall or precision values, object recall and object precision are defined as follows [38]:

$$\begin{aligned}
R_{OB}\left(\overline{\mathbf{G}}, \overline{\mathbf{D}}, t_r, t_p\right) &= \frac{\sum_k \sum_i Match_G\left(G_i^k, D^k, t_r, t_p\right)}{\sum_k \left|G^k\right|} \\
P_{OB}\left(\overline{\mathbf{G}}, \overline{\mathbf{D}}, t_r, t_p\right) &= \frac{\sum_k \sum_j Match_D\left(D_j^k, G^k, t_r, t_p\right)}{\sum_k \left|D^k\right|}
\end{aligned} \tag{8}$$

The object-related Recall and Precision measures in Equation (8) depends on two constraints t_r and t_p which impose constraints on the detection quality [38]. The performance diagrams (shown in Fig. 5 in [38]) are created by fixing one constraint to a set value, allowing the second one (the x-axis) to vary, and plotting object recall and precision on the y-axis of two graphs [38].

The graphs provide a straightforward interpretation: if object recall never drops to zero when area recall approaches 1 then most of the text rectangles detected rarely cuts parts of the ground truth rectangle (area coverage of 100%), while if object recall drops to zero when area precision approaches 1 then all result rectangles exceed the ground truth boundaries [38]. Also, the particular amount of area which is detected additionally can be seen by the point/range where the object recall dramatically drops when area precision increases [38].

By default, t_r is fixed to 0.8 while t_p is set to a lower value of 0.4 due to the fact that a detection result which cuts parts of the text rectangle is more disturbing than a detection which results in a too large rectangle [38]. In addition, the area of a rectangle grows with the square of its side lengths; this is shown in Fig. 3 in [38], wherein the detected rectangle has 50% area precision since it is twice as large as the ground truth rectangle, although the difference in the corner coordinates is quite small [38].

While Performance Diagrams are a very intuitive way to illustrate the performance of an object detection algorithm, it is often useful to determine a single performance value for an algorithm, either for direct comparison of the performances of different algorithms, or to optimize the parameters of the detection algorithm, or to control the algorithm [38]. This is a difficult task to achieve, since a "single value is hardly able to characterize the complex behavior of a detection algorithm" [38]. In addition, a "good indicator should cover the performance of the evaluated algorithm across a whole range of quality constraints" [38]. Hence, the proportion of the graph area which is beneath the performance graphs is used as a "reliable and objective measure," which is equivalent to the mean value of object measures over all possible constraint values [38]. Firstly, the area proportion is separately calculated for object recall and object precision [38]:

$$\begin{aligned} R_{OV} &= \frac{1}{2T}\sum_{i=1}^{T} R_{OB}\left(\overline{\mathbf{G}}, \overline{\mathbf{D}}, \frac{i}{T}, t_p\right) + \frac{1}{2T}\sum_{i=1}^{T} R_{OB}\left(\overline{\mathbf{G}}, \overline{\mathbf{D}}, t_r, \frac{i}{T}\right) \\ P_{OV} &= \frac{1}{2T}\sum_{i=1}^{T} P_{OB}\left(\overline{\mathbf{G}}, \overline{\mathbf{D}}, \frac{i}{T}, t_p\right) + \frac{1}{2T}\sum_{i=1}^{T} P_{OB}\left(\overline{\mathbf{G}}, \overline{\mathbf{D}}, t_r, \frac{i}{T}\right) \end{aligned} \tag{9}$$

The final performance value is the harmonic mean of the two measures [38]:

$$Perf_{OV} = 2\frac{P_{OV} \cdot R_{OV}}{P_{OV} + R_{OV}} \tag{10}$$

Note that the parameter T is a "granularity parameter which controls the trade-off between the computational complexity of the evaluation algorithm and the precision of the integration approximation" [38]. Since it is not likely that the object-related measures change sharply after changing the quality constraints in very small steps, T is set to 20 by default [38].

Since the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm (and its predecessor) is not yet able to give the actual bounding boxes and their coordinates, the final outputs are each inputted into the *adaptiveThreshold* OpenCV method with the following parameters to obtain their edge information [23, 24, 34]:

- $maxValue = 255$
- $adaptiveMethod = cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C$
- $thresholdType = cv2.THRESH\_BINARY$
- $blockSize = 15$
- $C = 0.01$

The edge information obtained from the *adaptiveThreshold* method for each result are then used as the basis for the manual drawing of the actual AOI Bounding Boxes and the collection of their coordinates. The bounding boxes used for the Overlapping Rectangles Evaluation procedure are shown here for the Ground Truth (Table 5), the Result of the New Algorithm (Tables 6 and 7), and the Result of the Old Algorithm (Tables 8 and 9).

The actual values of the AOI Bounding Box coordinates can be found in the Overlapping Rectangles Evaluation.xlsx Excel File in the Overlapping Rectangles Evaluation folder of the official repository of this paper. These values are then converted into XML format and then passed into the DetEval Program (the actual implementation of [38]) [37]. $evalfixed$ was used to evaluate the algorithm (using fixed constraint parameters), while *evalplots* was used to generate files containing data for plotting via the Matplotlib Python library (after the files were manually compiled into a single CSV file for each of the data groups) [37].

3.3.2 Benchmark Evaluation

Lastly, a benchmark comparison was also done to measure the runtime of the algorithms by using *time.time*() before (*start*) and after (*end*) running the function corresponding to the algorithm, then getting the difference between *end* and *start*, with the unit in seconds [22]. The benchmarking tests were done on a MacBook Pro (Mid-2014 13" Retina Display model) with a 2.6 GHz Dual-Core Intel Core i5 processor, an 8 GB 1600 MHz DDR3 memory, and an Intel Iris 1536 MB graphics processor, running macOS 10.15 Catalina (as of the time the paper was written).

4 Results

This section begins with the discussion of the benchmarking results, which is then followed by some experimental results involving changing the parameters for the SLIC Components and the RAG Cut Threshold, and as well changing the format of the temp image files to PNG instead of JPEG. The Results section concludes with a brief discussion of the Overlapping Rectangles Evaluation Algorithm results, which give the accuracy of the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm.

4.1 Benchmarking Results

The benchmarking results for the improved algorithm are shown in Table 10 [22]. It can obviously be seen that the runtime of the improved algorithm is significantly faster than the previous version (see Table 11). Please note, however, that the values shown here are rounded off to two decimal places; for the actual values, please see the AutomaticAOIBoundingBoxesEstimationBenchmarks_New.txt (benchmark results for the new algorithm) and AutomaticAOIBoundingBoxesEstimationBenchmarks_Old.txt (benchmark results for the old algorithm) files in the Benchmark Results folder of the official GitHub repository.

4.2 SLIC Components and RAG Cut Threshold Parameter Testing

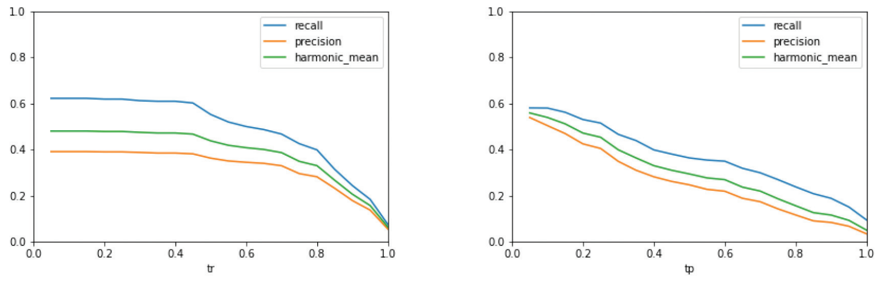
The Enhanced Automatic AOI Bounding Boxes Estimation Algorithm was tested for different values of the SLIC Components Parameter and the RAG Cut Threshold Parameter. Specifically, 5000, 7000, and 9000 were used as test values for the SLIC Components Parameter, while 3 and 5 were used as test values for the RAG Cut Threshold Parameter. Based from the results, a SLIC Components Parameter value of 9000 and a RAG Cut Threshold Parameter value of 3 qualitatively give the most accurate results in that boundaries between adjacent estimated AOI bounding boxes for non-text (GUI) elements are clear-cut and well-defined. Note that throughout the algorithm evaluation procedure, SLIC Compactness is set to 1 since that value balances the color and space factors during the SLIC Image Segmentation Algorithm [31]. The overall results of the parameter testing are given in Tables 12–17.

