Application of The Bounding Box Method For OKU Timur Character Image Segmentation

M. Fikri[1], Ilman Zuhri Yadi[1*], Yesi Novaria Kunang[1] , Leon Andretti Abdillah[1,2,3]
[1]Intelligent Systems Research Group, FakultasSainsTeknologi, Universitas Bina Darma, Palembang, Indonesia
[2]Research Fellow, INTI International University, Nilai, Malaysia
[3]Research Fellow, Chung Hua University, Hsinchu City, Taiwan

Corresponding author: Ilman Zuhri Yadi (e-mail:ilmanzuhriyadi@binadarma.ac.id).

ABSTRACT Ogan Komering Ulu (OKU) Timur is a region rich in cultural heritage, including historical relics such as traditional houses, traditional clothing, handicrafts, and traditional dances, as well as Ulu script. One of the important intangible cultural heritages from this area is the presence of ancient script artifacts, such as the Pallava script known as AksaraUlu or SuratUlu, recognized in the variant of Ulu script from OKU Timur, which reflects the development of civilization and writing systems of the past. This research aims to segment images of the OKU Timur script using the Bounding Box method. The segmentation process is carried out using the Python programming language and the Google Colaboratory platform. The segmentation of this image is very important in the effort to preserve and digitize cultural heritage in the modern era. The methods used in this research include data collection from artifacts in OKU Timur, processing images into binary format, and character separation using Bounding Box. The results of this study are an effective segmentation of the characters of the OKU Timur script, which can serve as a foundation for creating a dataset used for image classification.

KEYWORDS OKUTimur Character; Bounding Box; Image Segmentation

I. INTRODUCTION

OganKomeringUlu (OKU) East possesses a rich cultural heritage, particularly in relation to its historical artifacts and local traditions. OKU East is a region abundant in cultural legacy, including traditional houses, handicrafts, and dances passed down through generations. The region is also home to invaluable cultural relics, such as ancient inscriptions, that hold significant historical value. Various inscriptions and manuscripts written in Pallawa script and Ulu script have been discovered in this area, serving as critical evidence of the development of past civilizations and writing systems.

The visual symbol system known as script or writing system is used to express the expressive elements of language by inscribing them onto mediums such as paper, stone, wood, fabric, etc[1]. The Oku East script is a traditional writing system originating from the Oku East region in South Sumatra, Indonesia. This script holds distinct uniqueness and beauty, reflecting the cultural identity and historical legacy of the local community. In an effort to preserve this cultural heritage in the digital era, modern technology, such as image segmentation, can play a crucial role.

Image segmentation refers to the process of separating an object from its background, allowing the object to undergo further processing[2]. This research utilizes the Bounding Box method, a widely used data labeling annotation technique in computer vision. The method involves drawing rectangular boxes to locate target objects, enabling each character to be identified and separated from other elements within the image. The aim of this study is to segment the characters in the Oku East script. Therefore, this research is titled "Application Of The Bounding Box Method For OKU Timur Character Image Segmentation".

II. LITERATURE REVIEW

A. THEORETICAL FRAMEWORK

1) SEGMENTATION

Segmentation is a crucial aspect of image analysis, as it involves analyzing the desired image for further processes, such as pattern recognition[3].

2) IMAGE SEGMENTATION

Segmentation enables each object in an image to be isolated, allowing them to be used as input for other processes[4]. For example, segmentation is employed in human face recognition to distinguish the face from the background or other body parts, producing an image of the face to be identified. It is also used in object recognition to differentiate each object from the background, ensuring that the background is not processed during the recognition phase. Similar to the letter recognition process in text, segmentation is also required to identify the letters to be recognized.

3) BOUNDING BOX

The bounding process involves separating objects from one another and computing their features to identify each object. A bounding box is typically rectangular and is defined by the coordinates of the top-left corner (x_min, y_min) and the bottom-right corner (x_max, y_max), or by the center, width, and height. In a dilated image, the bounding process is used to separate row block objects. This process performs object segmentation by separating one object from another based on pixel connectivity in the dilated image. Subsequently, the bounding process computes object features to label individual objects within the row block. Objects are marked with red boxes, using the computed width and height dimensions derived from the labeling process[5].

4) DILATION

Dilation is a process aimed at thickening the white points on objects being detected, thereby making them more readily identifiable by computers[6].

Dilation is a morphological image processing method related to the shape of object structures. Specifically, dilation involves adding pixels to the boundaries of objects in a digital image. In other words, dilation is the process of expanding the boundaries of objects by adding pixels, resulting in an increase in the size of the image after the dilation operation is performed[7].

5) MORPHOLOGICAL OPERATIONS

Morphological image processing refers to a crucial technique in image processing that alters the shape and structure of objects within the original image[8]. Morphological operations are employed to make shapes (structures) more recognizable. This type of image processing is typically performed by applying a structural element to the image in a manner similar to convolution. The structural element (SE) is a fundamental component for morphological operations[9].

