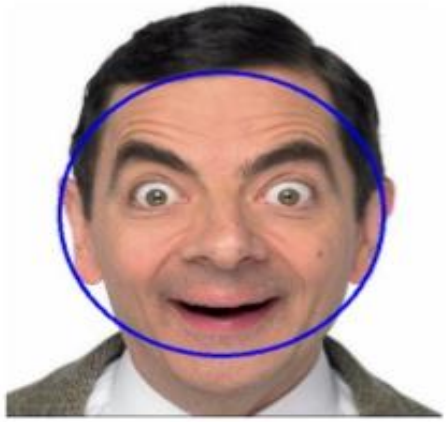
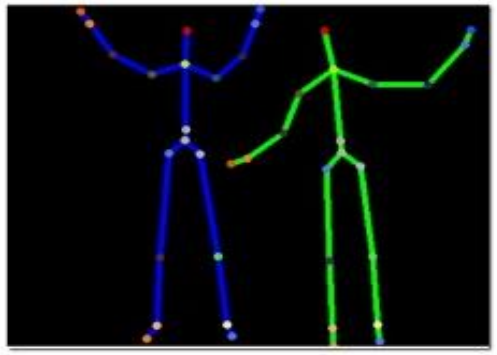
OpenCV Workshop





Session 4: Extracting Lines, Contours, and Components

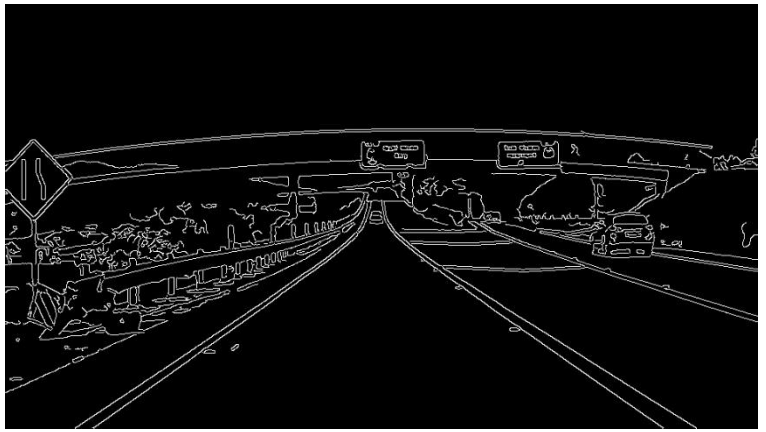
Elham Shabaninia

Detecting lines in images with the Hough transform

Hough transform

- The Hough transform is a classic algorithm that was initially developed to detect lines in images and, as we will see, it can also be extended to detect other simple image structures such as circles.
- In image processing, a sub problem often arises of detecting simple shapes, such as straight lines, circles or ellipses.
- In many cases an edge detector can be used as a pre-processing stage to obtain image points or image pixels that are on the desired curve in the image space.

