

**CSE 4128 Image Processing and Computer Vision Laboratory**

# **Alphabet Detection**

By

**Doniel Tripura**

Roll: 1907121



**Department of Computer Science and Engineering**

**Khulna University of Engineering & Technology**

**Khulna 9203, Bangladesh**

**September, 2024**

# **Alphabet Detection**

By

**Doniel Tripura**

Roll: 1907121

**Submitted To:**

**Dr. Sk. Md. Masudul Ahsan**

Professor

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

**Dipannita Biswas**

Lecturer

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Khulna 9203, Bangladesh

September, 2024

## **Acknowledgement**

All praise is due to the Almighty God, whose blessings and mercy have guided me to successfully complete this coursework. I would like to express my sincere gratitude to my course teachers, Dr. Sk. Md. Masudul Ahsan & Dipannita Biswas for their continuous support and guidance throughout this project.

**Author**

# Contents

	<b>PAGE</b>
Alphabet Detection	i
Acknowledgement	ii
Contents	iii
List of Figures	iv
 <b>CHAPTER I Introduction</b>	 <b>1</b>
1.1 Introduction	1
1.2 Objectives	1
<b>CHAPTER II Methodology</b>	<b>2</b>
2.1 Image Segmentation	2
2.2 Template Preprocessing	4
2.3 Template Matching using Cross-Correlation	4
2.4 Alphabet Recognition	6
2.5 Accuracy Testing	6
<b>CHAPTER III Implementation, Results and Discussion</b>	<b>7</b>
3.1 Pseudocode of the Alphabet Detection System	7
3.2 Output and Intermediate Images	8
3.3 Result & Analysis	11
<b>CHAPTER IV Findings and Recommendations</b>	<b>13</b>
4.1 Findings	13
4.2 Recommendations	13
<b>CHAPTER V Conclusions</b>	<b>15</b>
5.1 Discussion	15
5.2 Conclusion	16
 <b>References</b>	 <b>17</b>

## List of Tables

<b>Table No.</b>	<b>Description</b>	<b>Page</b>
3.1	Contours with their corresponding images and recognized alphabets	10
3.2	Performance of the alphabet detection system across different datasets	11

## List of Figures

Figure No.	Description	Page
3.1	Original Input Image	8
3.2	Gaussian Blurred Image	8
3.3	Grayscale Image	8
3.4	Thresholded Image	8
3.5	Contour 1	9
3.6	Contour 2	9
3.7	Contour 3	9
3.8	Contour 4	9
3.9	Contour 5	9
3.10	Contour 6	9
3.11	Contour 7	9

# **CHAPTER I**

## **Introduction**

### **1.1 Introduction**

In the digital age, the ability to recognize and interpret text accurately is crucial across various applications, from automated document processing to interactive systems. Alphabet detection, particularly through image processing, represents a fundamental aspect of optical character recognition (OCR). This project focuses on detecting English alphabets (A-Z and a-z) using a template matching approach, a method widely recognized for its simplicity and effectiveness.

The core of this project involves using the Calibri body font as a template. By comparing user-provided data against this template, the system identifies and detects the corresponding alphabet. This process highlights the practical application of image processing techniques in real-world scenarios, particularly in situations where traditional OCR methods may be too complex or resource-intensive.

### **1.2 Objectives**

The objectives of this project are as follows:

1. To develop an efficient alphabet detection system using image processing techniques.
2. To implement template matching as the primary method for detecting English alphabets (A-Z and a-z).
3. To utilize the Calibri body font as a standard template for accurate detection.
4. To ensure that the system can accurately identify and differentiate between uppercase and lowercase letters. To explore the potential of template matching in simplifying character recognition tasks, particularly in constrained environments.