4.3 Utilization of the JPEG Format for Temp Image File(s)

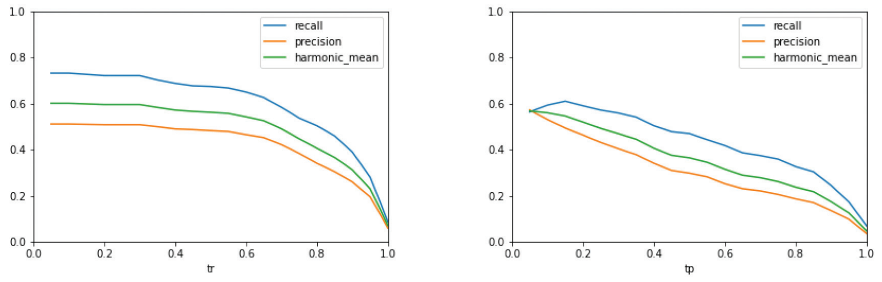
As mentioned earlier, the temp image file(s) generated during the execution of the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm are saved in JPEG format instead of PNG, unlike its predecessor [16]. However, it was decided to do an experiment wherein the temp image file(s) generated by the new algorithm are saved in PNG format similar to that of the old algorithm, and qualitatively, it gave less accurate results in that it causes oversegmentation (i.e., fails to estimate AOI bounding boxes in a more general, less-specific manner). Tables 18 and 19 show this point.

4.4 Overlapping Rectangles Evaluation Algorithm Results

Table 20 and Figure 3 show the results of the Overlapping Rectangles Evaluation Algorithm on the Old (Automatic AOI Bounding Boxes Estimation Algorithm) and New (Enhanced Automatic AOI Bounding Boxes Estimation Algorithm) Algorithms [37, 38]. It can be clearly seen that the new version has better Recall and Precision scores than the old version in [16], hence implying that the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm is more accurate than its predecessor. Please note that the values shown here are rounded off to at most two decimal places; for the actual values, please refer to the Predicted-New-Results.xml and Plot-Summary-New.txt files (for the new algorithm), and the Predicted-Old-Results.xml and Plot-Summary-Old.txt



(a) Old



(b) New

Figure 3: The resulting performance graphs of the old and new Automatic AOI Bounding Boxes Estimation Algorithms [37, 38].

files (for the old algorithm), in the Overlapping Rectangles Evaluation folder of the official GitHub repository.

5 Conclusion

The Enhanced Automatic AOI Bounding Boxes Estimation Algorithm is able to address the following research objectives:

- **To detect and keep track of AOIs automatically for each group of scenes which takes into consideration the presence of text.** Like its predecessor, the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm uses Image Sharpening followed by Histogram Equalization prior to the image segmentation proper to be able to estimate AOI bounding boxes for lines of text, regardless of the background.
- **To develop an AOI bounding boxes estimation algorithm using simple image segmentation techniques instead of deep learning algorithms.** The Enhanced Automatic AOI Bounding Boxes Estimation Algorithm utilizes free and open-sourced image segmentation algorithms in Python stacked together which are able to detect AOIs without the need for either specialized GPUs or cloud computing solutions.
- **To introduce new Image Segmentation techniques which would help improve the accuracy and efficiency of the previous Automatic AOI Bounding Boxes Estimation Algorithm.** The new algorithm uses a faster version of SLIC which utilizes AVX2 SIMD parallelization and replaces the second K-Means Image Segmentation procedure at the end of the previous algorithm with RAG Thresholding. Hence, it performs faster than the previous version, and its recall and precision values are also higher than that of the old algorithm. The evaluation method used has also improved significantly from the previous version, as it is now able to capture one-to-many and many-to-one scenarios [38].

There are, however, still areas for improvement, most especially on adding the capability for giving out the actual AOI bounding boxes (and their coordinates) automatically. But the use of Adaptive Thresholding to get the edge information looks very promising in meeting that goal. In addition, it is also planned to test out the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm on more video frames in order to get a much clearer picture of its accuracy.

The methodologies laid out in this study were applied to a study that compared how novice and expert programmers differ in the way they debug programs. However, these methods may be applied to almost any eye-tracking study that makes use of dynamic stimuli. User experience studies that make use of eye-tracking to monitor how users interact with dynamic interfaces with textual elements can make use of these methods to automatically define bounding boxes of areas of interest.

Table 5: AOI Bounding Boxes for the ground truth [23, 24, 34].

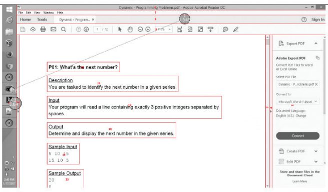
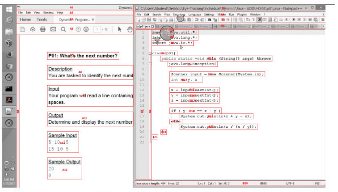
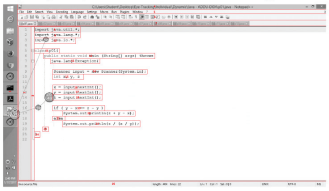
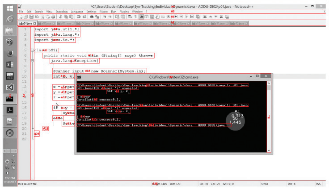
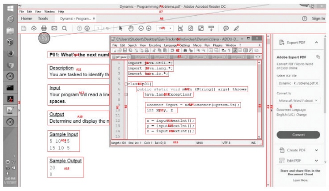
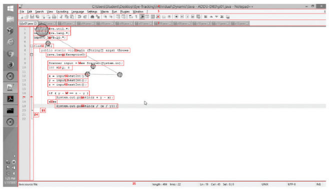
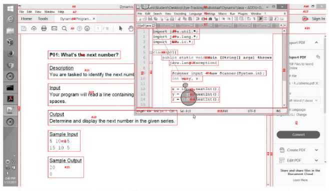
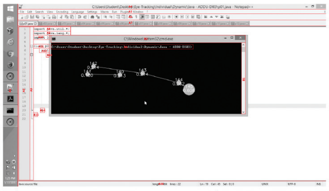
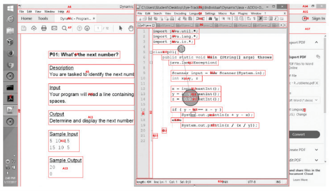
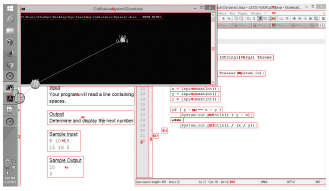
Frame	Ground Truth	Frame	Ground Truth
Frame 18		Frame 225	
Frame 125		Frame 267	
Frame 139		Frame 691	
Frame 170		Frame 703	
Frame 176		Frame 803	

Table 6: AOI Bounding Boxes for the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm (new version) (part 1) [23, 24, 34].

