IMAGE PROCESSING GROUP PROJECT

Project Name: Skew correction of Text Image

Group

Name	Batch	Roll Number
Praval Pattam	CS-04	B220057CS
Pranav Sai Sarvepalli	CS-04	B220055CS
Potluri Theenesh	CS-04	B221121CS

Proposed Methodology

Pre-processing

1. Converting Image to Grayscale

Purpose: Simplifies the image by reducing it to shades of gray, which eliminates the complexity of color information.

Reason: In the context of the Hough Line Transform, we are interested in the structural elements of the image (i.e., edges and lines) rather than color. Converting to grayscale is a crucial step because it reduces computational complexity and ensures that subsequent edge detection algorithms operate effectively.

2. Blur Image

Purpose: Smooths the image by averaging pixel values, which helps in reducing noise and detail.

Reason: Blurring is essential in preprocessing to minimize the effect of noise, which can lead to false edges in edge detection. By applying a blur filter, we can ensure that the edges detected are more accurate and represent significant structural features, rather than random noise.

3. Threshold Image

Purpose: Converts the image to a binary format, distinguishing objects from the background.

Reason: Thresholding helps in isolating the regions of interest (potential line segments) from the rest of the image. By converting to binary, we enhance the contrast between the objects and the background, making it easier for the Hough Line Transform to identify and focus on the prominent lines in the image.

4. Erode Image

Purpose: Removes small noise by shrinking the white regions in a binary image.

Reason: Erosion is used to eliminate tiny, irrelevant details that might have been captured during the thresholding process. This step refines the edges by stripping away small noise elements, ensuring that what remains are the most significant structural components needed for line detection.

5. Dilate Image

Purpose: Expands the white regions in a binary image, filling in small holes and gaps.

Reason: Dilation is often used after erosion to restore the size of objects that might have been slightly reduced during erosion. It helps in connecting broken lines and filling small gaps, making the lines more continuous and easier to detect using the Hough Line Transform.

Tools/Techniques

Probabilistic Hough Line Transform

The Probabilistic Hough Line Transform (PHT) is an optimized version of the standard Hough Line Transform used to detect lines in an image. Unlike the standard method, which considers all edge points, PHT randomly selects a subset of edge points to reduce computational complexity. This makes it more efficient, especially for large images or real-time applications.

OpenCV

We implement the following preprocessing steps using OpenCV's inbuilt functions

1. Converting Image to Grayscale

```
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
```

2. Blur Image

```
blur = cv2.GaussianBlur(gray, (5, 5), 0)
```

3. Threshold Image

```
_, threshed = cv2.threshold(blur,0,255,cv2.THRESH_BINARY_INV+
cv2.THRESH_OTSU)
```

4. Erode Image

```
eroded = cv2.erode(threshed,(3,3),1)
```

5. Dilate Image

```
dilate = cv2.dilate(eroded, (35, 35), iterations=3)
```

We find the Hough lines in the image by

```
lines = cv2.HoughLinesP(dilate,1,np.pi/180,200,None,150,10)
```

Existing Methods

Using OpenCV, find the HoughLinesP in the image using the inbuilt functions then process the lines detected in an image to identify and work with horizontal lines.

Initialize Lists:
horizontal_lines: To store detected horizontal lines.
angles: To store the angles of these lines.

Iterate Over Detected Lines:

For each line, extract its endpoints (x1, y1) and (x2, y2).
Draw the line on img_copy.

Calculate Differences:
Calculate the differences in x (diff_x) and y (diff_y).

Identify Horizontal Lines:
Check if the line is nearly horizontal and add such lines to horizontal_lines.

Calculate Angles:
Calculate the slope and angle of the line.
Store the angle in angles.

Display All Lines:
Show the image with all detected lines (img_copy).

Display Only Horizontal Lines:
Draw and show only the horizontal lines (img_copy_1).

Calculating Average Rotation Angle:
Calculate the average angle of all horizontal lines.

Rotate Image:
Rotate the image by rotation_angle.

Create a Rotation Matrix:
Generate a 2D rotation matrix for rotating the image around the center by rotation_angle degrees.

Rotate Original Image to Show Transformation:
Apply the affine transformation (rotation) using the rotation matrix.

Design and Coding

Preprocessing

```
# Read image from directory
image = cv2.imread('images\houghtransform.jpg')
# image = cv2.imread('images\houghtransform_2.jpg')
img_copy = image.copy()
img_copy_1 = image.copy()
height,width = image.shape[:2]
center = (width//2, height//2)
original_image = image.copy()
# converting image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
# blur image
blur = cv2.GaussianBlur(gray, (5, 5), 0)
# threshold image
_, threshed = cv2.threshold(blur,0,255,cv2.THRESH_BINARY_INV+
cv2.THRESH_OTSU)
# erode image
eroded = cv2.erode(threshed,(3,3),1)
# dilate image
dilate = cv2.dilate(eroded, (35, 35), iterations=3)
```

Finding the Hough Lines

```
lines = cv2.HoughLinesP(dilate,1,np.pi/180,200,None,150,10)
```

Processing the Hough Lines

```python
if lines is not None:
    horizontal_lines = []
    angles = []
    for i,line  in enumerate(lines):
        x1 = line[0][0]
        y1 = line[0][1]
        x2 = line[0][2]
        y2 = line[0][3]
        cv2.line(img_copy, (x1,y1), (x2,y2), (0, 0, 255), 1, cv2.LINE_AA)
        diff_x = x2-x1
        diff_y = y2-y1
        if abs(diff_y) < 30 and abs(diff_x) != 0:
            horizontal_lines.append((x1, y1, x2, y2))
            try:
                slope = diff_y / diff_x
                angle = math.degrees(math.atan(slope))
                print(angle)
                angles.append(angle)
            except Exception as e:
                print(e)
                continue

    cv2.imshow('All Extracted Lines', img_copy)
    for line in horizontal_lines:
        # print(line)
        cv2.line(img_copy_1, (line[0],line[1]), (line[2],line[3]), (0,0,255), 1, cv2.LINE_AA)
        cv2.imshow('Formated lines',img_copy_1)
    rotation_angle = sum(angles) / len(angles)
```

Finding the Rotation Matrix and applying the Affine transformation

```python
rotation_matrix = cv2.getRotationMatrix2D(center, rotation_angle, 1)
# rotate orignal image to show transformation
rotated_image = cv2.warpAffine(original_image, rotation_matrix, (width, height), borderValue=(255, 255, 255))
```