## CHAPTER II

### Methodology

#### 2.1 Image Segmentation

The first step in the process is to segment the input image into individual regions containing potential characters. This is achieved by converting the input image to grayscale and applying a binary threshold using Otsu's method to create a binary image:

$$\text{gray}(x, y) = \text{convertToGray}(R(x, y), G(x, y), B(x, y)) \quad (2.1)$$

$$\text{thresh}(x, y) = \begin{cases} 0 & \text{if } \text{gray}(x, y) < T \\ 255 & \text{if } \text{gray}(x, y) \geq T \end{cases} \quad (2.2)$$

Contours are then detected from the binary image to identify and isolate each segment that potentially represents a character.

Contours are curves or boundaries that connect all the continuous points along a boundary with the same color or intensity. In image processing, contours are used to identify and analyze the shape and location of objects within an image. They represent the outline or boundary of objects and are crucial for object detection and recognition tasks.

Contours are detected through the following process:

##### 2.1.1 Binary Image Creation

The input image is first converted into a binary image where the objects of interest are highlighted (usually in white) and the background is represented in black. This binary image simplifies the problem by reducing it to two colors.



### 2.1.2 Contour Detection

The 'findContours' function in OpenCV is used to detect contours in the binary image. This function retrieves the contours as a list of points, where each contour represents a curve outlining an object.

- **Contour Definition:** A contour is a list of points that forms a closed boundary of an object. The points along the contour can be used to approximate the shape of the object.

- **Contour Retrieval:** The 'findContours' function retrieves contours based on specified retrieval modes and approximation methods. The retrieved contours are used to isolate and identify individual objects within the binary image.

The detection process can be summarized as follows:

$$\text{contours} = \text{findContours}(\text{thresh}, \text{RETR\_EXTERNAL}, \text{CHAIN\_APPROX\_SIMPLE}) \quad (2.3)$$

- **RETR\_EXTERNAL:** This mode retrieves only the external contours of objects, ignoring any internal contours. It is useful for detecting the outer boundaries of objects.

- **CHAIN\_APPROX\_SIMPLE:** This method compresses horizontal, vertical, and diagonal segments of the contours and retains only their end points. It simplifies the contour by reducing the number of points stored, making it easier to process.

### 2.1.3 Bounding Contours

Each detected contour is bounded by a rectangle, known as the bounding box. This rectangle is defined by the top-left corner coordinates and the dimensions of the rectangle. The region of interest (ROI) for each detected contour is then extracted from the image using the bounding rectangle.

$$\text{ROI}(i) = \text{image}[y : y + h, x : x + w] \quad (2.4)$$

where  $(x, y)$  represents the top-left corner of the bounding rectangle, and  $(w, h)$  are the width and height of the rectangle.

The contour detection process is essential for isolating individual characters or objects in an image, enabling further analysis and recognition tasks.

## 2.2 Template Preprocessing

Templates of each alphabet are preprocessed to standardize their size and format. The templates are resized to a fixed size of  $100 \times 100$  pixels and converted to grayscale using OpenCV functions:

$$\text{template\_resized} = \text{resize}(\text{template}, (100, 100)) \quad (2.5)$$

## 2.3 Template Matching using Cross-Correlation

Template matching is performed using cross-correlation, a technique that measures the similarity between the input segment and the template.

$$r_{xy}(u, v) = \frac{\sum_{x=0}^{m-1} \sum_{y=0}^{n-1} [f(x, y) - \mu_f] [t(x - u, y - v) - \mu_t]}{\sqrt{\sum_{x=0}^{m-1} \sum_{y=0}^{n-1} [f(x, y) - \mu_f]^2 \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} [t(x - u, y - v) - \mu_t]^2}} \quad (2.6)$$

The process involves the following steps:

**1. Sliding Window:** The  $3 \times 3$  template is slid over the  $5 \times 5$  input image. For each position, a  $3 \times 3$  segment of the input image is compared to the template.