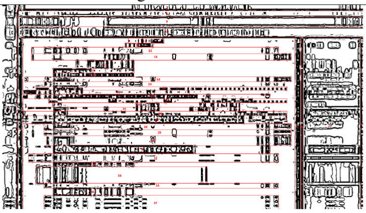
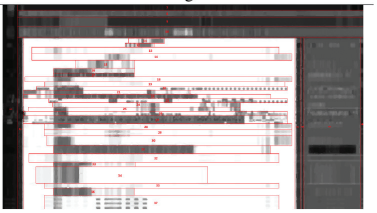
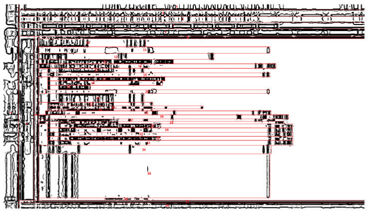
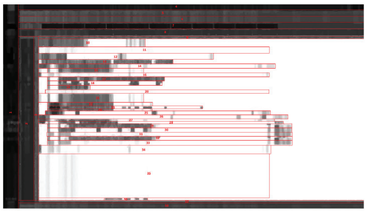
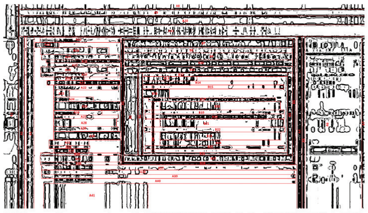
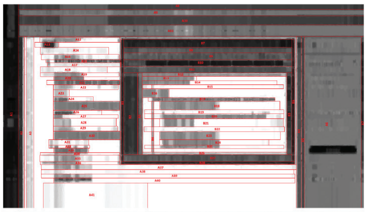
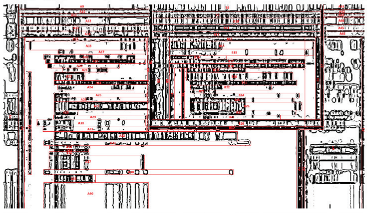
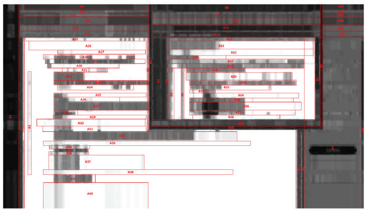
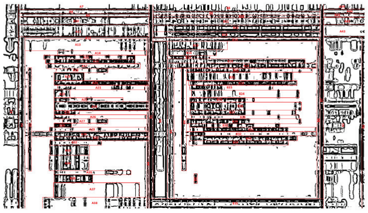
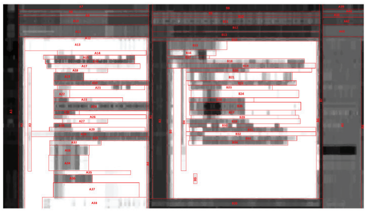
Frame	New Version	
	Edge Information	Bounding Boxes
Frame 18		
Frame 125		
Frame 139		
Frame 170		
Frame 176		

Table 7: AOI Bounding Boxes for the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm (new version) (part 2) [23, 24, 34].

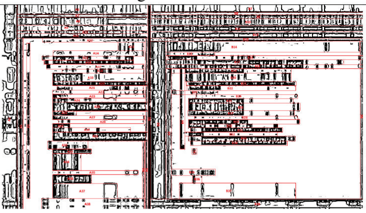
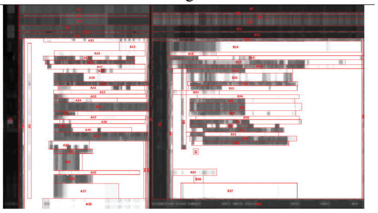
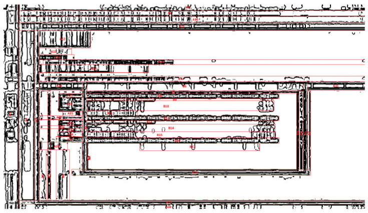
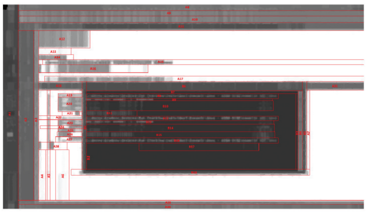
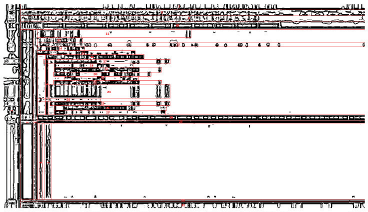
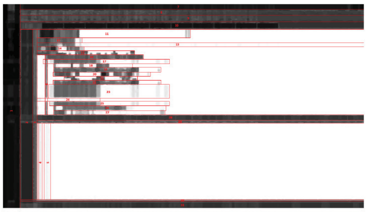
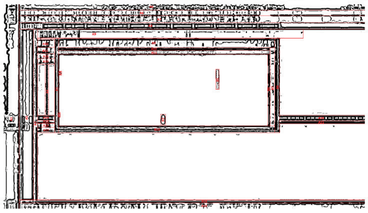
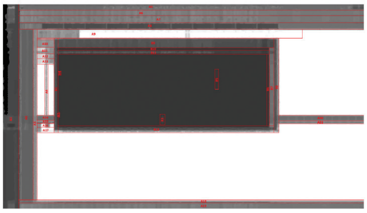
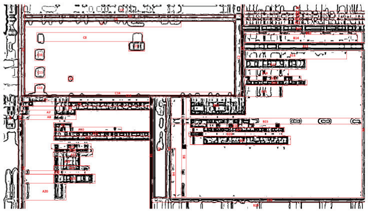
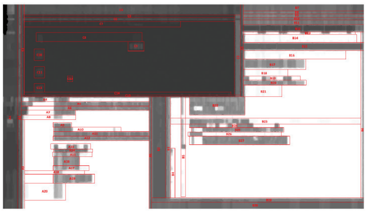
Frame	New Version	
	Edge Information	Bounding Boxes
Frame 225		
Frame 267		
Frame 691		
Frame 703		
Frame 803		

Table 8: AOI Bounding Boxes for the Automatic AOI Bounding Boxes Estimation Algorithm (old version) (part 1) [23, 24, 34].

