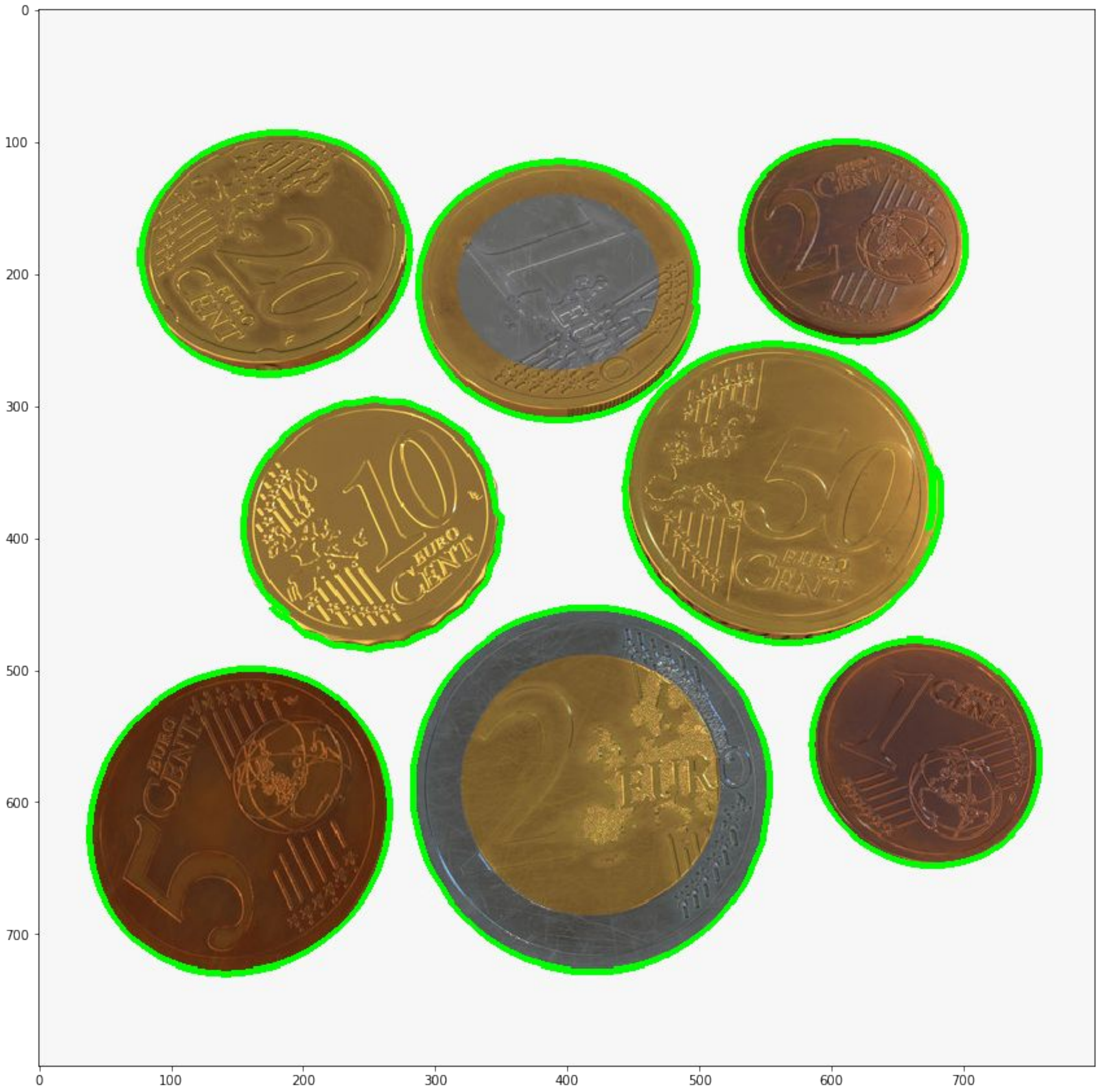


Image Segmentation & Contours

Finding Contours

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.

- For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
- Since OpenCV 3.2, findContours() no longer modifies the source image but returns a modified image as the first of three return parameters.
- 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.