**2. Cross-Correlation Calculation:**

For each position  $(u, v)$ , compute the normalized cross-correlation coefficient using the following formula:

$$r_{xy}(u, v) = \frac{\sum_{x=0}^2 \sum_{y=0}^2 [f(x, y) - \mu_f] [t(x - u, y - v) - \mu_t]}{\sqrt{\sum_{x=0}^2 \sum_{y=0}^2 [f(x, y) - \mu_f]^2 \sum_{x=0}^2 \sum_{y=0}^2 [t(x - u, y - v) - \mu_t]^2}} \quad (2.7)$$

where:

- $f(x, y)$  is the intensity of the pixel at position  $(x, y)$  in the  $3 \times 3$  segment of the input image.
- $t(x - u, y - v)$  is the intensity of the pixel at position  $(x - u, y - v)$  in the  $3 \times 3$  template.
- $\mu_f$  and  $\mu_t$  are the mean intensities of the  $3 \times 3$  segment and the template, respectively.

**Explanation:** The numerator of the equation calculates the covariance between the segment  $f$  and the template  $t$ . It measures how much the values of the segment and template vary together. The denominator normalizes this value by the product of the standard deviations of the segment and template. The standard deviation is computed as follows:

$$\sigma_f = \sqrt{\frac{\sum_{x=0}^2 \sum_{y=0}^2 [f(x, y) - \mu_f]^2}{9}} \quad (2.8)$$

$$\sigma_t = \sqrt{\frac{\sum_{x=0}^2 \sum_{y=0}^2 [t(x - u, y - v) - \mu_t]^2}{9}} \quad (2.9)$$

Here,  $\sigma_f$  and  $\sigma_t$  represent the standard deviations of the  $3 \times 3$  segment and the template, respectively. The standard deviation measures the spread of pixel intensities around the mean intensity. Normalizing by the product of the standard deviations ensures that the cross-correlation coefficient reflects the relative similarity between the segment and the template, independent of their absolute intensity values.

**3. Best Match Identification:** The position  $(u, v)$  with the highest cross-correlation coefficient indicates the best match for the template in the input image.

**4. Bounding Box:** Draw a bounding box around the detected template position to highlight the identified object in the input image.

**Example of Cross-Correlation Calculation:**

Consider the following example where we have a  $5 \times 5$  image and a  $3 \times 3$  template. We extract a  $3 \times 3$  segment from the image for comparison:

$$\text{Full Image Matrix: } \begin{bmatrix} f_{00} & f_{01} & f_{02} & f_{03} & f_{04} \\ f_{10} & f_{11} & f_{12} & f_{13} & f_{14} \\ f_{20} & f_{21} & f_{22} & f_{23} & f_{24} \\ f_{30} & f_{31} & f_{32} & f_{33} & f_{34} \\ f_{40} & f_{41} & f_{42} & f_{43} & f_{44} \end{bmatrix}$$

$$\text{Extracted Segment (at position (1, 1))}: \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}$$

$$\text{Template Matrix: } \begin{bmatrix} t_{00} & t_{01} & t_{02} \\ t_{10} & t_{11} & t_{12} \\ t_{20} & t_{21} & t_{22} \end{bmatrix}$$

This approach is based on the principles of normalized cross-correlation and is widely used

in image processing tasks for template matching [1].

## 2.4 Alphabet Recognition

The alphabet corresponding to the highest cross-correlation coefficient is selected as the recognized character. If multiple templates achieve similar correlation values, the one with the highest overall correlation is selected:

$$\text{recognized\_alphabet} = \arg \max_k \left( \max_{u,v} \left( r_{xy}^k(u, v) \right) \right) \quad (2.10)$$

## 2.5 Accuracy Testing

To evaluate the system's performance, the accuracy is calculated by comparing the detected alphabets with the expected ones across a test dataset:

$$\text{Accuracy} = \left( \frac{\text{Number of Correct Detections}}{\text{Total Number of Test Images}} \right) \times 100\% \quad (2.11)$$