Video frame with detected lane lines



Canny edge detection

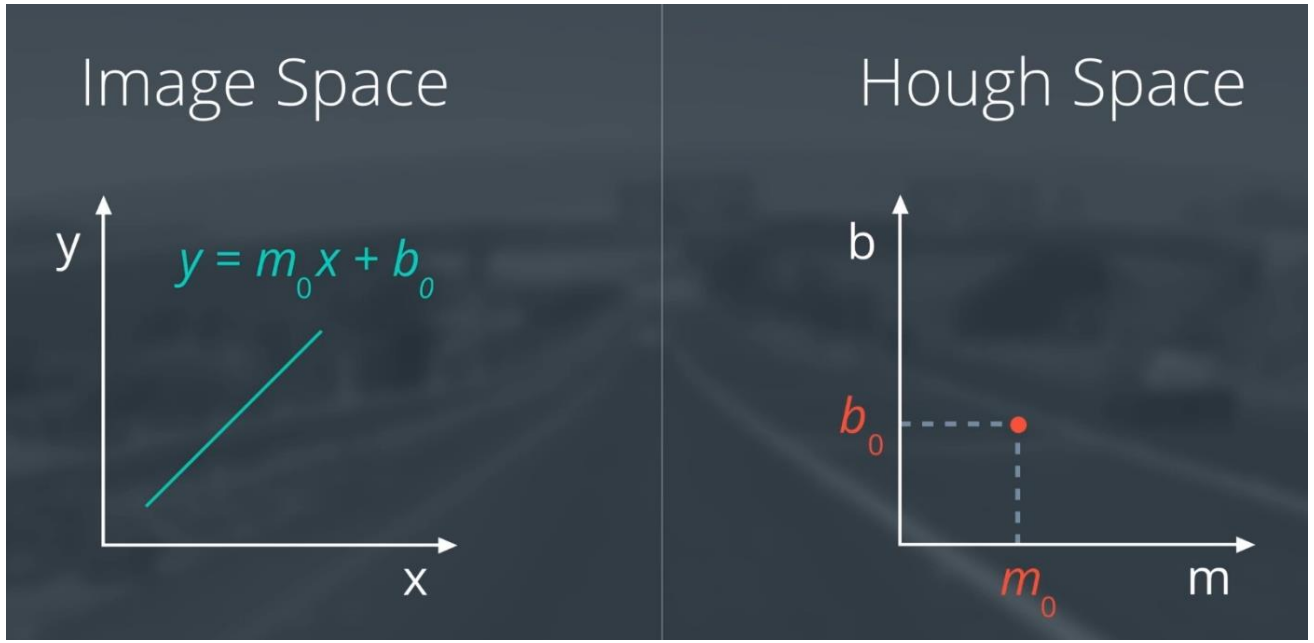
Hough transform

- Due to imperfections in either the image data or the edge detector, however, there may be missing points or pixels on the desired curves as well as spatial deviations between the ideal line/circle/ellipse and the noisy edge points as they are obtained from the edge detector.
- So it is often non-trivial to group the extracted edge features to an appropriate set of lines, circles or ellipses.
- The purpose of the Hough transform is groupings of edge points into object candidates by performing an explicit voting procedure over a set of parameterized image objects.

Hough transform

- Now let's see how Hough Transform works for lines.

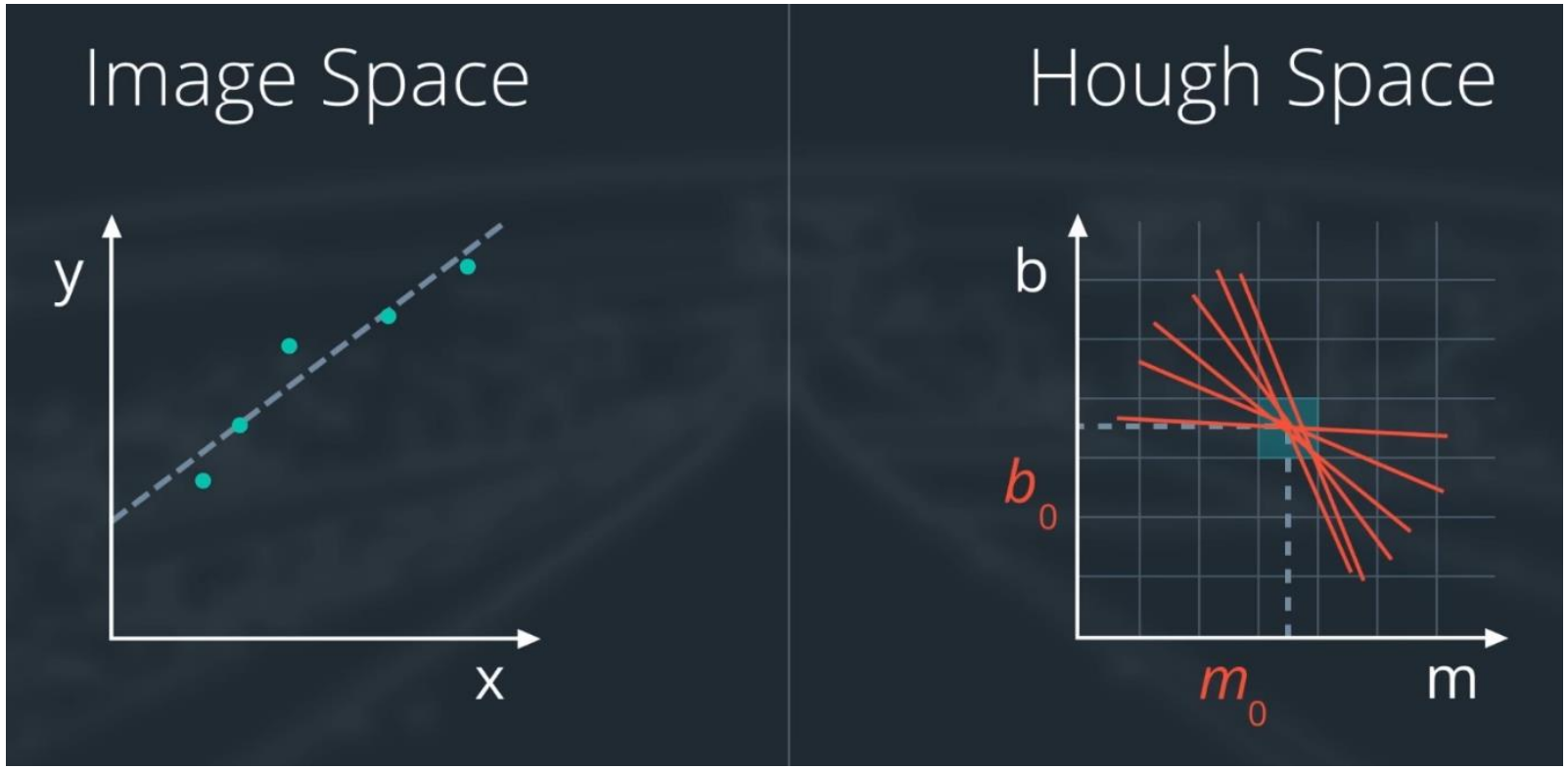
A **line** in Image Space is a **point** in Hough Space.