III. RESEARCH METHODOLOGY

In this research on the segmentation of Oku East script images, the researcher employs the Bounding Box method. The Bounding Box method is utilized to mark objects that have been grouped during object segmentation. The objects are marked using green boxes. The following are the methodological steps that can be used:

1. Data Collection
2. Row Segmentation
3. Character Segmentation using Bounding Box
4. Save Image

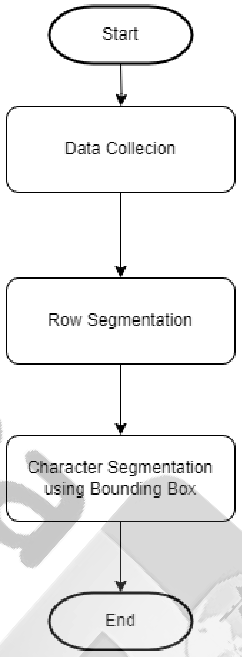


Figure 1. Flowchart

The methodological steps employed in image segmentation are carried out as follows:

A. DATA COLLECTION

The initial stage of this research involves data collection from script artifacts in the Oku East region, which are then processed by researchers involved in the ISRG project titled "Development of a Mobile-Based Ulu Script Transliteration Application for the OKU East Variant." The Oku East script is converted into a questionnaire for processing during the Image Preprocessing stage, which has been carried out by another team within the ISRG project. The questionnaire consists of 102 respondents, with each questionnaire containing 225 Oku East script characters. Below is an example of a page from the respondent's questionnaire.

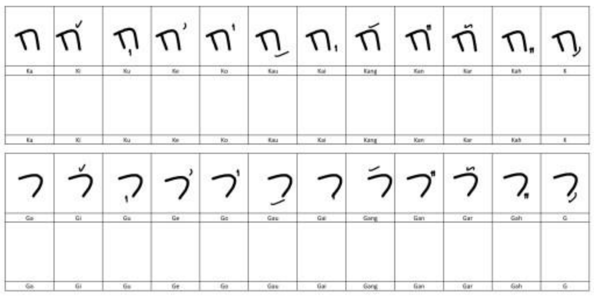


Figure 2.Sample Questionnaire of OKU East Script

B. ROW SEGMENTATION

After completing the Image Preprocessing stage, the resulting binary images facilitate the identification and extraction of text rows, as each text row can now be recognized as a cluster of isolated black pixels against a white background. By applying techniques such as connected component analysis or horizontal line detection, text rows can be effectively extracted for further processing. The objective of row segmentation is to determine the number of character rows present in the image and to identify the area of each character row. This is done to exclude unnecessary components from subsequent processing stages[10].

C. CHARACTER SEGMENTATION USING BOUNDING BOX

After successfully segmenting the text rows, the next step is to separate each character within those rows. This process typically begins with the detection of spaces between words, which appear as wider horizontal gaps between clusters of black pixels. Image processing algorithms then place bounding boxes around each detected character. These bounding boxes are rectangular shapes that encompass each character, identifying the top, bottom, left, and right coordinates of the character. By using bounding boxes, each character can be extracted as a distinct entity.

The Bounding Box process involves marking row blocks with boxes, which allows for the identification of individual objects. The following describes the testing of the dilation processing stage:

1. The Bounding Box process marks row blocks in the dilated image.
2. The row blocks are treated as individual objects within the dilated image[5].

IV. RESULTS AND DISCUSSION

A. DATA COLLECTION

The initial stage of this research involves the data collection process from script artifacts located in the OKU East region. The collected data is then processed by researchers involved in the ISRG research team for the study titled "Development of a Mobile-Based Transliteration Application for the Ulu Script Variant of OKU East." In this research, a questionnaire is carefully designed to encompass various character variations of the OKU East script. Each questionnaire contains 225 characters of the OKU East script, which will then be distributed to respondents for completion.

Subsequently, the collected Oku East script is processed and converted into a questionnaire. This questionnaire will be used in the Image Preprocessing stage. Below is an example of a page from the completed questionnaire, which has been processed during the Image Preprocessing stage.

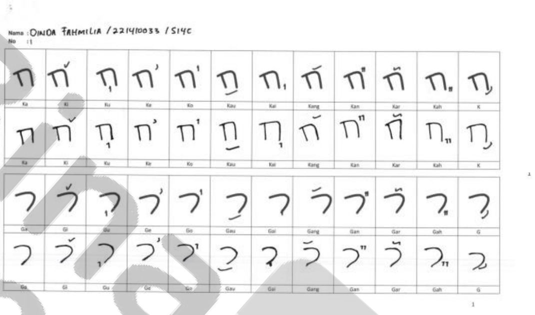
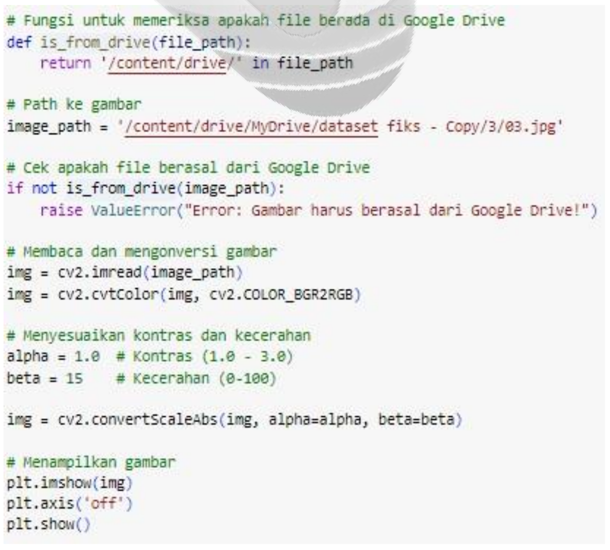