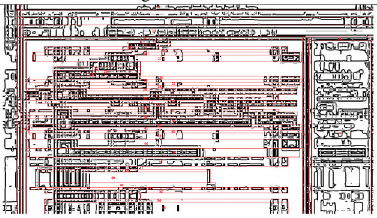
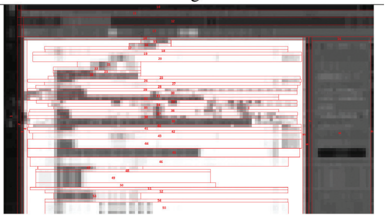
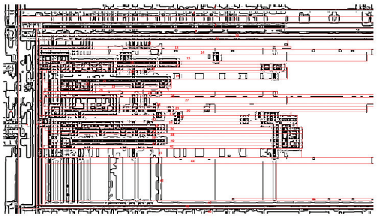
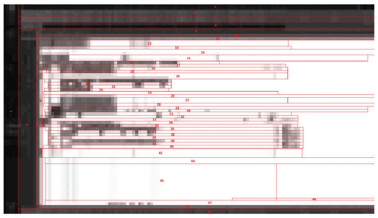
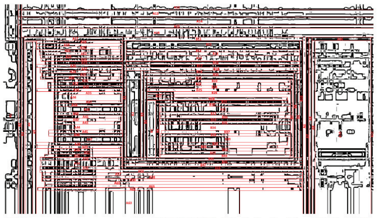
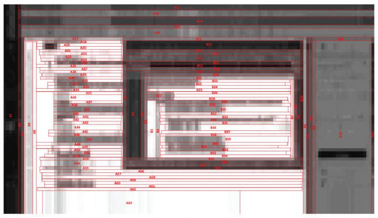
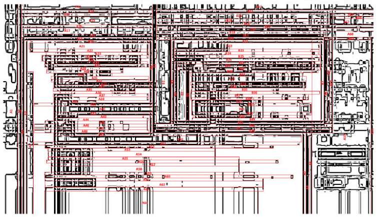
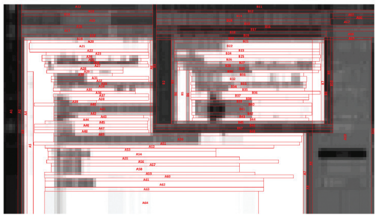
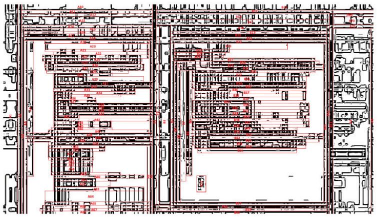
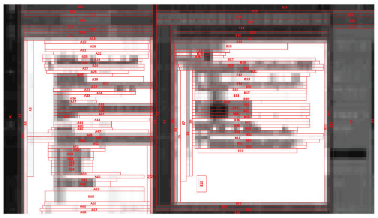
Frame	Old Version	
	Edge Information	Bounding Boxes
Frame 18		
Frame 125		
Frame 139		
Frame 170		
Frame 176		

Table 9: AOI Bounding Boxes for the Automatic AOI Bounding Boxes Estimation Algorithm (old version) (part 2) [23, 24, 34].

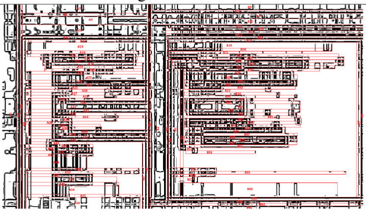
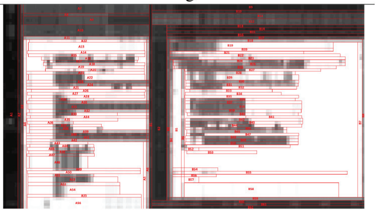
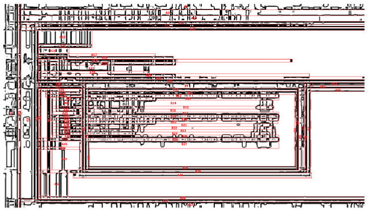
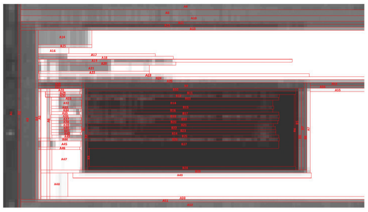
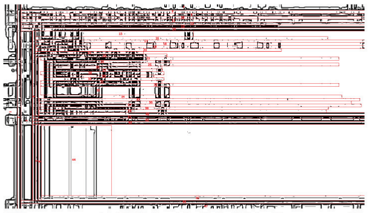
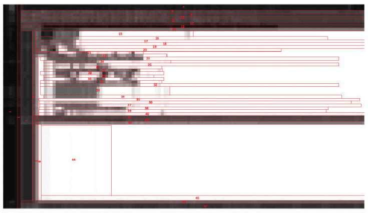
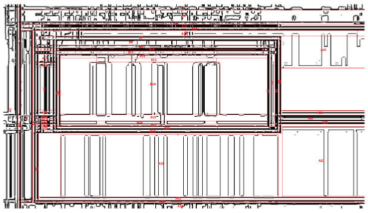
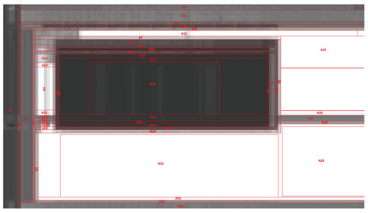
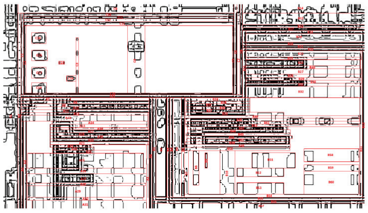
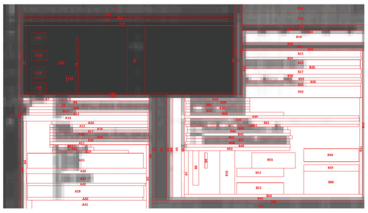
Frame	Old Version	
	Edge Information	Bounding Boxes
Frame 225		
Frame 267		
Frame 691		
Frame 703		
Frame 803		

Table 10: Benchmark results for the Enhanced Automatic AOI Bounding Boxes Estimation Algorithm on selected frames (new version) [22].

Attempt	Frame 18	Frame 125	Frame 139	Frame 170	Frame 176	Frame 225	Frame 267	Frame 691	Frame 703	Frame 803
1	25.37s	20.79s	25.89s	24.58s	25.86s	25.00s	21.21s	19.65s	18.75s	19.94s
2	24.11s	20.89s	25.85s	24.52s	26.04s	24.82s	21.25s	19.55s	18.14s	19.96s
3	23.99s	20.62s	25.93s	24.39s	25.53s	24.65s	21.15s	19.23s	18.07s	19.85s
4	24.01s	20.69s	25.87s	24.44s	25.64s	24.68s	21.11s	19.39s	18.14s	19.80s
5	24.04s	20.66s	25.68s	24.35s	25.62s	24.64s	21.09s	19.45s	17.98s	19.87s
Average	24.31s	20.73s	25.84s	24.46s	25.74s	24.76s	21.16s	19.45s	18.21s	19.88s

Table 11: Benchmark results for the Automatic AOI Bounding Boxes Estimation Algorithm on selected frames (old version) [22].

Attempt	Frame 18	Frame 125	Frame 139	Frame 170	Frame 176	Frame 225	Frame 267	Frame 691	Frame 703	Frame 803
1	34.31s	34.45s	34.97s	34.37s	34.00s	34.36s	34.40s	34.50s	35.10s	33.96s
2	34.46s	34.34s	34.40s	34.41s	34.37s	34.31s	34.28s	34.52s	34.10s	33.99s
3	34.58s	34.68s	34.64s	34.17s	34.81s	34.29s	34.55s	34.27s	34.33s	34.38s
4	34.18s	34.38s	34.59s	34.15s	34.66s	34.25s	34.60s	34.04s	34.28s	34.40s
5	34.34s	33.87s	34.25s	34.60s	34.24s	34.64s	34.39s	34.03s	33.88s	34.48s
Average	34.37s	34.34s	34.57s	34.34s	34.42s	34.37s	34.45s	34.27s	34.34s	34.24s

Table 12: SLIC Components and RAG Cut Threshold Parameter testing results (part 1a).