Finding Contours

There are three arguments in **cv2.findContours()** function,

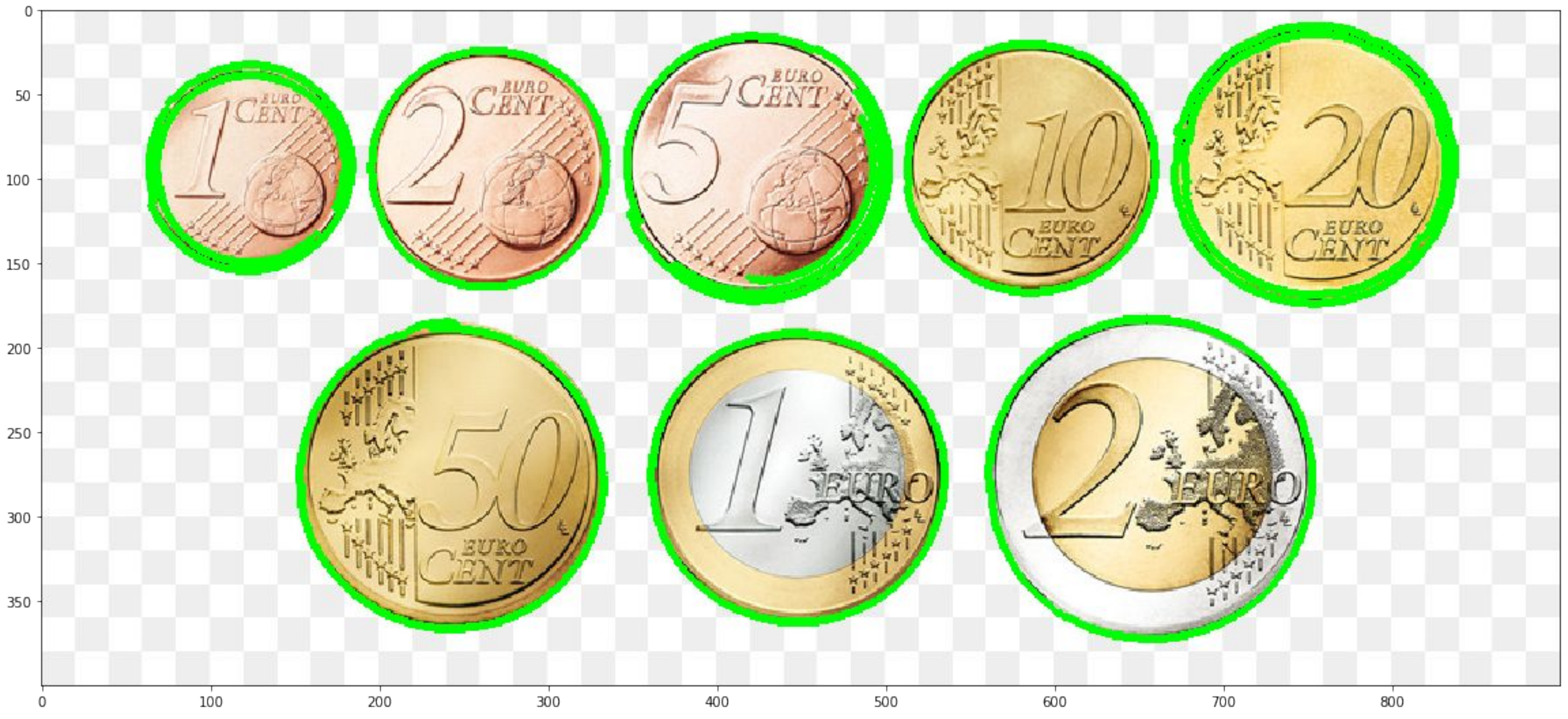
1. first one is source image,
2. second is contour retrieval mode, there are 4 options:
   a. RETR_LIST,
   b. RETR_EXTERNAL,
   c. RETR_CCOMP
   d. RETR_TREE
3. third is contour approximation method.

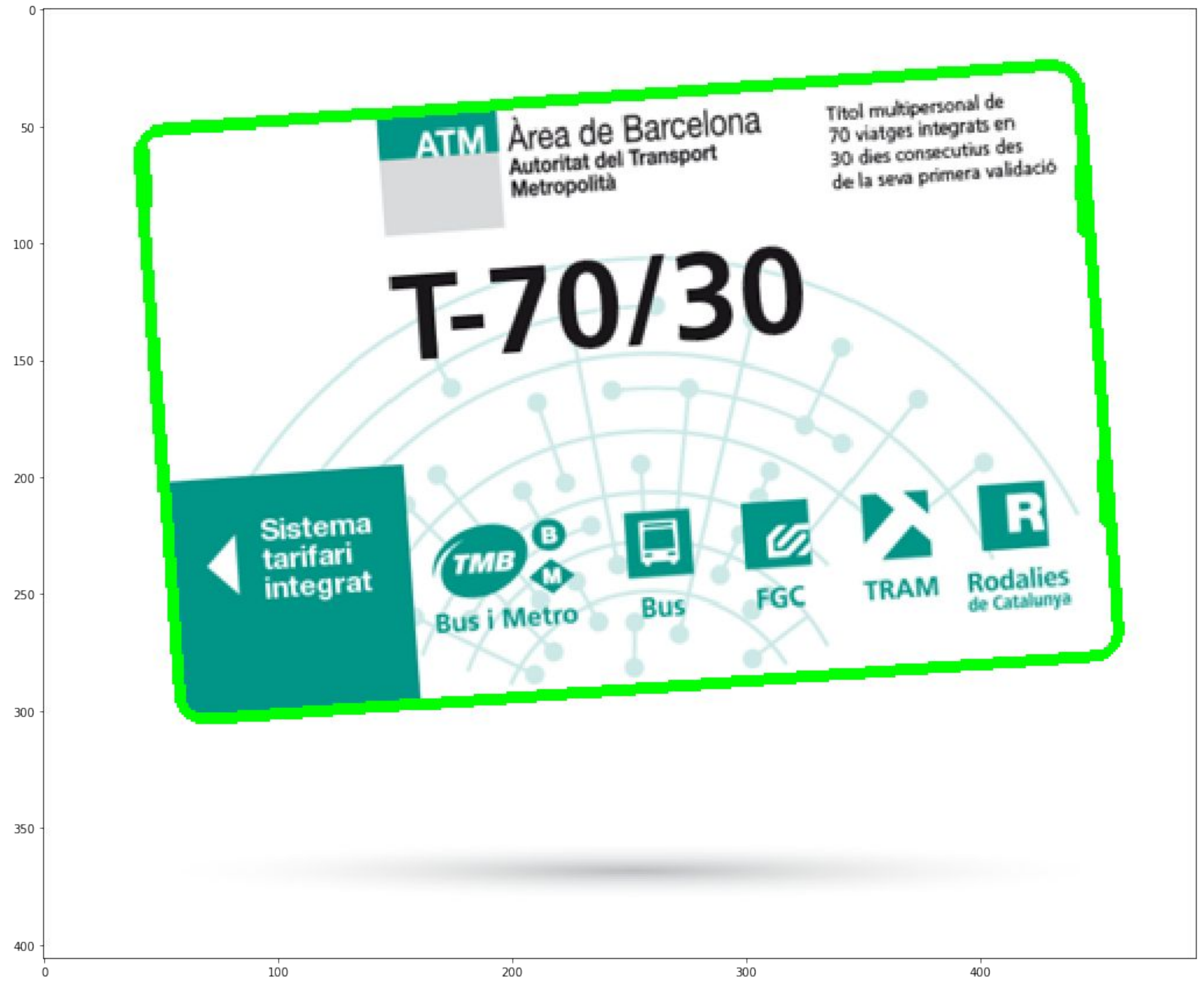
- It outputs a modified image, the contours and hierarchy. contours is a Python list of all the contours in the image.
- Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object.