Figure 3.Questionnaire Page

B. ROW SEGMENTATION

1) DISPLAYING EAST OKU SCRIPT IMAGES



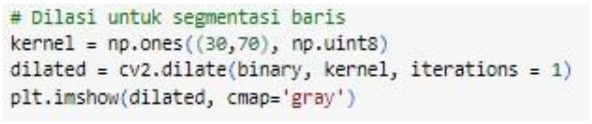
```
# Fungsi untuk memeriksa apakah file berada di Google Drive
def is_from_drive(file_path):
    return '/content/drive/' in file_path

# Path ke gambar
image_path = '/content/drive/MyDrive/dataset fiks - Copy/3/03.jpg'

# Cek apakah file berasal dari Google Drive
if not is_from_drive(image_path):
    raise ValueError("Error: Gambar harus berasal dari Google Drive!")

# Membaca dan mengonversi gambar
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Menyesuaikan kontras dan kecerahan
alpha = 1.0  # Kontras (1.0 - 3.0)
beta = 15    # Kecerahan (0-100)

img = cv2.convertScaleAbs(img, alpha=alpha, beta=beta)

# Menampilkan gambar
plt.imshow(img)
plt.axis('off')
plt.show()
```

Figure 4. Code For Displaying Images

The images processed in this stage are sourced from the previous step, specifically the Image Preprocessing stage, and are uploaded to Google Drive for processing in the segmentation stage. The code provided is designed to process and display images stored in Google Drive, first verifying the origin of the file. The initial step in the code is to check if the image file is located in Google Drive by searching for the string '/content/drive/' in the file path. If the image is not from this location, the code will terminate the process and provide an error message. Once verification is successful, the image is read using OpenCV, which by default reads images in BGR (Blue-Green-Red) color format. To be compatible with Matplotlib, the image is then converted to RGB (Red-Green-Blue) color format, which is more commonly used for visualization.

The next step in the code involves adjusting the contrast and brightness of the image. These adjustments are controlled through the variables alpha for contrast and beta for brightness. The alpha value governs the level of contrast, where a value of 1.0 maintains the original contrast, while higher values enhance the contrast of the image. Meanwhile, beta regulates the brightness, with higher values making the image appear brighter. These adjustments are applied to the image using the `cv2.convertScaleAbs()` function. After adjusting the contrast and brightness, the image is displayed using Matplotlib with the display axes hidden to focus solely on the image. Thus, this code enables users to read, verify, adjust, and display images from Google Drive with easy control over contrast and brightness.

2) DILATION ROW SEGMENTATION

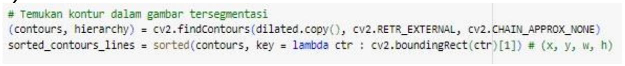


```
# Dilasi untuk segmentasi baris
kernel = np.ones((30,70), np.uint8)
dilated = cv2.dilate(binary, kernel, iterations = 1)
plt.imshow(dilated, cmap='gray')
```

Figure 5. Code Dilation Row Segmentation

After successfully displaying the image, it is processed in the dilation stage. The code applies dilation to the binary image, which is a technique in image processing used to expand the area of bright objects in the image. Initially, a kernel or structuring element of size 30x70 pixels is created using `np.ones`, where each element has a value of 1. This kernel functions as a "stamp" used in the dilation process. Subsequently, the `cv2.dilate` function is applied to the binary image (`binary`) with the kernel, and dilation is performed once (as specified by `iterations = 1`). This dilation makes the bright objects in the image thicker or larger, which is useful for connecting separated elements, such as text rows in document segmentation.

3) FINDING CONTOURS IN THE IMAGE



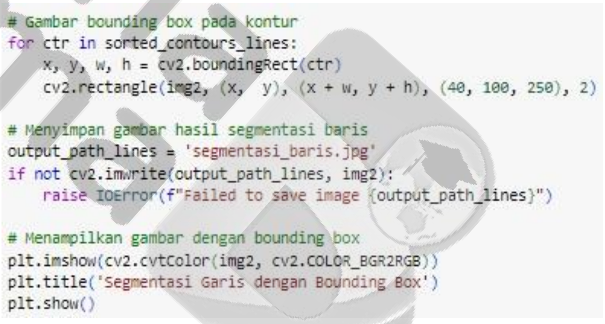
```
# Temukan kontur dalam gambar tersegmentasi
(contours, hierarchy) = cv2.findContours(dilated.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
sorted_contours_lines = sorted(contours, key = lambda ctr : cv2.boundingRect(ctr)[1]) # (x, y, w, h)
```

Figure 6.Code Finding Contours

The code above is used to find and sort contours in an image that has undergone dilation. First, the `cv2.findContours` function is employed to detect contours in the segmented image (which has previously undergone dilation). This function returns two values: `contours`, which contains a list of all detected contours, and `hierarchy`, which provides information about the hierarchical relationships among the contours. The parameter `cv2.RETR_EXTERNAL` ensures that only the outermost contours are retrieved, while `cv2.CHAIN_APPROX_NONE` retains all points in the contours without approximation.