Any line can be represented in these two terms (m, b). So first it creates a 2D array or accumulator (to hold values of two parameters) and it is set to 0 initially.

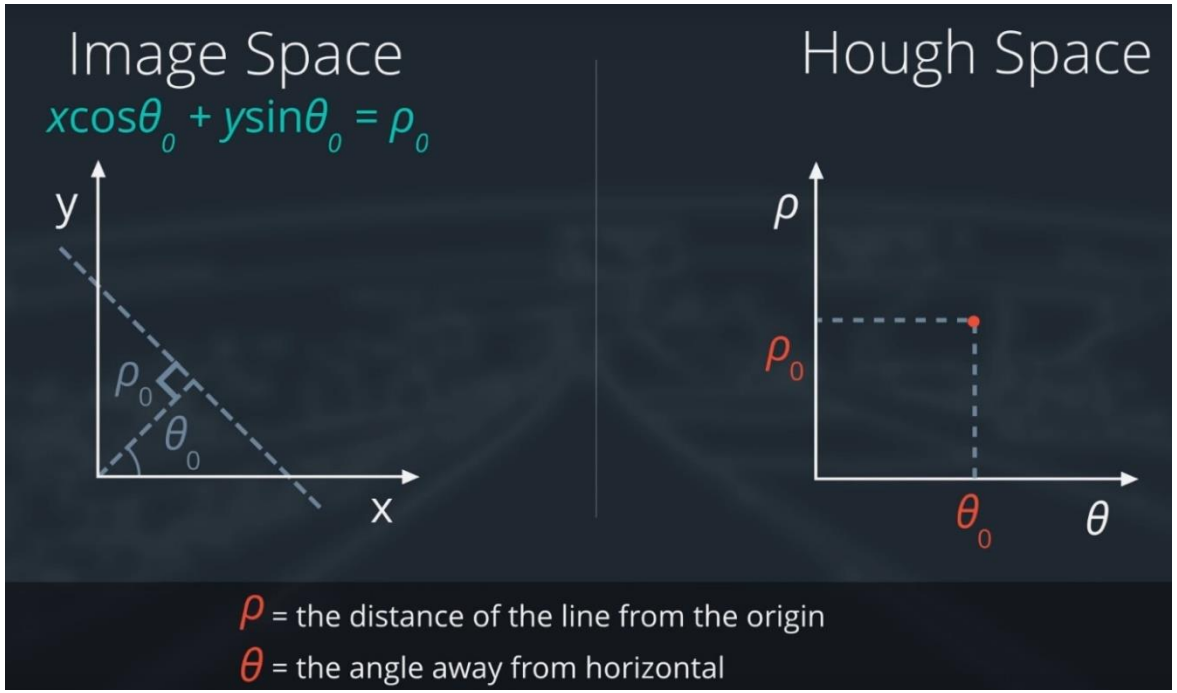
Hough transform

Then we divide the Hough Space into grid, and define the intersecting lines as all lines passing through a given grid cell.