Drawing contours

To draw the contours, **cv2.drawContours** function is used.

1. Its first argument is source image,
2. Second argument is the contours which should be passed as a Python list,
3. Third argument is index of contours (useful when drawing individual contour. To draw all contours, pass -1)
4. Remaining arguments are color, thickness.





Sorting contours

Once we find the contours of an image, if the order is important we will need to sort them.

We can sort contours by:

- **Orientation**: for example left to right
- **Area**: by finding the area inside the contours we can sort to get the biggest ones or smaller ones (normally discarting small contours is a way of getting rid of noise in the image)

Moments

Image moments help you to calculate some features like center of mass of the object and area of the object.

$$C_x = \frac{M_{10}}{M_{00}} \text{ and } C_y = \frac{M_{01}}{M_{00}}$$

- **Read about Image moment here:**
  - https://en.wikipedia.org/wiki/Image_moment

Contour Perimeter

It is also called arc length.

It can be found out using ***cv2.arcLength()*** function.

- First argument is the contour
- Second argument specify whether shape is a closed contour (if passed True), or just a curve.

perimeter = cv2.arcLength(cnt,True)

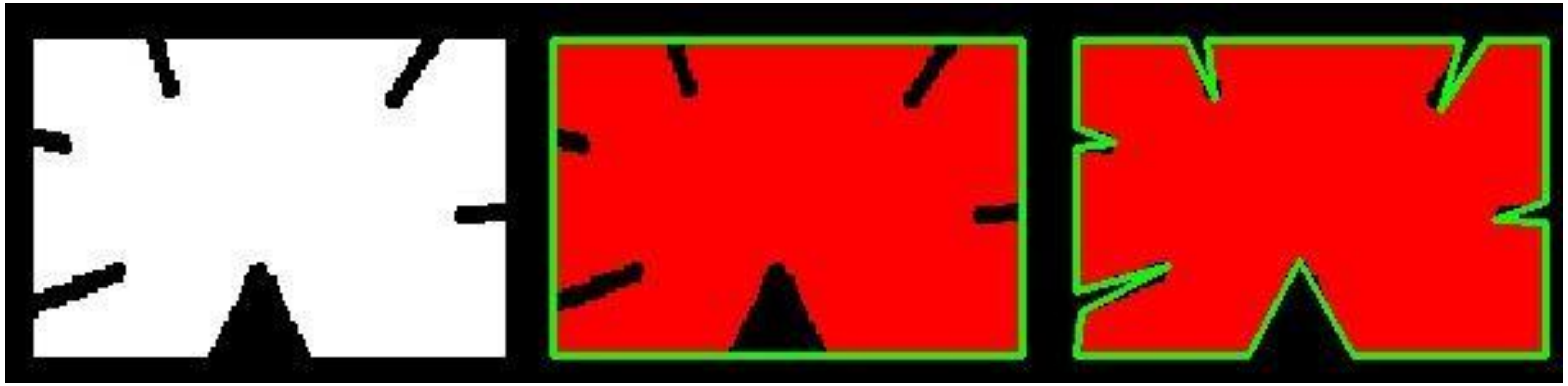
Contour Area

Contour area is given by the function:

1. ***cv2.contourArea()***
   a. *It receives only one parameter: the contour*
2. From moments: **M['m00'].**

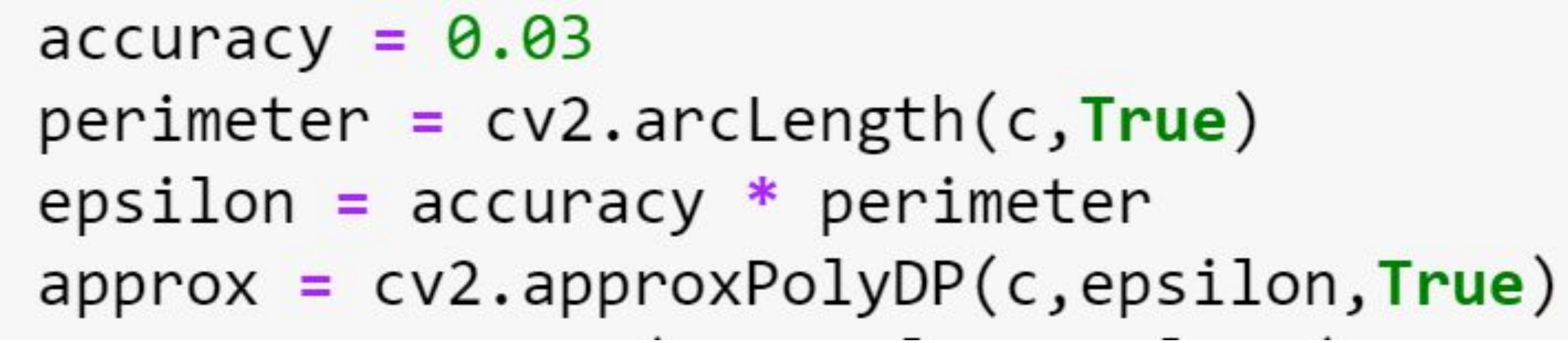
Approximating Contours

- It approximates a contour shape to another shape with less number of vertices depending upon the precision we specify.
- Suppose you are trying to find a square in an image, but due to some problems in the image, you didn't get a perfect square, but a "bad shape" you can approximate the shape with the function cv2.approxPolyDP the second argument is called epsilon, which is maximum distance from contour to approximated contour.
- It is an accuracy parameter. A wise selection of epsilon is needed to get the correct output. Depending on the epsilon value you will get different results as shown here