After the contours are detected, the code sorts them based on their vertical position (y-coordinate) using the `sorted` function. This sorting process is crucial in applications such as text line segmentation, where the order of contours determines the sequence of lines. The function `cv2.boundingRect(ctr)` is used to obtain the coordinates and dimensions of the bounding box for each contour, and `key = lambda ctr: cv2.boundingRect(ctr)[1]` ensures that the sorting is based on the y-value, which represents the vertical position of the contours in the image.

4) SAVING AND DISPLAYING THE RESULTS OF ROW SEGMENTATION IMAGES



```
# Gambar bounding box pada kontur
for ctr in sorted_contours_lines:
    x, y, w, h = cv2.boundingRect(ctr)
    cv2.rectangle(img2, (x,  y), (x + w, y + h), (40, 100, 250), 2)

# Menyimpan gambar hasil segmentasi baris
output_path_lines = 'segmentasi_baris.jpg'
if not cv2.imwrite(output_path_lines, img2):
    raise IOError(f"Failed to save image {output_path_lines}")

# Menampilkan gambar dengan bounding box
plt.imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))
plt.title('Segmentasi Garis dengan Bounding Box')
plt.show()
```

Figure 7.Code For Saving And Displaying Image Results

This code is designed to draw bounding boxes around each contour that has been previously sorted and then save and display the resulting image. First, the code iterates through each contour sorted by vertical position (`sorted_contours_lines`). For each contour, the coordinates of the top-left corner (`x, y`), as well as the width (`w`) and height (`h`) of the bounding box, are calculated using `cv2.boundingRect(ctr)`. The bounding box is then drawn on the original image (`img2`) using `cv2.rectangle`, with the box color specified by the RGB value `(40, 100, 250)` and a line thickness of 2 pixels.

After all bounding boxes have been drawn, the resulting image is saved to a file named segmentasi_baris.jpg using cv2.imwrite. If the saving process fails, the program will generate an error message. Finally, the image with bounding boxes is displayed using Matplotlib (plt.imshow), and the image is converted from the BGR color format (used by OpenCV) to the RGB format for proper display. Below is the result of the row segmentation stage.

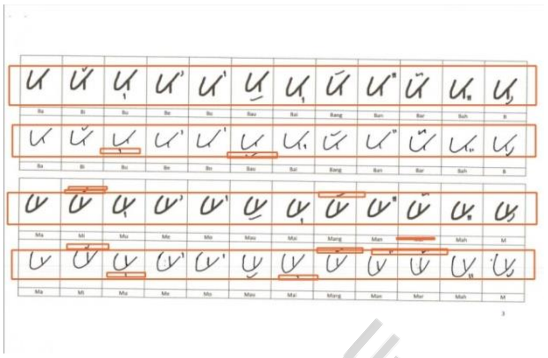
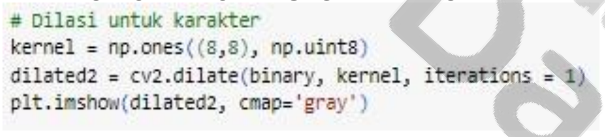


Figure 8. **Results Of The Row Segmentation Stage**

C. CHARACTER SEGMENTATION USING BOUNDING BOX

1) DILATION CHARACTER SEGMENTATION

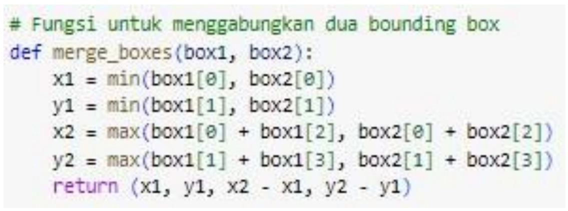


```
# Dilasi untuk karakter
kernel = np.ones((8,8), np.uint8)
dilated2 = cv2.dilate(binary, kernel, iterations = 1)
plt.imshow(dilated2, cmap='gray')
```

Figure 9. **Code Dilation Character Segmentation**

After successfully processing the row segmentation stage, the next step is to perform dilation on the characters in the image. This code applies dilation to the image to expand the character areas. First, a kernel of size 8x8 pixels is created using `np.ones`, where each kernel element has a value of 1. This kernel is used in the dilation process, where the `cv2.dilate` function enlarges the white areas (bright objects) in the binary image (`binary`). The dilation process is performed once, as specified by `iterations = 1`. The result of this dilation is an image with thicker characters and more prominent white areas, which can aid in connecting separated parts of characters or improving the visibility of characters in the image.