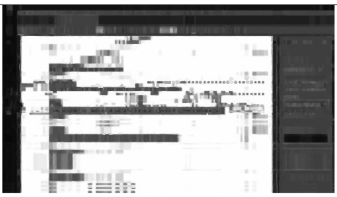
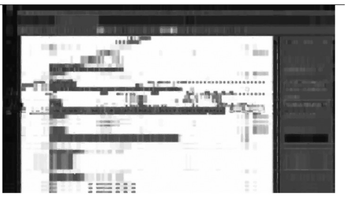
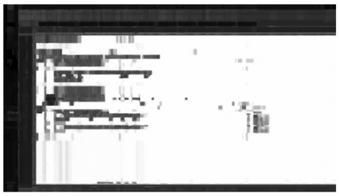
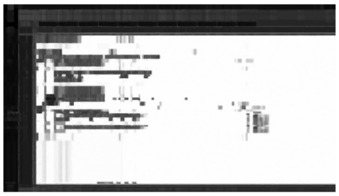
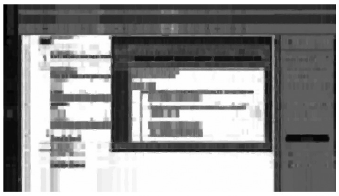
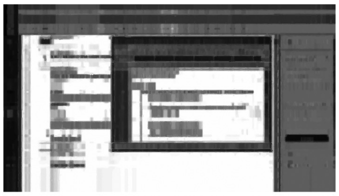
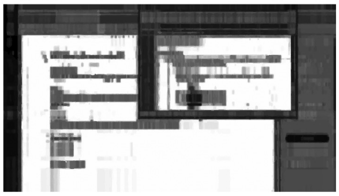
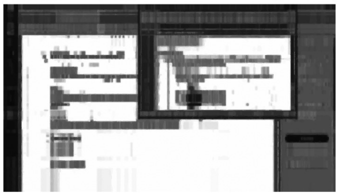
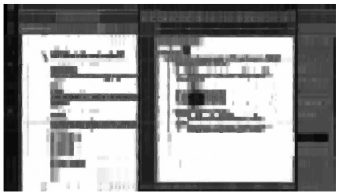
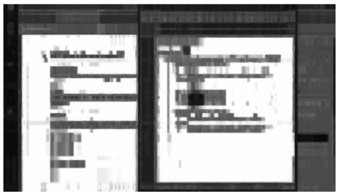
RAG Cut Threshold	3	
SLIC Components	5000	7000
Frame 18		
Frame 125		
Frame 139		
Frame 170		
Frame 176		

Table 13: SLIC Components and RAG Cut Threshold Parameter testing results (part 1b).

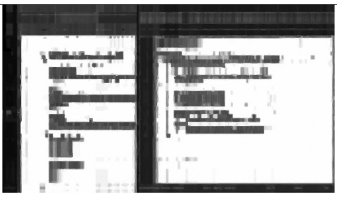
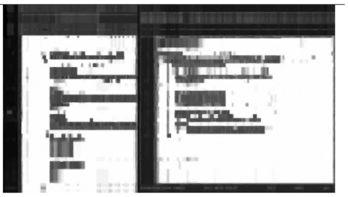
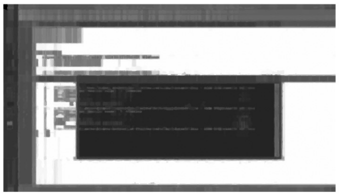
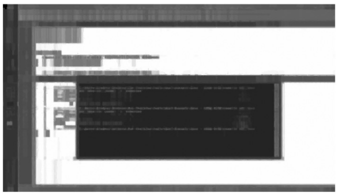
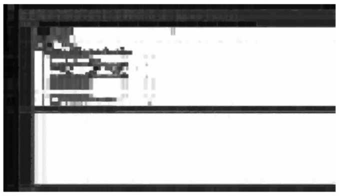
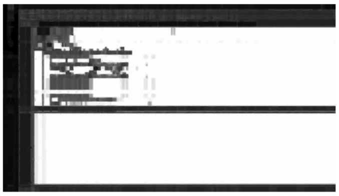
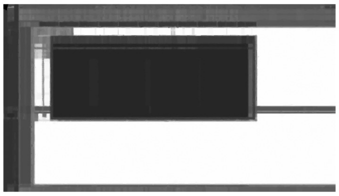
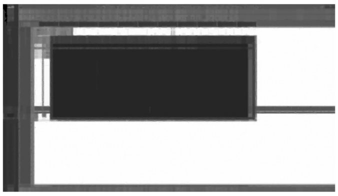
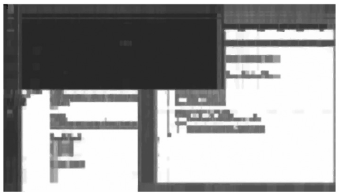
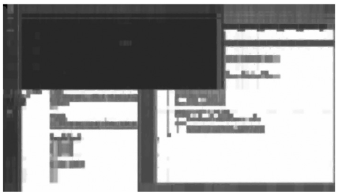
RAG Cut Threshold	3	
SLIC Components	5000	7000
Frame 225		
Frame 267		
Frame 691		
Frame 703		
Frame 803		

Table 14: SLIC Components and RAG Cut Threshold Parameter testing results (part 2a).

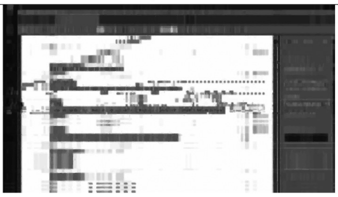
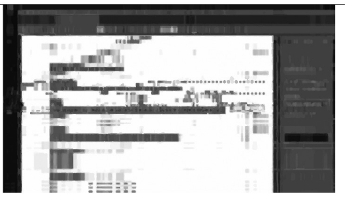
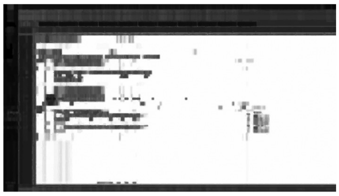
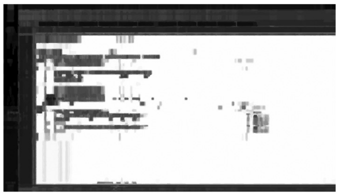
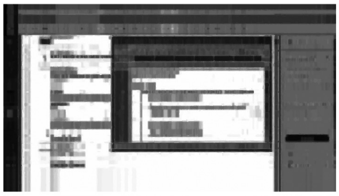
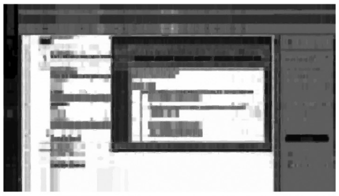
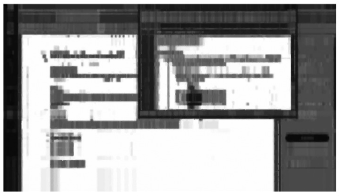
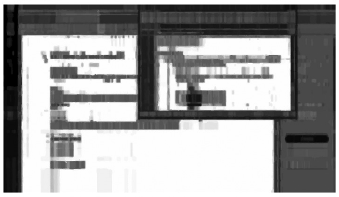
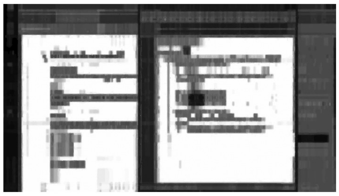
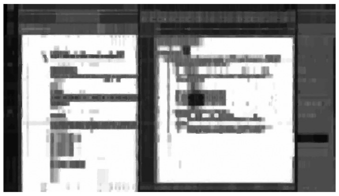
RAG Cut Threshold SLIC Components	3 9000	5 5000
Frame 18		
Frame 125		
Frame 139		
Frame 170		
Frame 176		

Table 15: SLIC Components and RAG Cut Threshold Parameter testing results (part 2b).