Advice for choosing the Epsilon value

- Small values give precise approximation
- Large values give more generic approximation
- A good rule of thumb is less than 5% of the contour perimeter

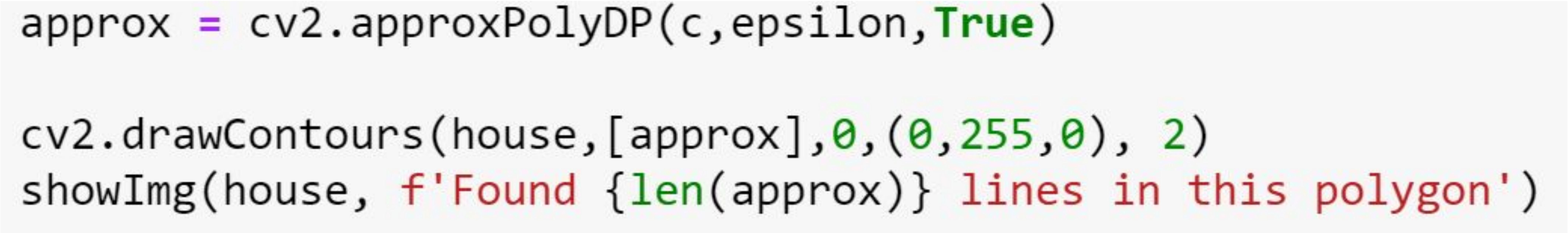


```
accuracy = 0.03
perimeter = cv2.arcLength(c,True)
epsilon = accuracy * perimeter
approx = cv2.approxPolyDP(c,epsilon,True)
```

Counting lines in a polygon

Once we got the image contour polygon approximations we can count the number of lines in each polygon by simply doing this:

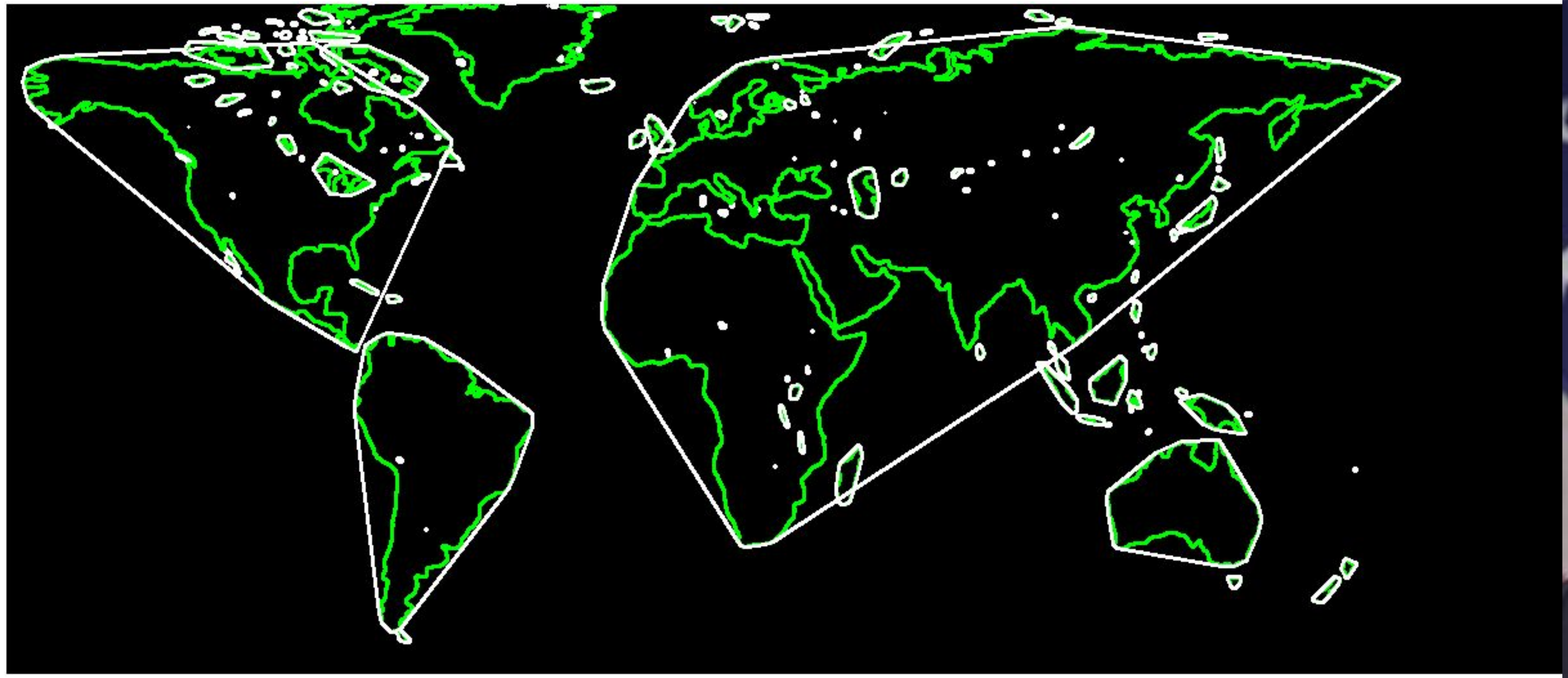
len(approximation)