2) FUNCTION TO MERGE TWO BOUNDING BOX



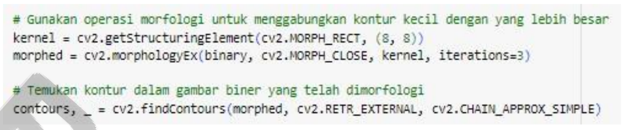
```
# Fungsi untuk menggabungkan dua bounding box
def merge_boxes(box1, box2):
    x1 = min(box1[0], box2[0])
    y1 = min(box1[1], box2[1])
    x2 = max(box1[0] + box1[2], box2[0] + box2[2])
    y2 = max(box1[1] + box1[3], box2[1] + box2[3])
    return (x1, y1, x2 - x1, y2 - y1)
```

Figure 10. **Code To Merge Two Bounding Boxes**

This code defines a function named `merge_boxes` that is used to combine two bounding boxes into a single, larger box that encompasses both of the original boxes. The function accepts two bounding boxes as input, each represented by a tuple `(x, y, w, h)`, where `x` and `y` are the coordinates of the top-left corner, and `w` and `h` are the width and height of the box.

To merge two bounding boxes, the function first determines the coordinates of the top-left corner of the combined box by taking the minimum values of `x` and `y` from the two boxes. Next, it determines the coordinates of the bottom-right corner of the combined box by taking the maximum values of `x + w` and `y + h` from both boxes. Finally, the function returns the merged bounding box in the form `(x1, y1, width, height)`, where `x1` and `y1` are the coordinates of the top-left corner, and `width` and `height` represent the width and height of the combined box.

3) MORPHOLOGICAL OPERATIONS



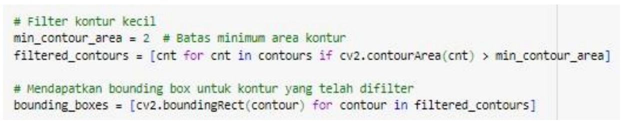
```
# Gunakan operasi morfologi untuk menggabungkan kontur kecil dengan yang lebih besar
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (8, 8))
morphed = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel, iterations=3)

# Temukan kontur dalam gambar biner yang telah dimorfologi
contours, _ = cv2.findContours(morphed, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Figure 11. **Code Morphological Operations**

This code employs morphological operations to merge small contours with larger ones in a binary image. First, a square-shaped kernel of size 8x8 pixels is created using `cv2.getStructuringElement`. This kernel is used in the morphological operation with the method `cv2.MORPH_CLOSE`, which combines adjacent white areas (bright objects) by closing small gaps between them. This process is performed three times (`iterations=3`) to ensure that small contours near larger contours are merged, resulting in larger, more cohesive contours.

After the morphological operation is completed, the contours in the modified image (`morphed`) are detected using `cv2.findContours`. The function `cv2.RETR_EXTERNAL` ensures that only the outer contours are retrieved, while `cv2.CHAIN_APPROX_SIMPLE` is used to simplify the contours by reducing the number of points that make up each contour. The result is a list of larger contours that are combinations of previously separate small contours.

4) CONTOUR FILTERING ON THE IMAGE



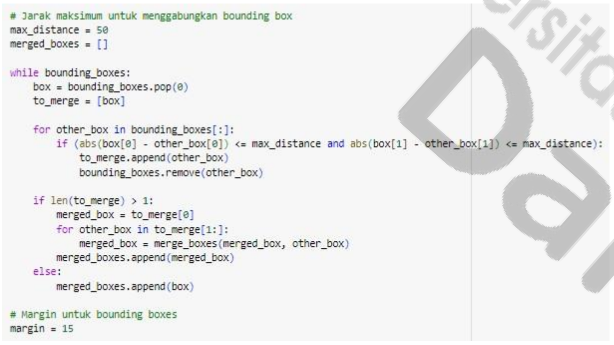
```
# Filter kontur kecil
min_contour_area = 2  # Batas minimum area kontur
filtered_contours = [cnt for cnt in contours if cv2.contourArea(cnt) > min_contour_area]

# Mendapatkan bounding box untuk kontur yang telah difilter
bounding_boxes = [cv2.boundingRect(contour) for contour in filtered_contours]
```

Figure 12. **Code Contour Filtering On The Image**

This code is designed to filter out small contours in the image, retaining only the larger contours. Initially, a minimum contour area threshold is set with `min_contour_area = 2`, meaning only contours with an area greater than 2 pixels will be preserved. Subsequently, contour filtering is performed by creating a new list, `filtered_contours`, which includes only those contours whose area exceeds this threshold. This filtering is accomplished using list comprehension, with `cv2.contourArea(cnt)` employed to calculate the area of each contour.

After the smaller contours are removed, bounding boxes for each of the remaining contours are computed and stored in the list `bounding_boxes`. These bounding boxes are calculated using `cv2.boundingRect(contour)`, which provides the coordinates of the top-left corner, width, and height of the bounding box surrounding each filtered contour. As a result, this code prepares the data for the next stage, where only the relevant and sufficiently large contours will be further processed.

5) DETERMINING THE DISTANCE FOR MERGING BOUNDING BOXES AND MARGIN



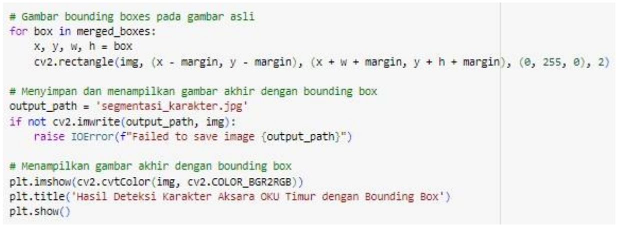
```
# Jarak maksimum untuk menggabungkan bounding box
max_distance = 50
merged_boxes = []