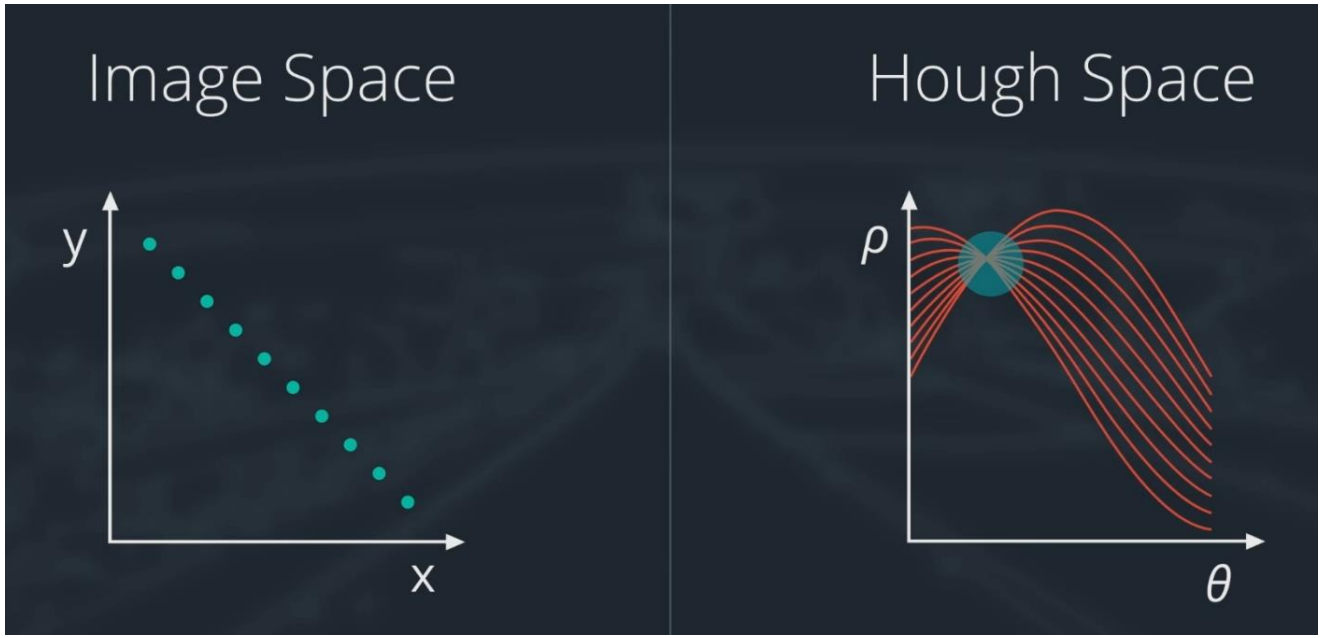
Hough transform

Since m, b may have infinite value in y = mx+b representation, the accumulator size should be consider as infinite.
To solve this problem we define a line in polar coordinates as shown in the following fiugre.



Hough transform

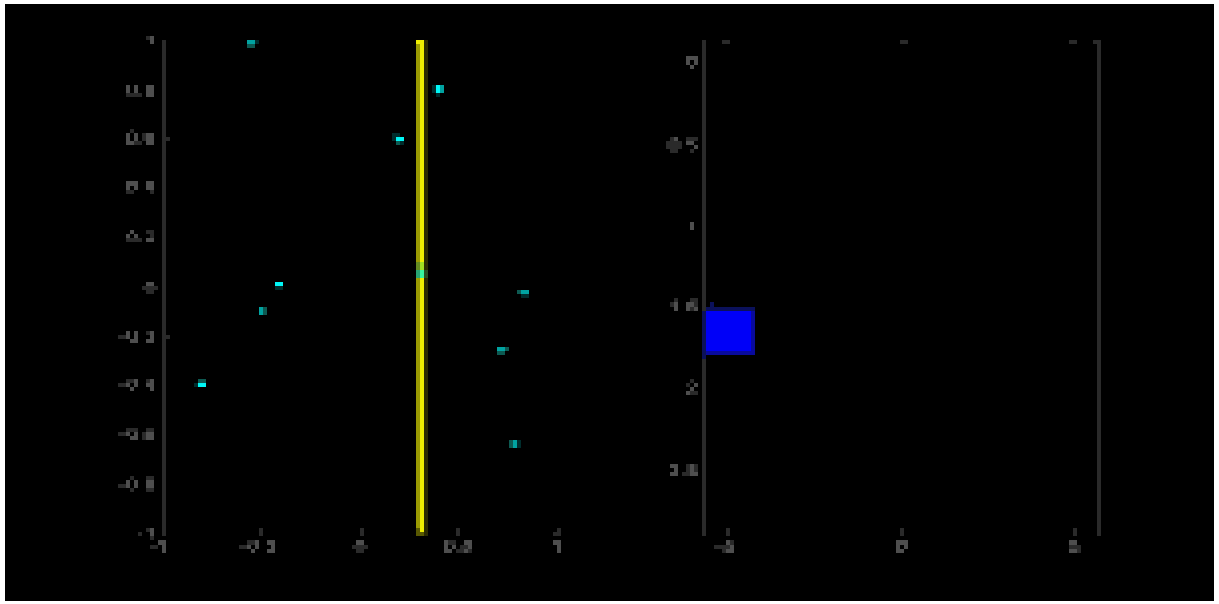
Each **point** in image space corresponds to a **sine curve** in Hough Space. Therefore, if we take a line of points, it translates into a whole bunch of sine curves in Hough Space, and the **intersection** of those sine curves gives the parameters of the line.