## CHAPTER III

### Implementation, Results and Discussions

#### 3.1 Pseudocode of the Alphabet Detection System

Alphabet\_Detection\_System(source, template\_dirs)

BEGIN

    InitializeSystemComponents()

    preprocessed\_templates = PreprocessTemplates(template\_dirs)

    WHILE system is active DO

        input\_image = AcquireImageFromSource(source)

        blurred\_image = ApplyGaussianBlur(input\_image)

        gray\_image = ConvertToGrayscale(blurred\_image)

        binary\_image = ApplyThresholding(gray\_image)

        segments, image\_with\_boxes, bounding\_boxes = SegmentImage(binary\_image)

        processed\_segments = [PreprocessImage(segment) FOR segment IN segments]

        recognized\_alphabets = []

        match\_coords = []

        FOR each processed\_segment IN processed\_segments DO

            max\_match\_val = -

            recognized\_alphabet = None

            best\_match\_coord = None

            FOR each template IN preprocessed\_templates DO

                match\_val = CompareImages(processed\_segment, template)

                IF match\_val > max\_match\_val THEN

                    max\_match\_val = match\_val

                    recognized\_alphabet = TemplateToAlphabet(template)

                    best\_match\_coord = BoundingBoxForSegment(segment)

                END IF

```

        END FOR
        recognized_alphabets.append((recognized_alphabet, max_match_val))
        match_coords.append(best_match_coord)
    END FOR
    VisualizeResults(image_with_boxes, recognized_alphabets, match_coords)
    OptimizeSystem()
    IF user_requests_exit() THEN
        BREAK
    END IF
END WHILE
FinalizeSystemComponents()
END

```

## 3.2 Output and Intermediate Images

The following figures illustrate the output generated by the alphabet detection system, showcasing both the final detected alphabets and the intermediate steps in the image processing pipeline. Each figure demonstrates a critical stage in preparing the input image for accurate alphabet detection:



Figure 3.1: Original Input Image



Figure 3.2: Gaussian Blurred Image



Figure 3.3: Grayscale Image



Figure 3.4: Thresholded Image

The figures above illustrate the processing steps involved in preparing the input image for

alphabet detection. The application of Gaussian blur helps to reduce noise and smooth the image, while the grayscale conversion simplifies the color information, making it easier to analyze. The thresholding step then converts the image into a binary format, highlighting the contours of the alphabets, which are crucial for accurate segmentation.



Figure 3.5: Contour 1



Figure 3.6: Contour 2



Figure 3.7: Contour 3



Figure 3.8: Contour 4



Figure 3.9: Contour 5



Figure 3.10: Contour 6



Figure 3.11: Contour 7

The figures depicting individual contours illustrate the segmented regions after processing the binary image. Each contour represents a potential alphabet detected in the image. The contours are identified and labeled to facilitate the recognition process, as shown in the subsequent table.

The table above presents the details of each contour detected in the input image. Each row corresponds to a specific contour, showing the contour number, the cropped image of the contour, and the recognized alphabet. This table summarizes the results of the alphabet detection process, providing a clear view of how each detected contour is classified into an alphabet. The images in the "Image" column are extracted directly from the processed input image, and the "Alphabet" column indicates the result of the template matching for each contour.



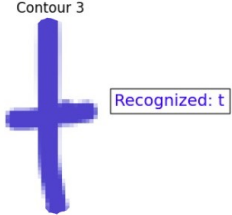


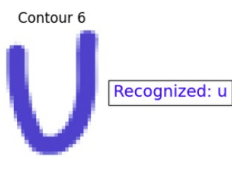
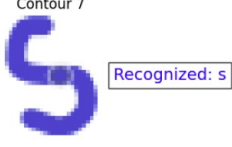
Contour No	Image	Alphabet
1	<p>Contour 1</p> 	O
2	<p>Contour 2</p> 	c
3	<p>Contour 3</p> 	t
4	<p>Contour 4</p> 	o
5	<p>Contour 5</p> 	p
6	<p>Contour 6</p> 	u
7	<p>Contour 7</p> 	s

Table 3.1: Contours with their corresponding images and recognized alphabets



This comprehensive visualization of the intermediate steps and detected contours highlights the effectiveness of the image processing pipeline in identifying and classifying the alphabets accurately. The use of Gaussian blur, grayscale conversion, and thresholding prepares the image for contour detection, which is crucial for the final recognition of the alphabets.