while bounding_boxes:
    box = bounding_boxes.pop(0)
    to_merge = [box]

    for other_box in bounding_boxes[:]:
        if (abs(box[0] - other_box[0]) <= max_distance and abs(box[1] - other_box[1]) <= max_distance):
            to_merge.append(other_box)
            bounding_boxes.remove(other_box)

    if len(to_merge) > 1:
        merged_box = to_merge[0]
        for other_box in to_merge[1:]:
            merged_box = merge_boxes(merged_box, other_box)
        merged_boxes.append(merged_box)
    else:
        merged_boxes.append(box)

# Margin untuk bounding boxes
margin = 15
```

Figure 13.Code To Determine Distance And Margin

This code is designed to merge bounding boxes that are close to each other into a single, larger box. First, the maximum allowed distance for merging bounding boxes is set with `max_distance = 50`. The code then processes each bounding box individually using a `while` loop. For each bounding box, the code searches for other bounding boxes whose distance from the current bounding box does not exceed `max_distance` in both horizontal and vertical directions. All bounding boxes meeting this distance criterion are collected in the list `to_merge`, and the merged bounding boxes are removed from the list `bounding_boxes`.

If there is more than one bounding box in the `to_merge` list, they are merged into a single larger bounding box using the `merge_boxes` function, and the result is stored in `merged_boxes`. If there is only one bounding box in `to_merge`, it is directly added to `merged_boxes`. Additionally, the code sets an extra margin with a value of `margin = 15` to provide spacing around the merged bounding box. This ensures that the combined box covers a sufficiently broad area around it.

6) SAVING AND DISPLAYING THE RESULTING IMAGE



```
# Gambar bounding boxes pada gambar asli
for box in merged_boxes:
    x, y, w, h = box
    cv2.rectangle(img, (x - margin, y - margin), (x + w + margin, y + h + margin), (0, 255, 0), 2)

# Menyimpan dan menampilkan gambar akhir dengan bounding box
output_path = 'segmentasi_karakter.jpg'
if not cv2.imwrite(output_path, img):
    raise IOError(f"Failed to save image {output_path}")