Hough transform

Consider a 100x100 image with a horizontal line at the middle. Take the first point of the line. You know its (x,y) values. Now in the line equation, put the values $\theta = 0, 1, 2, ..., 180$ and check the ρ you get. For every (ρ, θ) pair, you increment value by one in our accumulator in its corresponding (ρ, θ) cells. So now in accumulator, the cell (50,90) = 1 along with some other cells.

Now take the second point on the line. Do the same as above. Increment the the values in the cells corresponding to (ρ, θ) you got. This time, the cell (50,90) = 2. What you actually do is voting the (ρ, θ) values. You continue this process for every point on the line. At each point, the cell (50,90) will be incremented or voted up, while other cells may or may not be voted up. This way, at the end, the cell (50,90) will have maximum votes. So if you search the accumulator for maximum votes, you get the value (50,90) which says, there is a line in this image at distance 50 from origin and at angle 90 degrees. It is well shown in below animation (Image Courtesy: Amos Storkey)



HoughLines

Finds lines in a binary image using the standard Hough transform.

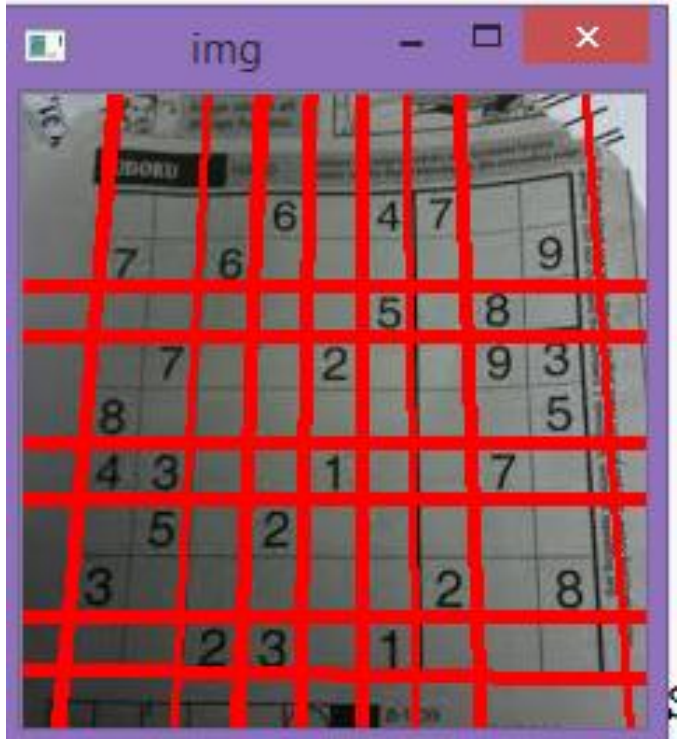
Python: cv2.HoughLines(image, rho, theta, threshold, lines, srn, stn) → lines

C++: void HoughLines(InputArray image, OutputArray lines, double rho, double theta, int threshold, double srn=0, double stn=0)

- **image** – 8-bit, single-channel binary source image. The image may be modified by the function.
- **lines** – Output vector of lines. Each line is represented by a two-element vector (ρ, θ)
- **rho** – Distance resolution of the accumulator in pixels.
- **theta** – Angle resolution of the accumulator in radians.
- **threshold** – Accumulator threshold parameter. Only those lines are returned that get enough votes
- same way as CV_HOUGH_STANDARD.

Let’s write some code…

Finding lines in an image



Let’s write some code…

Finding Lane Lines on the Road

Probabilistic Hough Transform

- Probabilistic Hough Transform is an optimization of Hough Transform we saw.
- It doesn't take all the points into consideration, instead take only a random subset of points and that is sufficient for line detection.
- it directly returns the two endpoints of lines.

Python: cv2.HoughLinesP(image, rho, theta, threshold, lines, minLineLength, maxLineGap) → lines

C++: void HoughLinesP(InputArray **image**, OutputArray **lines**, double **rho**, double **theta**, int **threshold**, double **minLineLength**=0, double **maxLineGap**=0)

Finds line segments in a binary image using the probabilistic Hough transform.