The next section will discuss the performance evaluation of the alphabet detection system, including accuracy metrics and potential areas for improvement.

### 3.3 Result & Analysis

The performance of the alphabet detection system was evaluated using different datasets. The following table summarizes the dataset types, their sizes, and the corresponding accuracy achieved by the system:

Dataset Type	Dataset Size	Accuracy
Calibri Body	26 images	100%
Times New Roman	26 images	48%
Hand-Written	26 images	76.92%

Table 3.2: Performance of the alphabet detection system across different datasets

The table above provides a detailed view of how the alphabet detection system performed with different datasets. The accuracy percentages reflect the system's ability to correctly identify alphabets from the test images.

- **Calibri Body:** The system achieved a perfect accuracy of 100% with this dataset. This suggests that the Calibri Body font was well-defined and easily recognizable by the detection algorithm, indicating optimal performance with clean and consistent data.

- **Times New Roman:** The accuracy dropped significantly to 48% with the Times New Roman dataset. This lower accuracy could be attributed to the variability in character shapes and styles within this font, which may have led to challenges in accurate detection.

- **Hand-Written:** The accuracy was 76.92% for the hand-written dataset. This result indicates that the system performed reasonably well with hand-written text, although there may have been variability in writing styles or inconsistencies in character formation.

These results underscore the importance of dataset characteristics on the performance of the alphabet detection system. While the system performed exceptionally well with the Calibri Body dataset, it faced challenges with other types of datasets due to differences in font styles

and image quality. Future work could focus on improving the system's robustness to handle a broader range of fonts and handwritten texts by incorporating more diverse training data and refining preprocessing techniques.

## CHAPTER IV

### Findings and Recommendations

#### 4.1 Findings

The implementation of the alphabet detection system, which involves contour detection, revealed several key findings:

1. **Multiple Contours Issue:** The system often detected multiple contours for the same character, particularly in cases where the characters were close together or had overlapping features. For instance, contours labeled as 'i' and 'j' were sometimes misinterpreted as separate entities, leading to errors in character recognition.
2. **Contour Detection Challenges:** The current contour detection approach struggled with varying character sizes and spacing, which resulted in inaccuracies. The algorithm sometimes produced contours that were not representative of actual characters, leading to unreliable results.
3. **Effectiveness of Preprocessing:** While preprocessing steps like Gaussian blur and thresholding were effective to some extent, they were not sufficient for handling all types of images. Variations in font styles and backgrounds impacted the consistency of contour detection.
4. **Impact of Dataset Characteristics:** The performance of the system varied significantly depending on the dataset used. Datasets with more complex backgrounds or diverse fonts posed additional challenges, affecting the overall accuracy of the detection system.

#### 4.2 Recommendations

To address the issues identified and enhance the performance of the alphabet detection system, the following recommendations are proposed:

1. **Alternative Contour Detection Methods:** Investigate alternative methods for contour detection, such as the Canny edge detector or the Hough transform, which may be more

effective in handling overlapping contours and complex backgrounds. Additionally, incorporating techniques like contour filtering based on contour area or shape could help in reducing false positives.

2. **Advanced Preprocessing Techniques:** Implement advanced preprocessing techniques to improve image clarity before contour detection. Techniques such as bilateral filtering, adaptive histogram equalization, or dynamic thresholding can help in reducing noise and enhancing the features of characters.