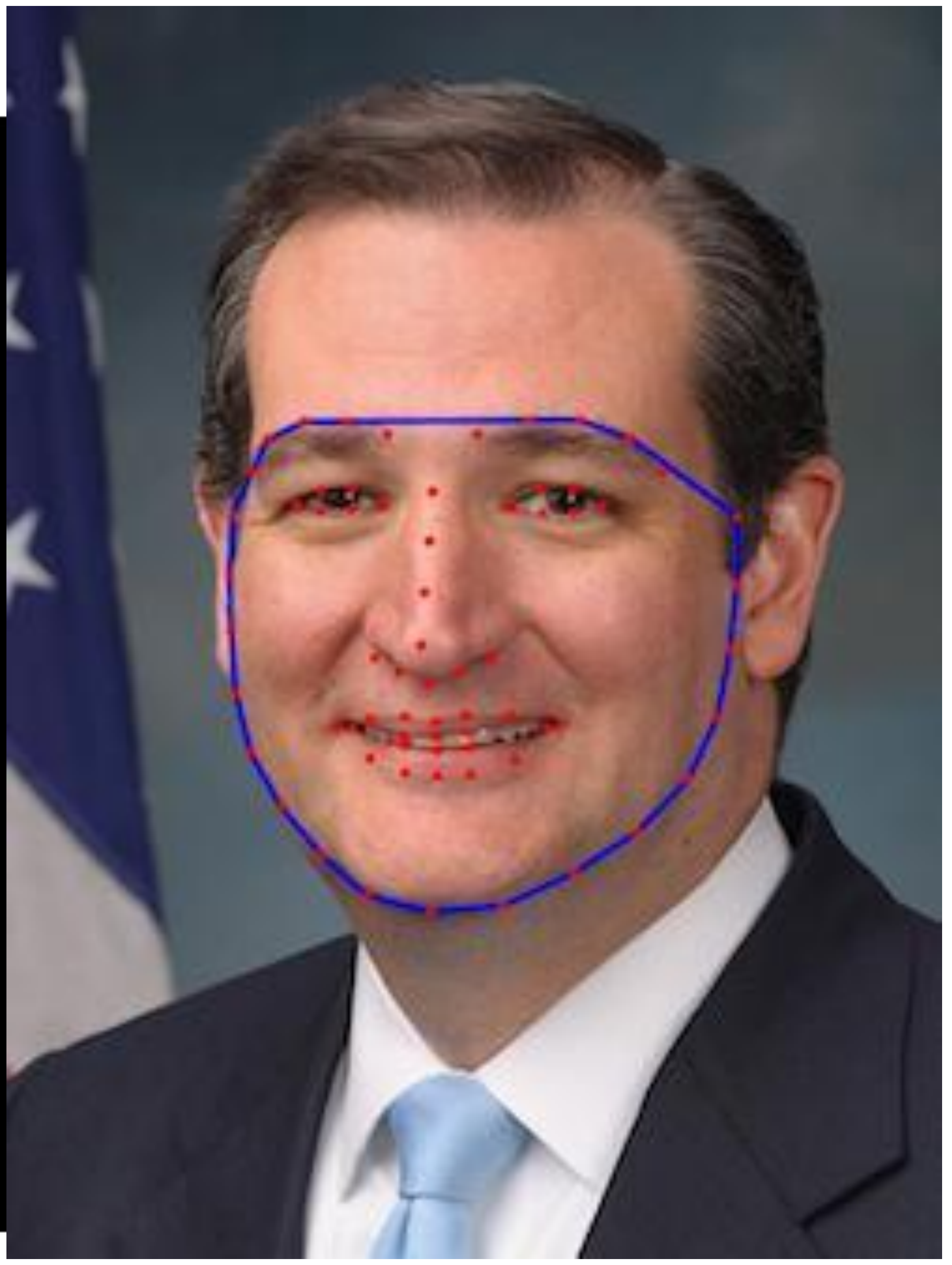


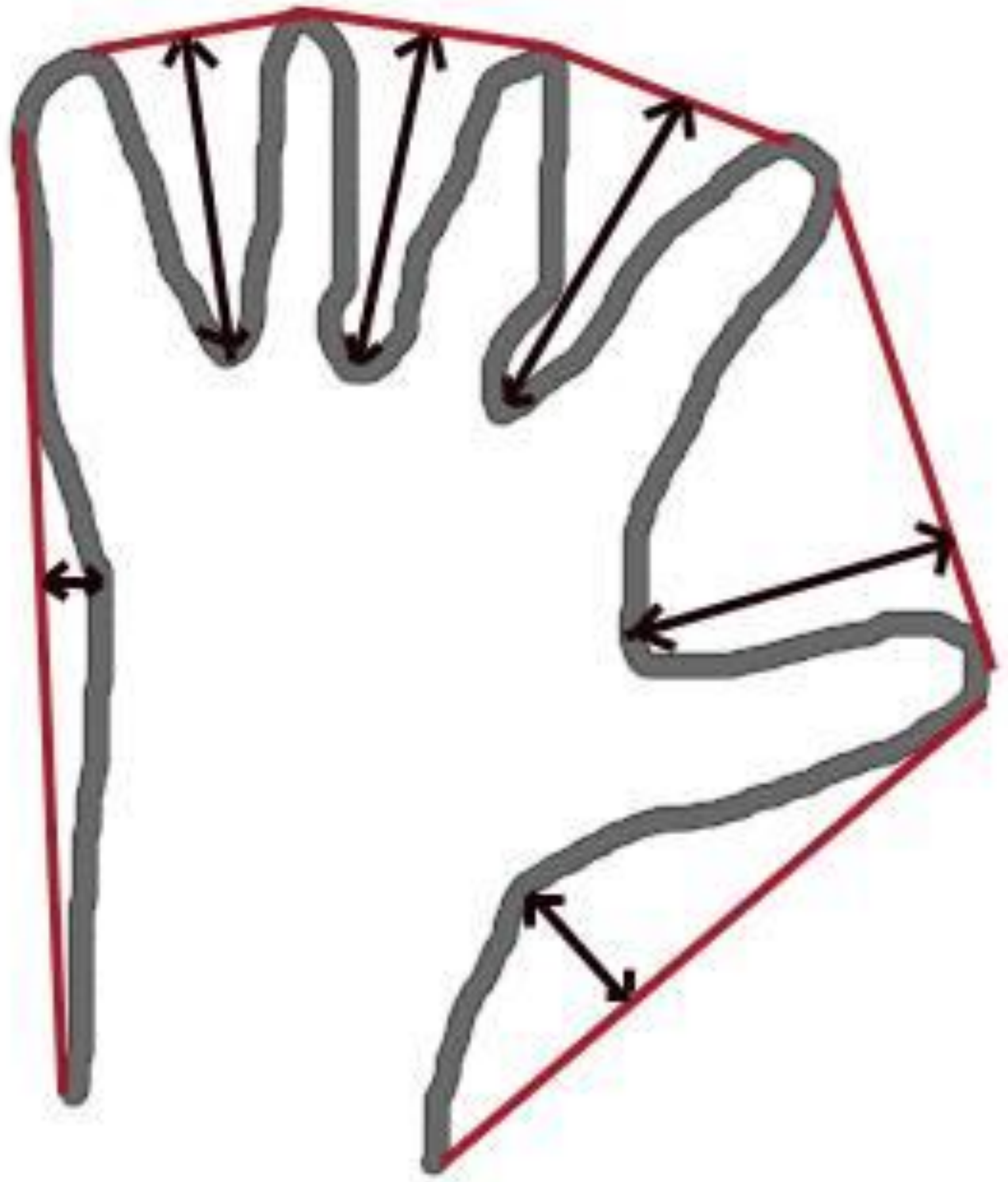
```
approx = cv2.approxPolyDP(c,epsilon,True)

cv2.drawContours(house,[approx],0,(0,255,0), 2)
showImg(house, f'Found {len(approx)} lines in this polygon')
```

Convex Hull

The *Convex Hull* of a shape or a group of points is a tight fitting convex boundary around the points or the shape.