Hough Circle Transform

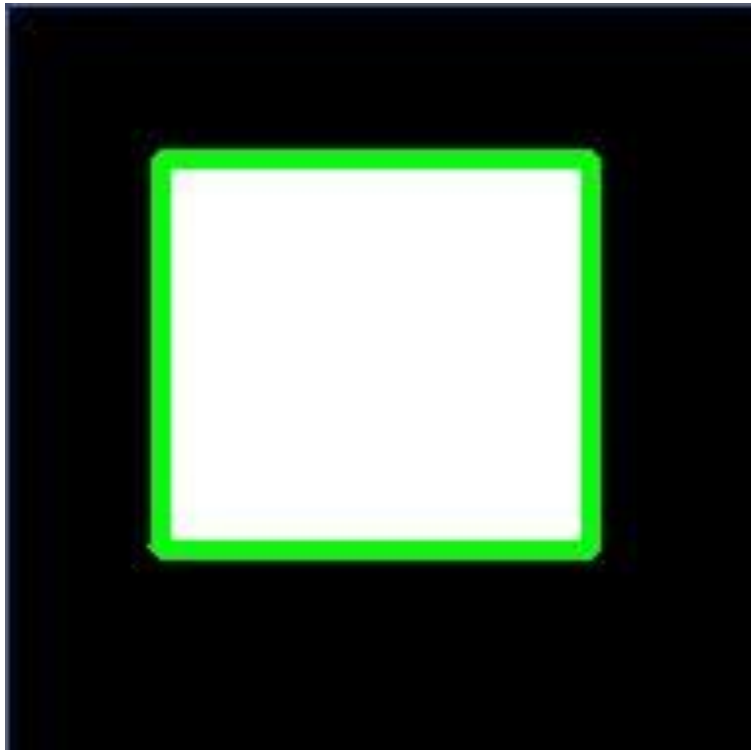
•The Hough Circle Transform works in a *roughly* analogous way to the Hough Line Transform explained in the previous tutorial.
•In the line detection case, a line was defined by two parameters (r, θ). In the circle case, we need three parameters to define a circle: $C : (x_{center}, y_{center}, r)$

Python: cv2.HoughCircles(image, method, dp, minDist[, circles[, param1[, param2[, minRadius[, maxRadius]]]]]) → circles

C++: void HoughCircles(InputArray **image**, OutputArray **circles**, int **method**, double **dp**, double **minDist**, double **param1**=100, double **param2**=100, int **minRadius**=0, int **maxRadius**=0)

Finds circles in a grayscale image using the Hough transform.

contours



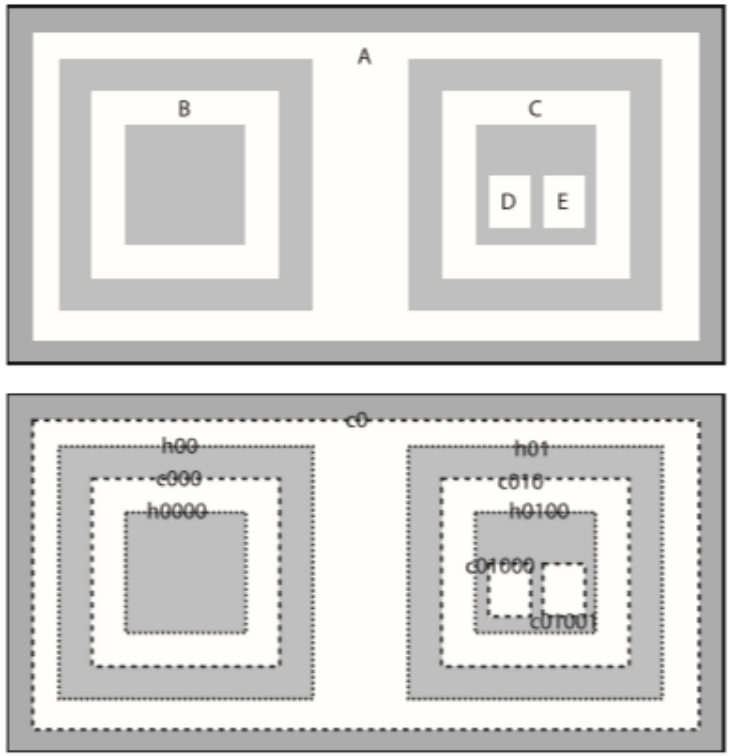
What is contour?

- Contours are simply the boundaries of an object.
- Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity.
- The contours are a useful tool for shape analysis and object detection and recognition.

