Your Flowchart Secretary: Hand-Written Flowchart Converter

Team bh454nd
IIIT Hyderabad

December 4, 2024

Contents

1 Introduction

Flowcharts serve as crucial tools for organizing and visualizing processes and ideas, especially in early development stages. However, digitizing hand-drawn flowcharts for formal use can be labor-intensive. This project leverages image processing techniques to automate this conversion process.

2 Assumptions

1. **Closed Graph Components:** Shapes in the flowchart must have clearly defined boundaries, allowing detection as closed regions. This simplifies contour detection and avoids misclassifications.

2. **Straight Arrows:** Arrows are assumed to be straight, ensuring easy separation from shapes. Intersection-free arrows help maintain consistent directional flow extraction.

3. **Recognizable Shapes:** The system currently supports basic flowchart elements: rectangles, diamonds, circles, and arrows.

3 Pipeline

The pipeline consists of:

1. **Input and Preprocessing:** Cleans the input image for further processing.

2. **Component Extraction:** Identifies and separates individual flowchart elements.

3. **Digital Reconstruction:** Reconstructs the digital version using extracted components.

4 Preprocessing

4.1 Image Preparation and Binarization

Image preparation involves reducing the input complexity and isolating essential components.

Theory: Grayscale conversion simplifies processing by reducing three color channels to one. Gaussian filtering removes noise by averaging pixel values in a localized region. Adaptive thresholding dynamically identifies foreground pixels.

OpenCV Functions:

- `cv2.cvtColor`: Converts image to grayscale.

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

- `cv2.GaussianBlur`: Applies Gaussian blur to reduce noise.

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

- `cv2.adaptiveThreshold`: Performs adaptive thresholding.

```
binary = cv2.adaptiveThreshold(blurred, 255,
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
    cv2.THRESH_BINARY_INV, 11, 2)
```

4.2 Denoising and Small Region Removal

Denoising ensures small artifacts are eliminated, and key components remain intact. Contour-based filtering removes small noise regions by identifying and analyzing their size.

OpenCV Functions:

- `cv2.findContours`: Finds contours in the image.
- `cv2.contourArea`: Calculates area to filter noise.

```
contours, _ = cv2.findContours(binary,
                                cv2.RETR_EXTERNAL,
                                cv2.CHAIN_APPROX_SIMPLE)
for cnt in contours:
    if cv2.contourArea(cnt) < threshold:
        cv2.drawContours(binary, [cnt], -1, 0, -1)
```

4.3 Hough Transform and Rotation

Theory: The Hough Transform detects straight lines by identifying parameterized line equations. The primary angle frequency is then used to orient the flowchart.

OpenCV Functions:

- `cv2.HoughLines`: Detects lines in the image.

```
lines = cv2.HoughLines(edges, 1, np.pi / 180, 100)
```

- `cv2.getRotationMatrix2D` and `cv2.warpAffine`: Rotates the image.

```
M = cv2.getRotationMatrix2D(center, angle, 1)
rotated = cv2.warpAffine(image, M, (w, h))
```

5 Component Extraction

5.1 Separating Arrows and Shapes

Theory: Morphological operations such as erosion and dilation help isolate different flowchart elements.

OpenCV Functions:

- `cv2.erode`: Removes arrows while preserving shapes.
- `cv2.dilate`: Restores shape dimensions.

```
eroded = cv2.erode(binary, kernel, iterations=1)
dilated = cv2.dilate(eroded, kernel, iterations=1)
```

5.2 Identifying Shapes

Theory: Circles are identified by analyzing contour circularity. Rectangles and diamonds are distinguished based on bounding box ratios.

OpenCV Functions:

- `cv2.HoughCircles`: Detects circles.

```
circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1, 20)
```

- `cv2.boundingRect`: Finds bounding boxes for other shapes.

```
x, y, w, h = cv2.boundingRect(cnt)
```

6 Digital Reconstruction

Using coordinates and dimensions, the flowchart is digitally reconstructed by drawing the components.

OpenCV Functions:

- `cv2.rectangle`, `cv2.circle`, `cv2.arrowedLine`: Draw rectangles, circles, and arrows.

```
cv2.rectangle(image, (x, y), (x+w, y+h), color, thickness)
cv2.circle(image, (center_x, center_y), radius, color, thickness)
cv2.arrowedLine(image, start, end, color, thickness)
```

7 Future Work

- Real-time flowchart conversion through a mobile app.
- Incorporating OCR to detect and digitize handwritten text within shapes.
- Generalizing the pipeline to accept arbitrary shapes using geometric properties.