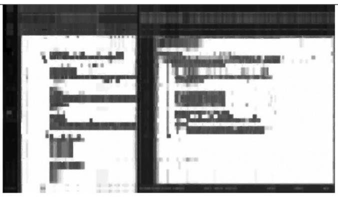
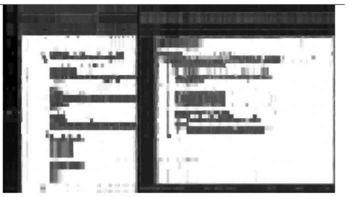
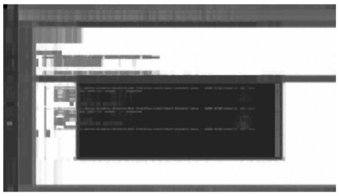
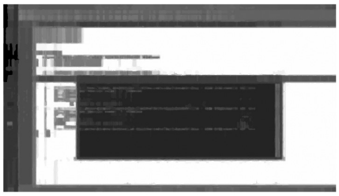
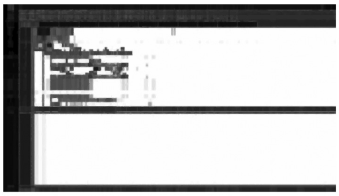
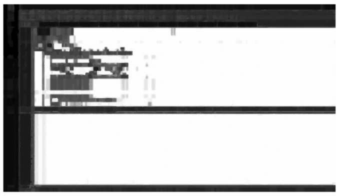
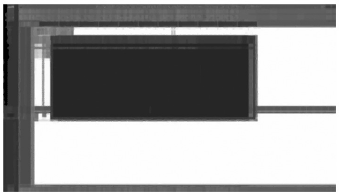
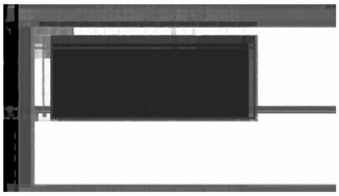
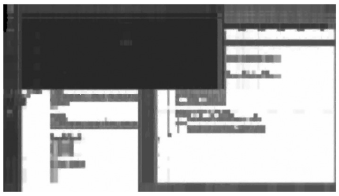
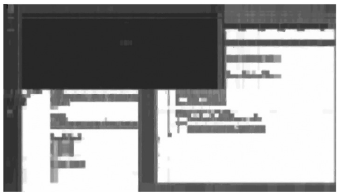
RAG Cut Threshold SLIC Components	3 9000	5 5000
Frame 225		
Frame 267		
Frame 691		
Frame 703		
Frame 803		

Table 16: SLIC Components and RAG Cut Threshold Parameter testing results (part 3a).

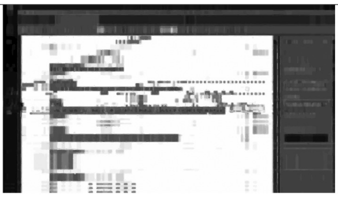
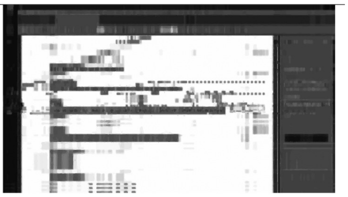
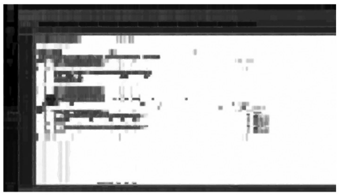
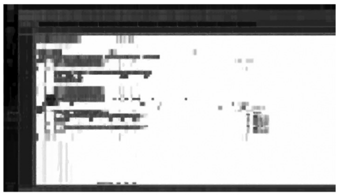
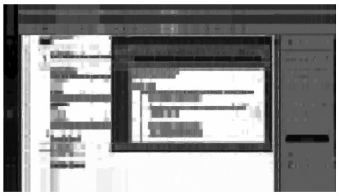
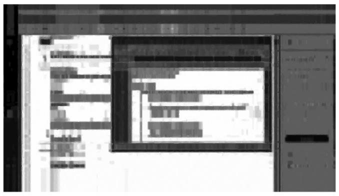
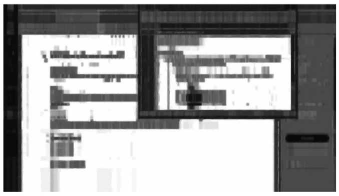
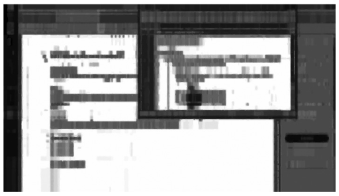
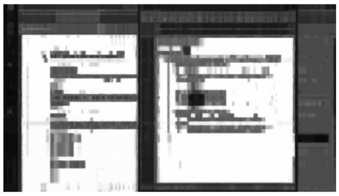
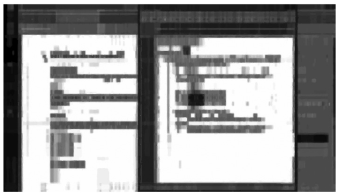
RAG Cut Threshold	5	
SLIC Components	7000	9000
Frame 18		
Frame 125		
Frame 139		
Frame 170		
Frame 176		

Table 17: SLIC Components and RAG Cut Threshold Parameter testing results (part 3b).

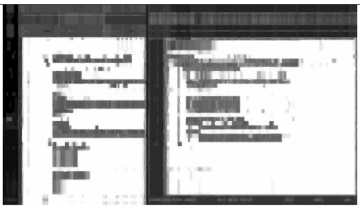
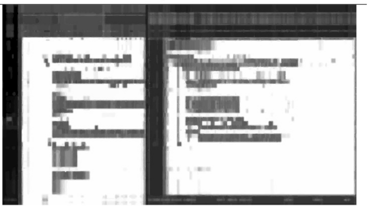
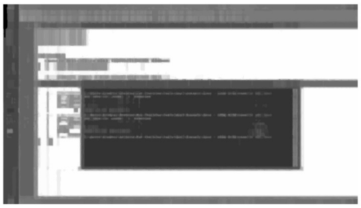
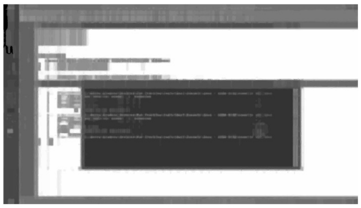
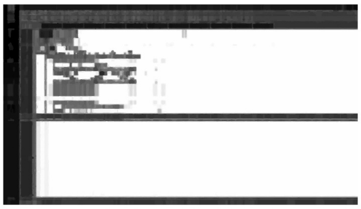
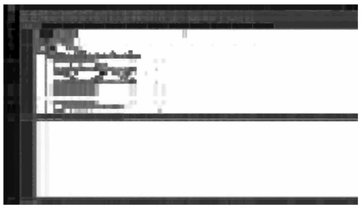
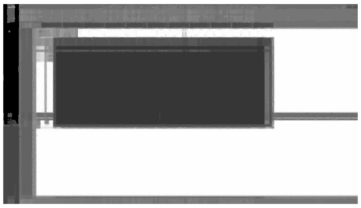
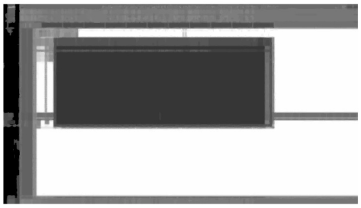
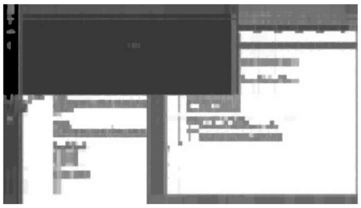
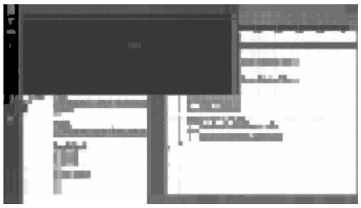
RAG Cut Threshold	5	
SLIC Components	7000	9000
Frame 225		
Frame 267		
Frame 691		
Frame 703		
Frame 803		

Table 18: Comparison of results when the temp image file(s) are saved in PNG format in the new algorithm (part 1).