3. **Character Segmentation Enhancement:** Explore machine learning-based character segmentation methods, such as convolutional neural networks (CNNs) or region-based approaches, which could provide better accuracy and robustness compared to traditional contour-based methods.

4. **Dataset Enrichment:** Expand and diversify the dataset to include a wider range of font styles, sizes, and backgrounds. This will help the system generalize better and improve performance across various image conditions.

5. **System Optimization and Testing:** Continuously optimize the detection algorithm and conduct rigorous testing with diverse datasets. Implementing real-time feedback mechanisms and user testing will help in refining the system and addressing any emerging issues.

6. **Integration of Post-Processing Techniques:** Consider incorporating post-processing techniques such as contextual analysis or character recognition algorithms to refine and validate the detected contours, ensuring accurate character identification.

By adopting these recommendations, the alphabet detection system can be significantly improved, resulting in enhanced accuracy, reliability, and overall performance. Future work should focus on implementing these changes and evaluating their effectiveness in real-world scenarios.

## CHAPTER V

### Conclusions

#### 5.1 Discussion

The image processing pipeline designed for alphabet detection has demonstrated its effectiveness in preprocessing and analyzing input images. The pipeline employs Gaussian blur, grayscale conversion, and thresholding to prepare the image for contour detection and subsequent alphabet recognition. Each step in this process contributes to the accuracy and reliability of the system.

- **Gaussian Blur:** Applying Gaussian blur to the input image helps in reducing noise and smoothing the image, which is crucial for the subsequent processing steps. By blurring the image, small variations and high-frequency noise are minimized, ensuring that the contours detected are more robust and less affected by noise.
- **Grayscale Conversion:** The conversion of the blurred image to grayscale simplifies the image by removing color information. This reduction in complexity allows for more focused processing on structural details necessary for contour detection.
- **Thresholding:** Thresholding converts the grayscale image into a binary format, distinguishing the foreground (text) from the background. This step is vital for segmenting and identifying contours, making the recognition of individual alphabet characters more straightforward.
- **Contour Detection:** Detecting contours in the thresholded image isolates the regions of interest, and bounding boxes are drawn around these contours to visualize the detected segments. This segmentation is essential for isolating alphabet characters, which are then compared against preprocessed template images for recognition.
- **Template Matching:** Comparing the preprocessed segments against template images using cross-correlation allows for accurate identification of alphabet characters. The

system's performance in matching segments with templates demonstrates the effectiveness of the recognition approach.

The results presented in the figures and table highlight the successful implementation of these methods. However, challenges such as variations in text size, font, and orientation affect the system's performance. Future work could explore integrating advanced techniques, such as machine learning algorithms, to enhance the robustness and adaptability of the detection system.

## 5.2 Conclusion

In conclusion, the alphabet detection system effectively integrates several image processing techniques to achieve accurate recognition of letters from input images. The methodology, which includes Gaussian blur, grayscale conversion, thresholding, and contour detection, provides a robust foundation for text detection applications.

Key findings from this implementation include:

- **Effective Preprocessing:** The preprocessing steps significantly enhance the quality of the image for contour detection, leading to more accurate results.
- **Accurate Contour Detection:** The system successfully isolates and identifies contours representing individual alphabets, which is crucial for precise recognition.
- **Robust Recognition:** The template matching technique effectively classifies detected segments, demonstrating the effectiveness of the recognition approach.

Overall, the system proves to be effective for alphabet detection in controlled environments. Future enhancements could include the integration of advanced machine learning models to address challenges posed by varying text styles and image conditions. Such improvements would increase the system's adaptability and suitability for a wider range of applications. The results and methodologies outlined in this report contribute to the advancement of image processing techniques for text recognition and provide a foundation for further research and development in this field.

## References

- [1] Mehpara Saghir, Zeenat Bibi, Saba Bashir, and Farhan Hassan Khan. Churn prediction using neural network based individual and ensemble models. In *2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pages 634–639, 2019.