https://www.learnopencv.com/convex-hull-using-opencv-in-python-and-c/

https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html

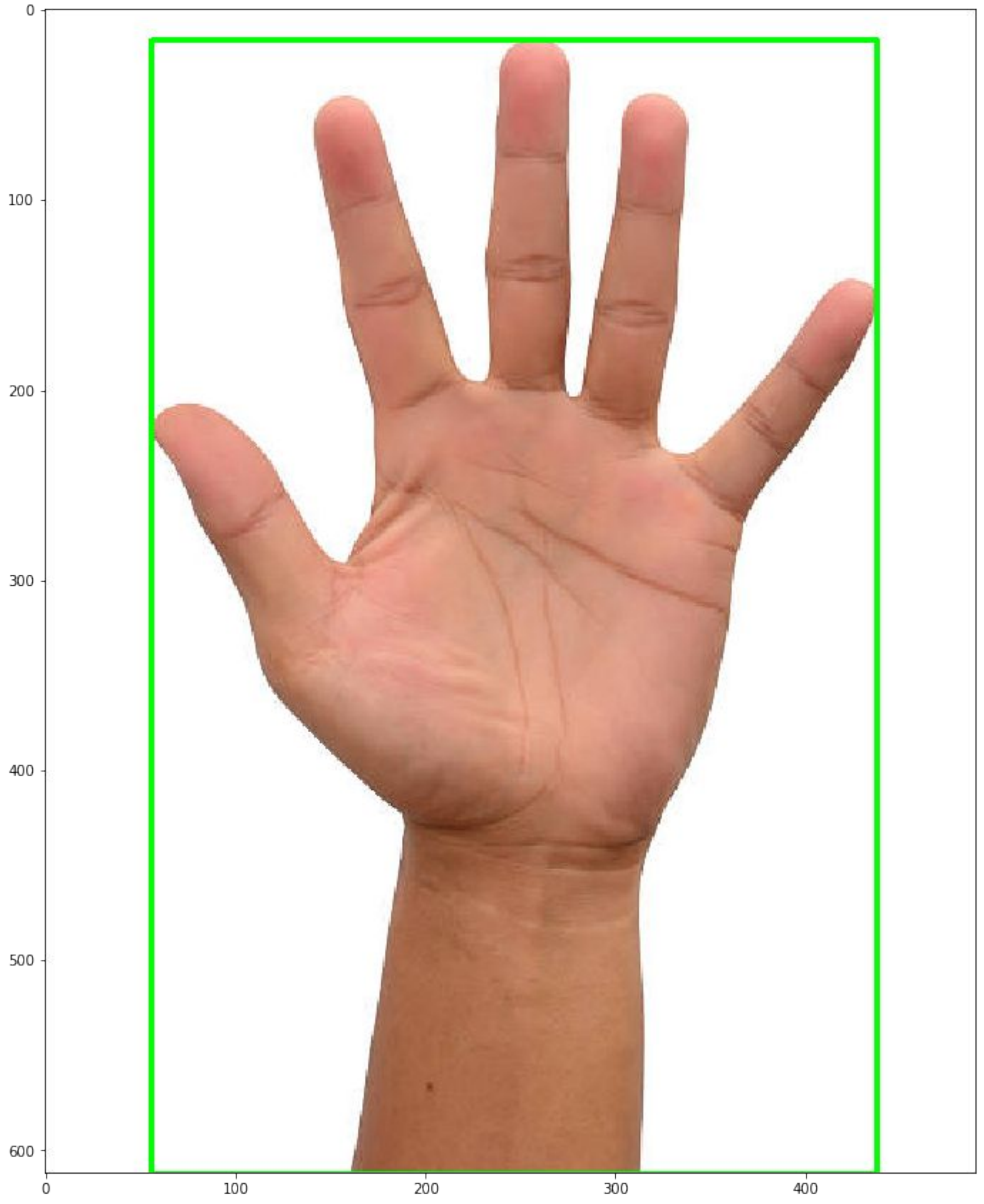
Bounding Rectangle

Is the minimum fitting rectangle around an object, it does not take into consideration rotation the rectangle coordinates can be found with:

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

Where *cnt* are the contours of the image

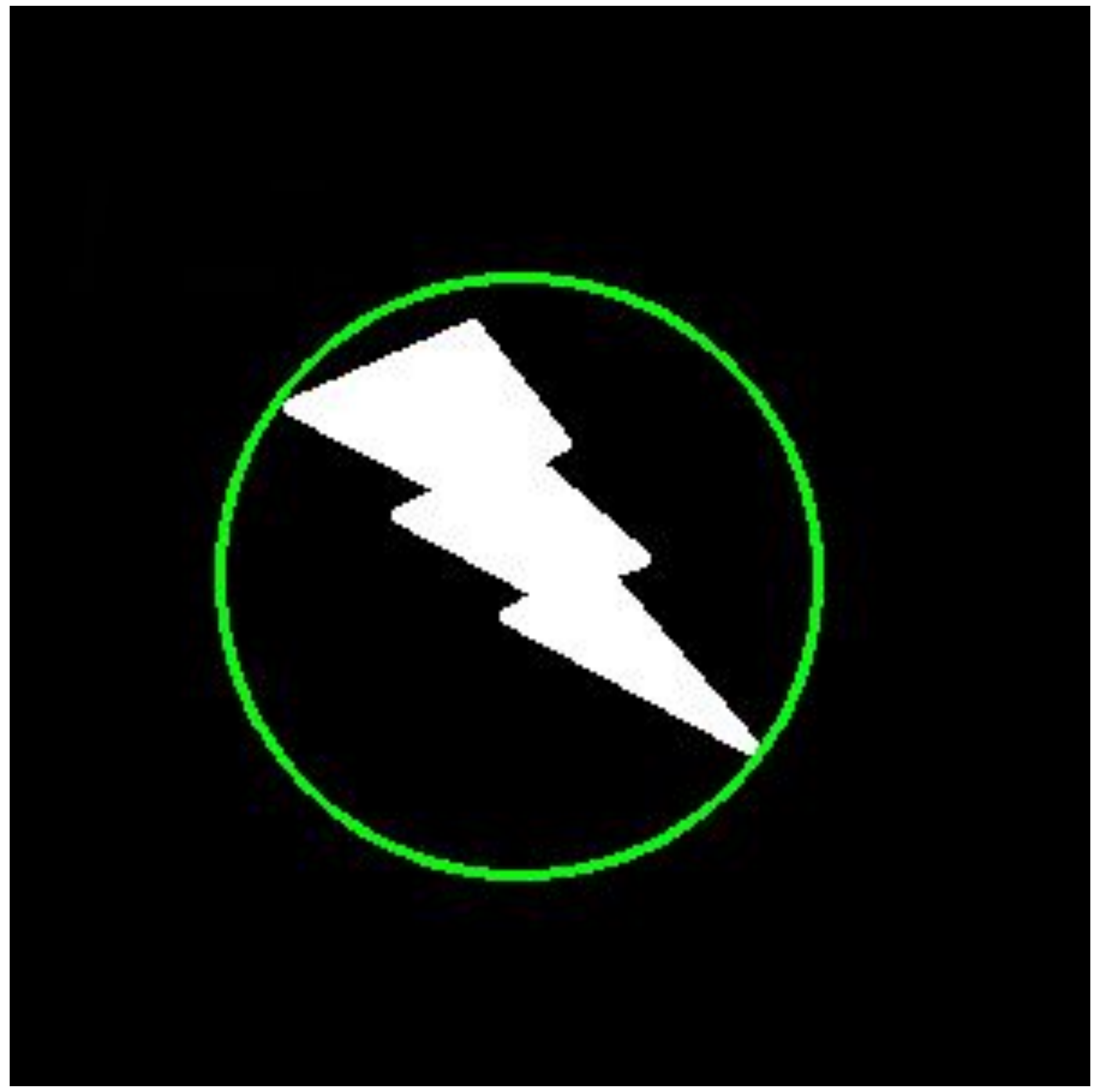
Interesting fact: it also works with a threshed image