# Menampilkan gambar akhir dengan bounding box
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Hasil Deteksi Karakter Aksara OKU Timur dengan Bounding Box')
plt.show()
```

Figure 14. Code Saving And Displaying The Resulting Image

This code is used to draw the merged bounding boxes on the original image, and then save and display the result. First, the code draws each bounding box listed in `merged_boxes` on the original image (`img`). For each bounding box, the coordinates and dimensions (`x, y, w, h`) are extracted, and the bounding box is drawn around it with an added margin on each side. This margin, set to `margin = 15`, ensures that the bounding box encompasses a surrounding area around the merged bounding boxes.

After all the bounding boxes have been drawn, the image with the bounding boxes is saved to a file named `segmentasi_karakter.jpg` using `cv2.imwrite`. If the image saving process fails, the code will output an error message. Finally, the resulting image with bounding boxes is displayed using Matplotlib. The displayed image is converted from BGR to RGB format to ensure correct visualization and is titled "Character Detection Results for OKU East Script with Bounding Boxes" to provide context for the visual results. Below is the image from the character segmentation stage.

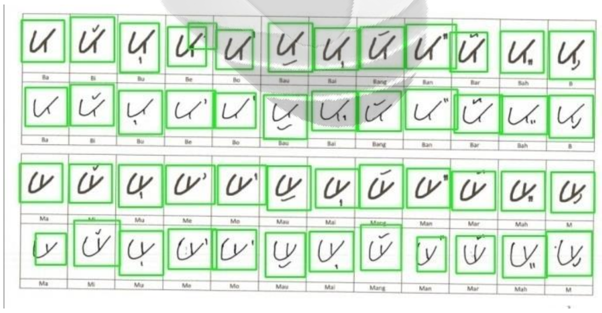


Figure 15. Image Result From The Character Segmentation Stage

Once the image has been successfully saved, the next steps involve cropping and clustering. Subsequently, the dataset created will be trained using a deep learning model, enabling it to classify images in the future.

V. CONCLUSION

A. CONCLUSION

Based on the research conducted, a total of 1,020 images were processed. These images were obtained from completed and scanned questionnaires using a scanning device. Subsequently, the images were processed during the image preprocessing stage to enhance their quality before undergoing image segmentation. This study employed the Bounding Box method to process the OganKomeringUlu (OKU) East script images obtained from the preprocessing stage prior to entering the image segmentation process. This method effectively assists in separating characters within the respondent's page images. By utilizing the Bounding Box method, the segmented characters in the images can be efficiently processed in the subsequent stages, namely cropping and clustering.

B. SUGGESTION

Here are some suggestions to improve the segmentation process: It is recommended to find additional code that allows for batch input of images into the process. Currently, images must be input one at a time, which is time-consuming when dealing with a large number of images. Implementing batch processing would significantly reduce the time required for image input.

AUTHORS CONTRIBUTION

M. Fikri: Conceptualization, Methodology, Validation, Investigation, Data Curation, Original Draft Preparation, Visualization;
Ilman Zuhri Yadi: Conceptualization, Methodology, Validation, Formal Analysis, Investigation, Data Curation, Original Draft Preparation, Review and Editing, Visualization, Supervision, Project Administration.
Yesi Novaria Kunang: Conceptualization, methodology, validation, formal analysis, investigation, data curation, original draft preparation, review and editing, visualization, supervision, and project administration.
Leon Andretti Abdillah: Conceptualization, methodology, validation, formal analysis, investigation, data curation, original draft preparation, review and editing, visualization, supervision, and project administration.

COPYRIGHT

REFERENCES

[1] P. A. Setiyawan, A. A. K. A. C. W, and I. P. A. Bayupati, "Balinese Alphabet Sebagai Aplikasi Media Pembelajaran Aksara Bali Berbasis Android Mobile Platform," *Merpati*, vol. 2, no. 2, pp. 226–237, 2014.
[2] R. Adipranata, J. Siwalankerto, and S. Telp, "Kombinasi Metode Morphological Gradient Dan Transformasi Watershed Pada Proses Segmentasi Citra Digital," *J. Inform. Petra*, no. 031, 2014.