Find contours

- In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.
- Contours are extracted using the OpenCV function findContours.
- It supports different contour extraction **modes**:
  - cv2.RETR_EXTERNAL: For extracting only external contours
  - cv2.RETR_CCOMP: For extracting both internal and external contours, and organizing them into a two-level hierarchy
  - cv2.RETR_TREE: For extracting both internal and external contours, and organizing them into a tree graph
  - cv2.RETR_LIST: For extracting all contours without establishing any relationships
- Also, you can specify **method** whether contour compression is required **:**
  - use cv2.CHAIN_APPROX_SIMPLE for collapsing vertical and horizontal parts of contours into their respective end points) or not (cv2.CHAIN_APPROX_NONE).
- The function returns a tuple of three elements, modified image, list of contours, and list of contour hierarchy attributes.

Find contours



CV_RETR_EXTERNAL

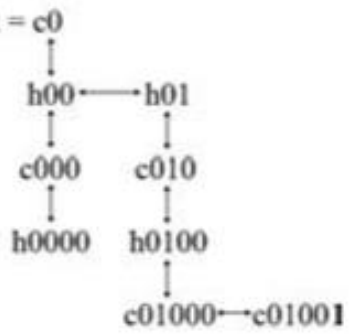
first = c0

CV_RETR_CCOMP

first = c01000↔c01001↔c010↔c000↔c0

h0100 h0000 h01↔h00

CV_RETR_TREE



CV_RETR_LIST

first = c01000↔c01001↔h0100↔c010↔c000↔h01↔h00↔c0

findContours()

- Finds contours in a binary image.

Python: cv2.findContours(image, mode, method, contours, hierarchy, offset) → contours, hierarchy

C++: void findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point())

hierarchy – Optional output vector, containing information about the image topology. It has as many elements as the number of contours. For each i-th contour contours[i], the elements hierarchy[i][0], hiearchy[i][1], hiearchy[i][2], and hiearchy[i][3] are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively.

drawContours()

Draws contours outlines or filled contours.

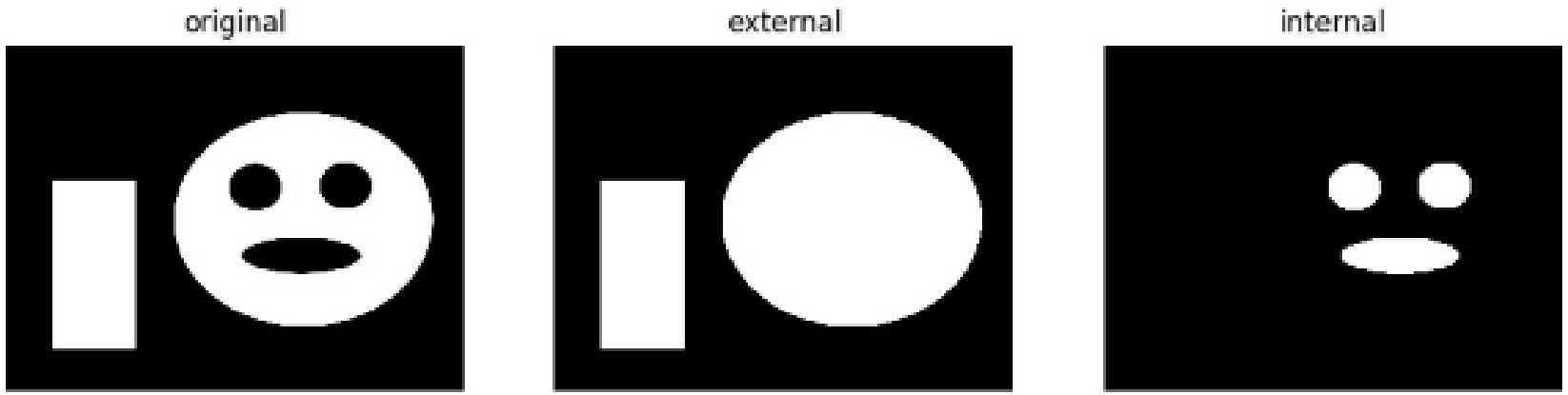
Python: cv2.drawContours(image, contours, contourIdx, color, thickness, lineType, hierarchy, maxLevel, offset) → None

C++: void drawContours(InputOutputArray image, InputArrayOfArrays contours, int contourIdx, const Scalar& color, int thickness=1, int lineType=8, InputArray hierarchy=noArray(), int maxLevel=INT_MAX, Point offset=Point())