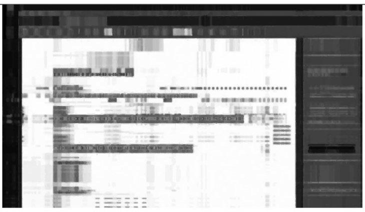
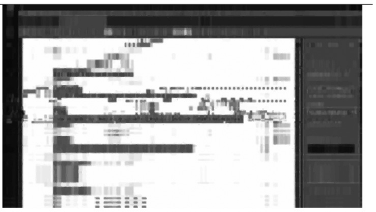
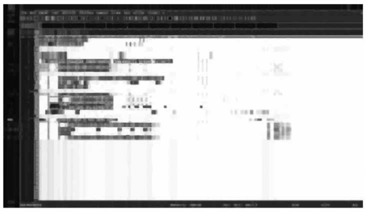
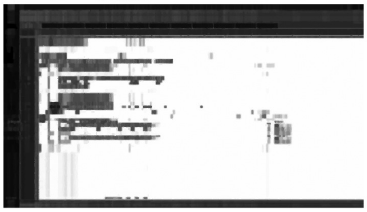
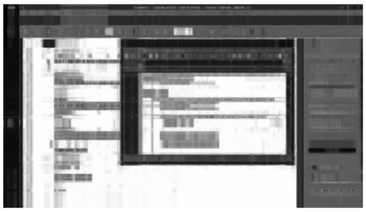
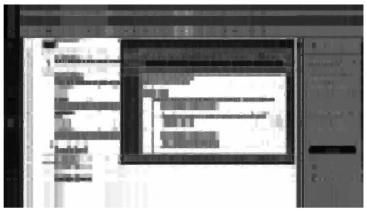
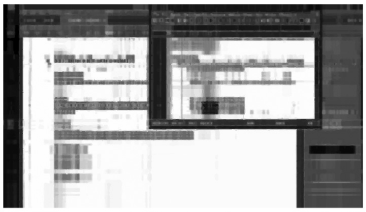
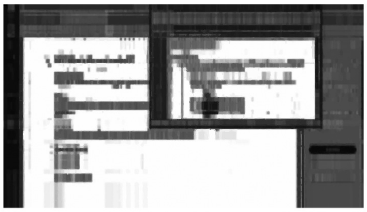
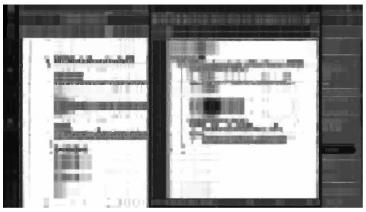
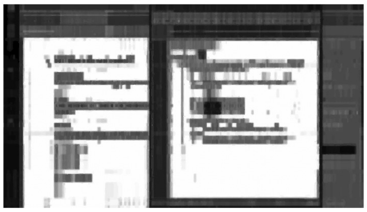
	PNG	JPEG
Frame 18		
Frame 125		
Frame 139		
Frame 170		
Frame 176		

Table 19: Comparison of results when the temp image file(s) are saved in PNG format in the new algorithm (part 2).

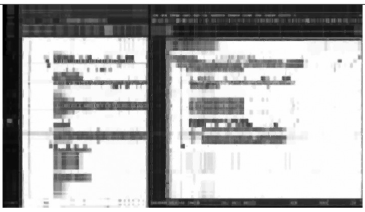
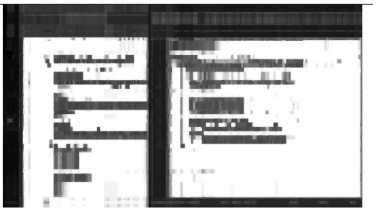
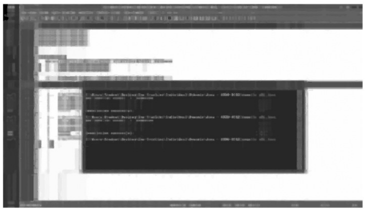
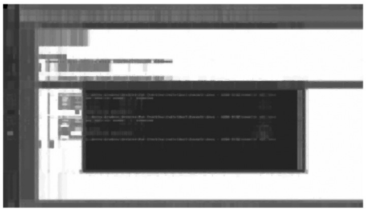
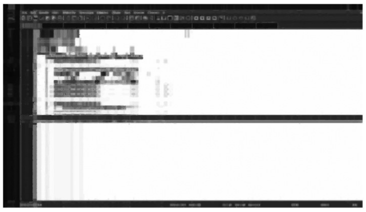
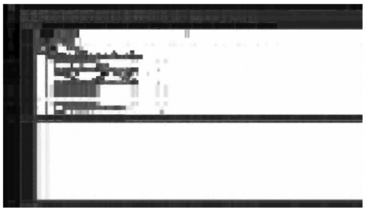
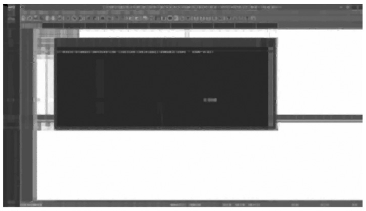
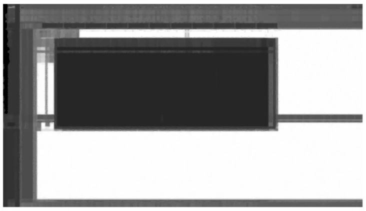
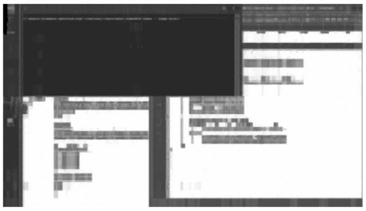
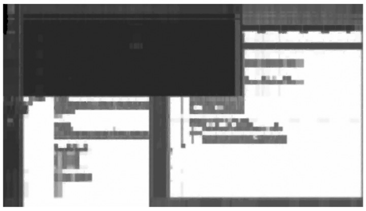
	PNG	JPEG
Frame 225		
Frame 267		
Frame 691		
Frame 703		
Frame 803		

Table 20: Overlapping rectangles evaluation algorithm results for the old and new Automatic AOI Bounding Boxes Estimation Algorithm [37, 38].

	Recall			Precision			Harmonic mean		Final single
	ICDAR			ICDAR			ICDAR		performance
Algorithm	2003	Score	Overall %	2003	Score	Overall %	2003	Score	value (%)
Old	0.56	0.40	42.40	0.29	0.28	28.80	0.38	0.33	34.30
New	0.64	0.50	51.10	0.41	0.34	36.00	0.50	0.41	42.20

Biographies

Ezekiel Adriel D. Lagmay received a degree in Computer Science with Specialization in Data Science and Analytics from the Ateneo de Manila University in 2018 and received his Masters degree in Computer Science in 2020. He is currently a Learning Management Systems (LMS) Analysis Research Assistant for the Ateneo Laboratory for the Learning Sciences. His main research interests are Eye-Tracking, Learning Management Systems Analysis, Web Development, and Software Engineering.