[3] Z. Y. Malik Gumiwang, A. Haikal Nuqqy Zahhar, and H. Maulana, "Perbandingan Segmentasi Citra Menggunakan Algoritma K-Means Dan Algoritma Fuzzy C-Means," *J. Manaj. Inform. Jayakarta*, vol. 3, no. 1, pp. 21–26, 2023, [Online]. Available: http://journal.stmikjayakarta.ac.id/index.php/JMIJayakarta
[4] N. L. Kartika Sari, P. Hartoyo, and A. Ajrun, "Analisis Karakteristik Segmen Pada Citra Mamografi Dengan Menggunakan Metode Segmentasi Watershed," *J. Pembelajaran Fis.*, vol. 11, no. 2, p. 59, 2022, doi: 10.19184/jpf.v11i2.31643.
[5] F. Maedjaja and Efraim, "Sistem deteksi teks pada cover buku dengan pendekatan karakter teks," *Infact Ukrim*, vol. 6, no. 2, 2021.
[6] R. Riandini and D. Kuncoro, "Estimasi Panjang Antrean Kendaraan pada Persimpangan Jalan Raya dengan Sensor Kamera Menggunakan Metode Queue Length Estimation," *J. Comput. Eng. Network, Intell. Multimed.*, vol. 1, no. 1, pp. 14–20, 2023, doi: 10.59378/jcenim.v1i1.4.
[7] V. M. Sutama, I. R. Magdalena, and I. Wijayanto, "Identifikasi Objek Dominan Citra Digital Menggunakan Metode Markov Random Field (mrf)," *eProceedings Eng.*, vol. 5, no. 3, pp. 4859–4865, 2018, [Online]. Available: https://openlibrarypublications.telkomuniversity.ac.id/index.php/engineering/article/view/7839
[8] M. D. Hamanrora, Y. N. Kunang, I. Z. Yadi, and Mahmud, "Image segmentation of Komering script using bounding box," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 35, no. 3, pp. 1565–1578, 2024, doi: 10.11591/ijeecs.v35.i3.pp1565-1578.
[9] I. Boyke Nainggolan, I. Rita Magdalena, and R. Yunendah Nur Fu, "Matched Filter Dan Operasi Morfologi Untuk Estimasi Derajat Kebengkokan Tulang Matched Filters and Morphological Operations for Estimating Design of Bone Grass," vol. 5, no. 3, p. 5108, 2018.
[10] A. Septiarini, K. Kunci, and P. Proyeksi, "Segmentasi Karakter Menggunakan Profil Proyeksi," *J. Inform. Mulawarman Ed. Juli*, vol. 7, no. 2, pp. 66–69, 2012.

SURAT PERNYATAAN

Saya yang bertanda tangan di bawah ini:

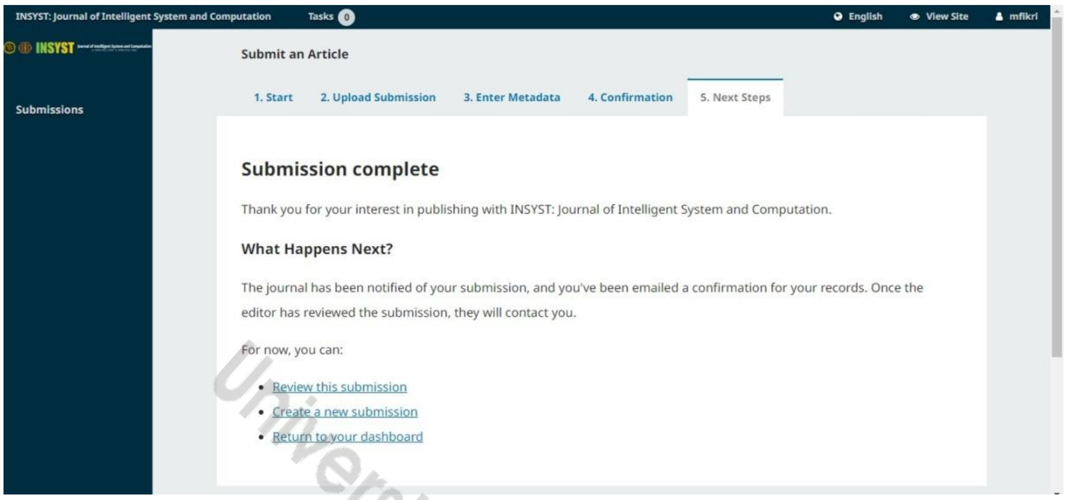
Nama	: M. Fikri
Nim	: 201401012
Program Studi	: Sistem Informasi
Fakultas	: Sains Teknologi
No. WA	: 083172653818
Nama Pembimbing	: Ilman Zuhri Yadi, M.M., M.Kom
Judul Artikel	: Application Of The Bounding Box Method For OKU Timur Character Image Segmentation

Menyatakan memang benar belum mendapatkan *Letter of Acceptance* (LoA) dan masih tahap *submit*/menunggu proses *review* dari pihak penerbit jurnal. Mengingat pendaftaran wisuda sedang berlangsung, untuk itu saya mohon dapat diizinkan mendaftar wisuda walaupun belum mendapatkan LoA, dengan konsekuensi tidak mendapatkan Transkrip Akademik saya. Saya secara sadar tidak akan menuntut Transkrip Akademik saya sebelum saya mendapatkan LoA dan mengumpulkan ke Pusat Pelayanan Mahasiswa (PPM).

Demikian surat pernyataan ini saya buat dengan sebenarnya untuk dipergunakan sebagaimana mestinya. Terima kasih.

Mengetahui,	Palembang, September 2024
Ketua Program Studi	Hormat saya,
[illegible]	10000 METERAI TEMPEL D0ALX35302353 [illegible]
Nita Rosa Damayanti., M.Kom., Ph.D.	M. Fikri

Lampiran:
Bukti submit artikel


INSYST: Journal of Intelligent System and Computation

Submit an Article

1. Start 2. Upload Submission 3. Enter Metadata 4. Confirmation 5. Next Steps

Submission complete

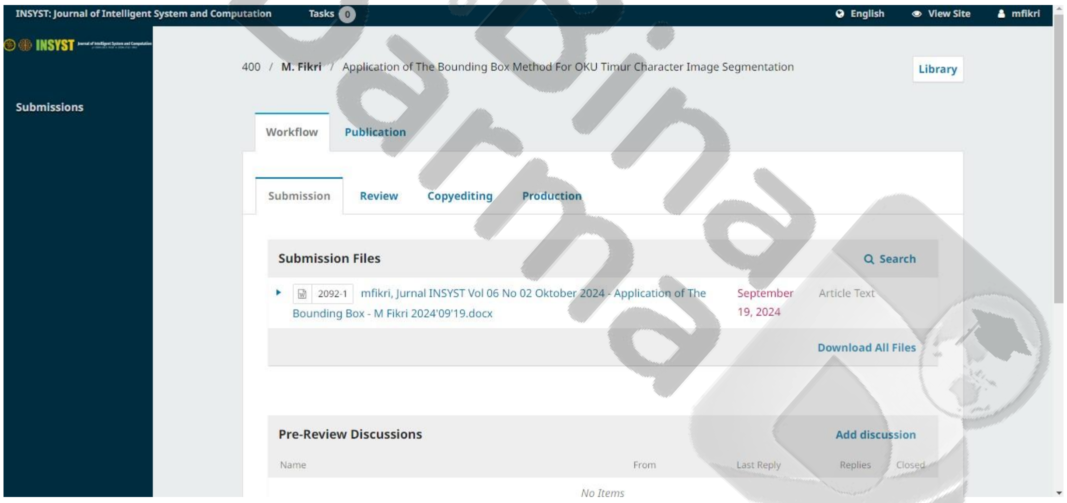
Thank you for your interest in publishing with INSYST: Journal of Intelligent System and Computation.

What Happens Next?

The journal has been notified of your submission, and you've been emailed a confirmation for your records. Once the editor has reviewed the submission, they will contact you.

For now, you can:

- Review this submission
- Create a new submission
- Return to your dashboard


INSYST: Journal of Intelligent System and Computation

400 / M. Fikri / Application of The Bounding Box Method For OKU Timur Character Image Segmentation

Submission Files

2092-1	mfikri, Jurnal INSYST Vol 06 No 02 Oktober 2024 - Application of The Bounding Box - M Fikri 2024'09'19.docx	September 19, 2024	Article Text

Pre-Review Discussions

Name	From	Last Reply	Replies	Closed
No Items				