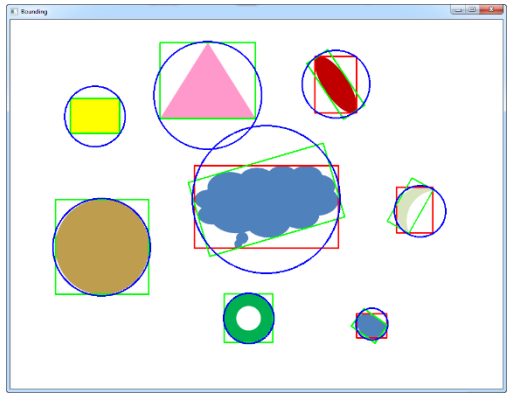
contourIdx – Parameter indicating a contour to draw. If it is negative, all the contours are drawn.
thickness – Thickness of lines the contours are drawn with. If it is negative (for example, thickness=CV_FILLED), the contour interiors are drawn.

Let’s write some code…

Finding and drawing contours in an image



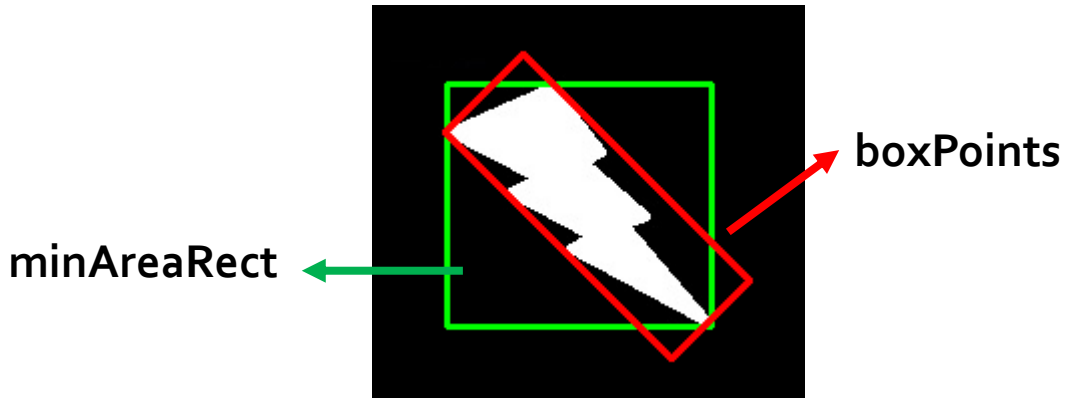
Working with contours



We will review the routines for computing a curve's length and area, getting the convex hull, and checking whether a curve is convex or not. Also, we will study how to approximate the contour with a smaller number of points. All of these things can be useful when you're developing an algorithm based on contour handling.

Working with contours

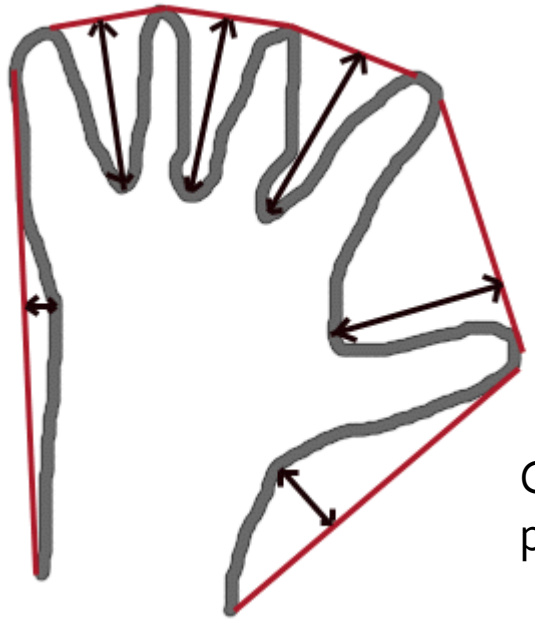
- **minAreaRect():** Calculates the up-right bounding rectangle of a point set.
- **boxPoints():** Calculates the rotated rectangle of a point set.
- **arcLength() :**Calculates a contour perimeter or a curve length.
- **contourArea():** Calculates a contour area.
- **fitEllipse():** Fits an ellipse around a set of 2D points.



Working with contours

- **convexHull() :** Finds the convex hull of a point set.
- **convexityDefects():** Finds the convexity defects of a contour.
- **isContourConvex():**Tests a contour convexity.

Convexity Defect is an area that do not belong to the object but is located inside of its convex hull



Convex hull is the smallest convex polygon that contains the contour

Exercise…

Finding finger tips