Contour Detection in Image Processing :

Thresholding Operation: Given a grayscale image $I(x, y)$, where (x, y) are the pixel coordinates:

Choose a threshold value T. Classify each pixel:

$$B(x, y) = \begin{cases} 0 & \text{if } I(x, y) < T \text{ (background)} \\ 255 & \text{if } I(x, y) \geq T \text{ (foreground)} \end{cases}$$

Otsu's Method: Otsu's method is a technique to automatically calculate an optimal threshold T by minimizing the intra-class variance or maximizing inter-class variance between foreground and background pixels.

Steps Involved:

1. Compute the histogram $H(i)$ of the grayscale image I.
2. Normalize the histogram to obtain the probability distribution $p(i) = \frac{H(i)}{N}$, where N is the total number of pixels.
3. Compute the cumulative distribution $P(i) = \sum_{k=0}^{i} p(k)$ and the cumulative mean $\mu(i) = \sum_{k=0}^{i} k \cdot p(k)$.
4. Calculate the global mean intensity $\mu_T = \mu(N - 1)$.
5. Maximize the between-class variance

   $$\sigma_B^2(T) = P(T) \cdot (1 - P(T)) \cdot (\mu_T \cdot P(T) - \mu(T))^2$$

   to find the optimal threshold T.

Result: After applying the threshold T, the grayscale image I is transformed into a binary image B, where each pixel is assigned a value (0 or 255) based on the thresholding condition.

Finding Contours

Contour Detection: Given a binary image $B(x, y)$, where each pixel is either 0 (background) or 255 (foreground):

Algorithm:

- Connected Component Analysis: Traverse the binary image B to find connected sets of foreground pixels.
- Boundary Extraction: Extract the boundary of each connected component as a sequence of points.
- Representation: Store contours as lists of coordinates or as polygonal approximations (depending on the contour approximation method used).

Drawing Contours

Contour Rendering: For each contour:

- Retrieve the sequence of points that describe the contour.
- Connect these points with straight lines or curves based on the rendering settings.
- Draw these lines on an image to visually represent the contours.

Parameters:

- Color: Specify the color (in RGB or BGR format) to draw the contours.
- Thickness: Set the thickness of the contour lines.
- Line Type: Determine the line style (e.g., anti-aliasing) for smoother contour visualization.

Face Detection Using Haar Cascades: Mathematical Equations :

1. Integral Image Calculation

The integral image $I(x, y)$ is computed as:

$$I(x, y) = \sum_{x' \leq x, y' \leq y} \text{image}(x', y')$$

To compute efficiently, the integral image can be recursively defined as:

$$I(x, y) = I(x - 1, y) + I(x, y - 1) - I(x - 1, y - 1) + \text{image}(x, y)$$

2. Haar-like Features

Haar-like features are defined over rectangular regions and are computed using the integral image $I(x, y)$. The value of a Haar-like feature at position (x, y) with size (w, h) and type A (where type can be edge, line, or center-surround) is computed as:

$$\text{Haar}(x, y, w, h, A) = \sum_{(x', y') \in A} \text{image}(x', y')$$

Here, A represents the area covered by the Haar-like feature.

3. Cascade Classifier

A cascade classifier consists of multiple stages S, each containing several weak classifiers C_i. Each weak classifier C_i operates on a specific Haar-like feature and evaluates whether the feature pattern resembles a face.

The output of the cascade classifier CC for a window at position (x, y) with size (w, h) is computed as:

$$CC(x, y, w, h) = \begin{cases} 1 & \text{if } \sum_i C_i(x, y, w, h) > T_i \text{ for all stages } S_i \\ 0 & \text{otherwise} \end{cases}$$

Here, T_i represents the threshold for each stage S_i.

4. Detection Algorithm

The detection algorithm involves sliding a window across the image and applying the cascade classifier at multiple scales. The algorithm computes the response $R(x, y, w, h)$ for each window:

$$R(x, y, w, h) = CC(x, y, w, h)$$

Eye Detection Using Haar Cascades: Mathematical Equations :

1. ROI (Region of Interest) Extraction

Once a face is detected, a region of interest (ROI) around the detected face area is defined. For simplicity, let's denote the coordinates of the detected face as (x_f, y_f) for the top-left corner, and (w_f, h_f) for the width and height of the face bounding box.

The ROI coordinates for the eyes are often derived by considering a fraction or specific offsets from the face boundaries.

2. Haar-like Features for Eyes

Haar-like features specific to eyes are computed within the ROI defined by the detected face. These features capture patterns characteristic of eyes, such as edges around the eye sockets, corners of the eyes, and the iris region.

Let's denote the coordinates of the detected eye region within the face ROI as (x_e, y_e) for the top-left corner, and (w_e, h_e) for the width and height of the eye bounding box.

The value of a Haar-like feature for eyes at position (x, y) with size (w, h) and type A is computed similarly to face detection but specifically for eye features within the ROI.

$$\text{Haar\_Eye}(x, y, w, h, A) = \sum_{(x', y') \in A} \text{image}(x', y')$$

Here, A represents the area covered by the Haar-like feature specific to eyes.

3. Cascade Classifier Application

The cascade classifier for eye detection is applied similarly within the ROI defined by the detected face. Each stage of the classifier evaluates whether the pattern of Haar-like features resembles eyes.

The output of the cascade classifier CC_{eye} for the eye region at position (x_e, y_e) with size (w_e, h_e) is computed as:

$$CC_{eye}(x_e, y_e, w_e, h_e) = \begin{cases} 1 & \text{if } \sum_i C_i(x_e, y_e, w_e, h_e) > T_i \text{ for all stages } S_i \\ 0 & \text{otherwise} \end{cases}$$

Here, T_i represents the threshold for each stage S_i of the cascade classifier for eyes.