Maria Mercedes T. Rodrigo is a professor of the Department of Information Systems and Computer Science at Ateneo de Manila University, and established the Ateneo Laboratory for the Learning Sciences back in 2011. Her areas of specialization are educational technology, intelligent tutoring systems, and affective computing. She is currently studying students' behavior and affect while using intelligent tutors, educational games, and other learning software.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk, "SLIC Superpixels," *EPFL Technical Report*, 149300, 2010.

[2] A. Almutawakel, "OpenCV: Filters & Arithmetic Operations," 2017, https://medium.com/@almutawakel.ali/opencv-filters-arithmetic-operations-2f4ff236d6aa.

[3] AWS, "AWS Free Tier," https://aws.amazon.com/free/?nc2=h_ql_pr_ft&all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=tier%2312monthsfree%7Ctier%23always-free&awsf.Free%20Tier%20Categories=categories%23ai-ml.

[4] G. Bertasius, L. Torresani, and J. Shi, "Object Detection in Video with Spatiotemporal Sampling Networks," *ECCV 2018*, 2018.

[5] W. Cao, J. Yuan, Z. He, Z. Zhang, and Z. He, "Fast Deep Neural Networks With Knowledge Guided Training and Predicted Regions of Interests for Real-Time Video Object Detection," *Special Section on Sequential Data Modeling and Its Emerging Applications*, 6, 2018, 8990–9.

[6] Z. Chen, Z. Qi, F. Meng, L. Cui, and Y. Shi, "Image Segmentation via Improving Clustering Algorithms with Density and Distance," *Procedia Computer Science*, 55, 2015, 1015–22.

[7] A. Cranz, "Apple and Nvidia Are Over," 2019, https://gizmodo.com/apple-and-nvidia-are-over-1840015246.

[8] N. Dhanachandra, K. Manglem, and Y. J. Chanu, "Image Segmentation using K-Means Clustering Algorithm and Subtractive Clustering Algorithm," *Procedia Computer Science*, 54, 2015, 764–71.

[9] A. T. Duchowski, *Eye Tracking Methodology: Theory and Practice*, 2nd ed., Springer-Verlag, 2007.

[10] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed., Pearson, 2008.

[11] Google, "Google Cloud Free Program," 2021, https://cloud.google.com/free/docs/gcp-free-tier.

[12] M. Grundmann, V. Kwatra, M. Han, and I. Essa, "Efficient Hierarchical Graph-Based Video Segmentation," in *Computer Vision and Pattern Recognition (CVPR)*, 2010.

[13] S. Gupta, D. Palsetia, M. M. A. Patwary, A. Agrawal, and A. Choudhary, "A New Parallel Algorithm for Two-pass Connected Component Labeling," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, 2014, 1355–62.

[14] J. Huang and Z.-N. Li, "Automatic Detection of Object of Interest and Tracking in Active Video," *The Journal of Signal Processing Systems*, 65(9), 2010, 49–62.

[15] A. Kim, "FastSLIC: Optimized SLIC Superpixel."

[16] E. A. D. Lagmay, *Development of an Algorithm for Dividing Video Stimuli into Areas of Interests (AOI) For Dynamic Eye-Tracking Data Pre-Processing*, Ateneo de Manila University, 2020.

[17] E. A. D. Lagmay and M. M. T. Rodrigo, "A Method for Automatically Estimating Areas of Interest Boundaries for Text Areas," in *Proceedings of Information and Computing Education Conference (ICE2019)*, Davao City, Philippines, 2019.

[18] J. Liang, I. T. Phillips, and R. M. Haralick, "Performance evaluation of document layout analysis algorithms on the UW data set," *Document Recognition IV, Proceedings of the SPIE*, 1997, 149–60.

[19] B. Liu, M. Sun, Q. Liu, A. Kassam, C.-C. Li, and R. J. Sclabassi, "Automatic Detection of Region of Interest Based on Object Tracking in Neurosurgical Video," in *Proceedings of the 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, Shanghai, China, 2005, 6273–6.

[20] Microsoft, "Azure Machine Learning Pricing," https://azure.microsoft.com/en-us/pricing/details/machine-learning/%5C#pricing.

[21] A. Mordvintsev, K. Abid, and eastWillow, "Smoothing Images," 2016, https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html.

[22] NPE, "Measure Time Elapsed in Python (Answer)," 2011, https://stackoverflow.com/questions/7370801/measure-time-elapsed-in-python.

[23] OpenCV, "Image Thresholding," 2017, https://docs.opencv.org/3.4.0/d7/d4d/tutorial_py_thresholding.html.

[24] OpenCV, "Miscellaneous Image Transformations," 2018, https://docs.opencv.org/3.4.3/d7/d1b/group__imgproc__misc.html.

[25] K. Panetta, Q. Wan, S. Rajeev, A. Kaszowska, A. L. Gardony, K. Naranjo, H. A. Taylor, and S. Agaian, "ISeeColor: Method for Advanced Visual Analytics of Eye Tracking Data," *IEEE Access*, 8, 2020, 52278–87.

[26] Pythontic.com, "Sharpen-filter Using Pillow – The Python Image Processing Library," https://pythontic.com/image-processing/pillow/sharpen-filter.

[27] A. Rosebrock, "Intersection Over Union (IoU) for Object Detection," 2016, https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[28] Scikit-Image, "Histogram Equalization," https://scikit-image.org/docs/0.9.x/auto_examples/plot_equalize.html.

[29] Scikit-Image, "Histogram Equalization (Examples)," https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_equalize.html.

[30] Scikit-Image, "RAG Thresholding," https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_rag_mean_color.html.

[31] Scikit-Image, "slic," https://scikit-image.org/docs/dev/api/skimage.segmentation.html#skimage.segmentation.slic.

[32] S. Sudhakar, "Histogram Equalization," 2017, https://towardsdatascience.com/histogram-equalization-5d1013626e64.

[33] C. L. S. Tablatin and M. M. T. Rodrigo, "Analysis of Static Code Reading Patterns," in *18th Philippine Computing Science Congress*, Cagayan de Oro City, Philippines, 2018, 32–43.

[34] TutorialsPoint, "OpenCV - Adaptive Threshold," https://www.tutorialspoint.com/opencv/opencv_adaptive_threshold.htm.

[35] UCI, "Histogram Equalization," https://www.math.uci.edu/icamp/courses/math77c/demos/hist_eq.pdf.

[36] M. M. Villamor and M. M. T. Rodrigo, “Do Friends Collaborate and Perform Better?: A Pair Program Tracing and Debugging Eye Tracking Experiment,” in *18th Philippine Computing Science Congress*, Cagayan de Oro City, Philippines, 2018, 9–16.

[37] C. Wolf and M. Bacconnier, “DetEval – Evaluation Software for Object Detection Algorithms,” https://perso.liris.cnrs.fr/christian.wolf/software/deteval/index.html.

[38] C. Wolf and J.-M. Jolion, “Object Count/Area Graphs for the Evaluation of Object Detection and Segmentation Algorithms,” *Technical Report LIRIS-RR-2005-024*, 2005.

[39] J. Wolf, S. Hess, D. Bachmann, Q. Lohmeyer, and M. Meboldt, “Automating Areas of Interest Analysis in Mobile Eye Tracking Experiments based on Machine Learning,” *The Journal of Eye Movement Research*, 11(6), 2018, https://dx.doi.org/10.16910%2Fjemr.11.6.6.

[40] M. Wuerthele, “OpenGL, OpenCL deprecated in favor of Metal 2 in macOS 10.14 Mojave,” 2018, https://appleinsider.com/articles/18/06/04/opengl-opencl-deprecated-in-favor-of-metal-2-in-macos-1014-mojave.