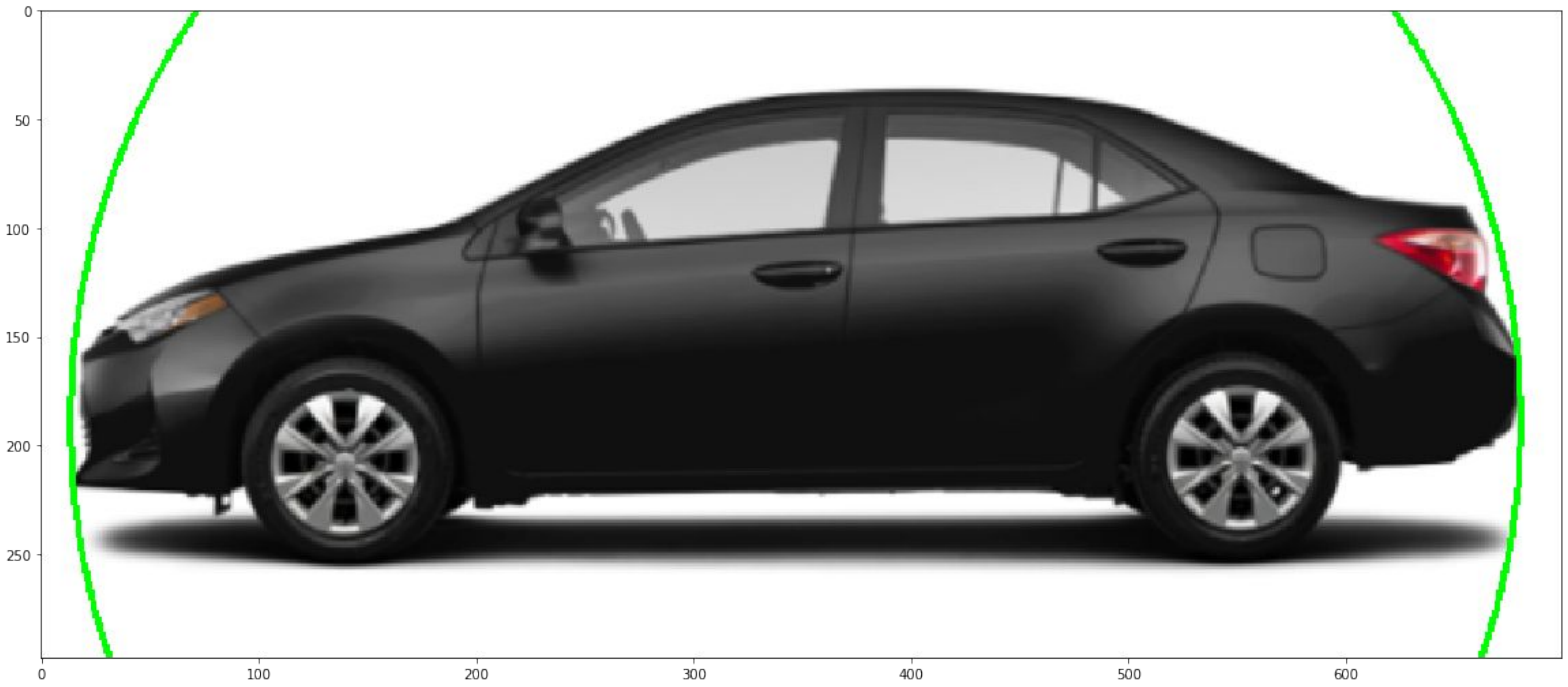


Minimum Enclosing Circle

The circumcircle of an object using the function *cv2.minEnclosingCircle()*

It is a circle which completely covers the object with minimum area.



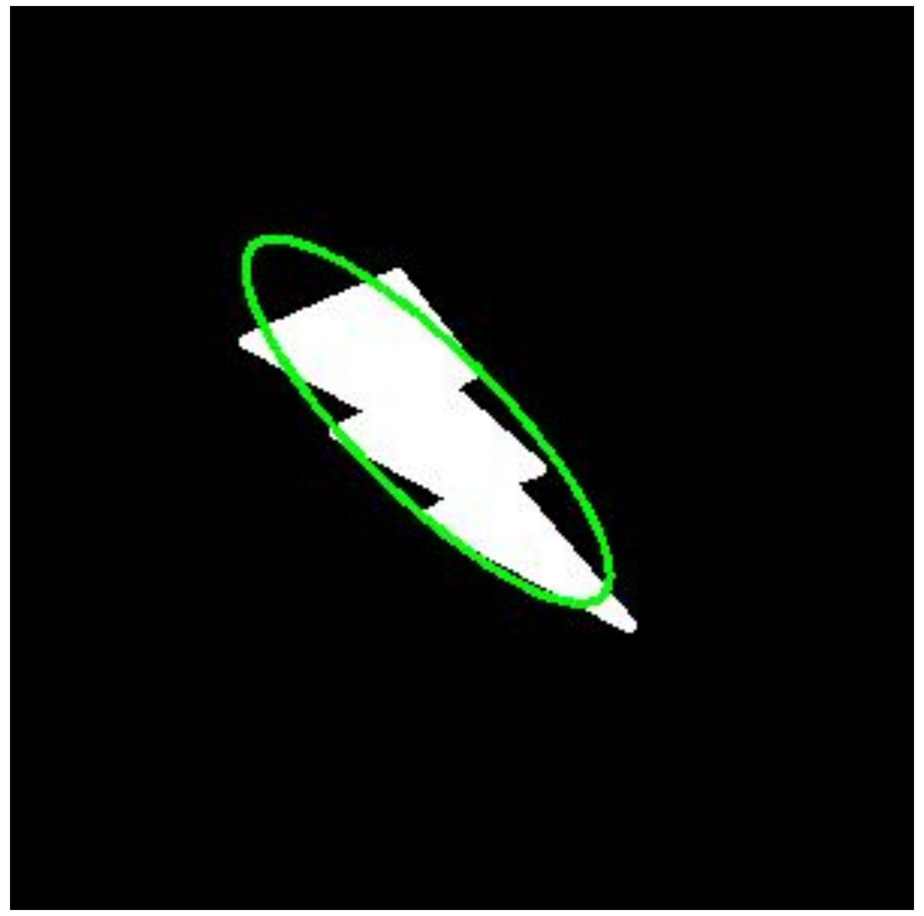


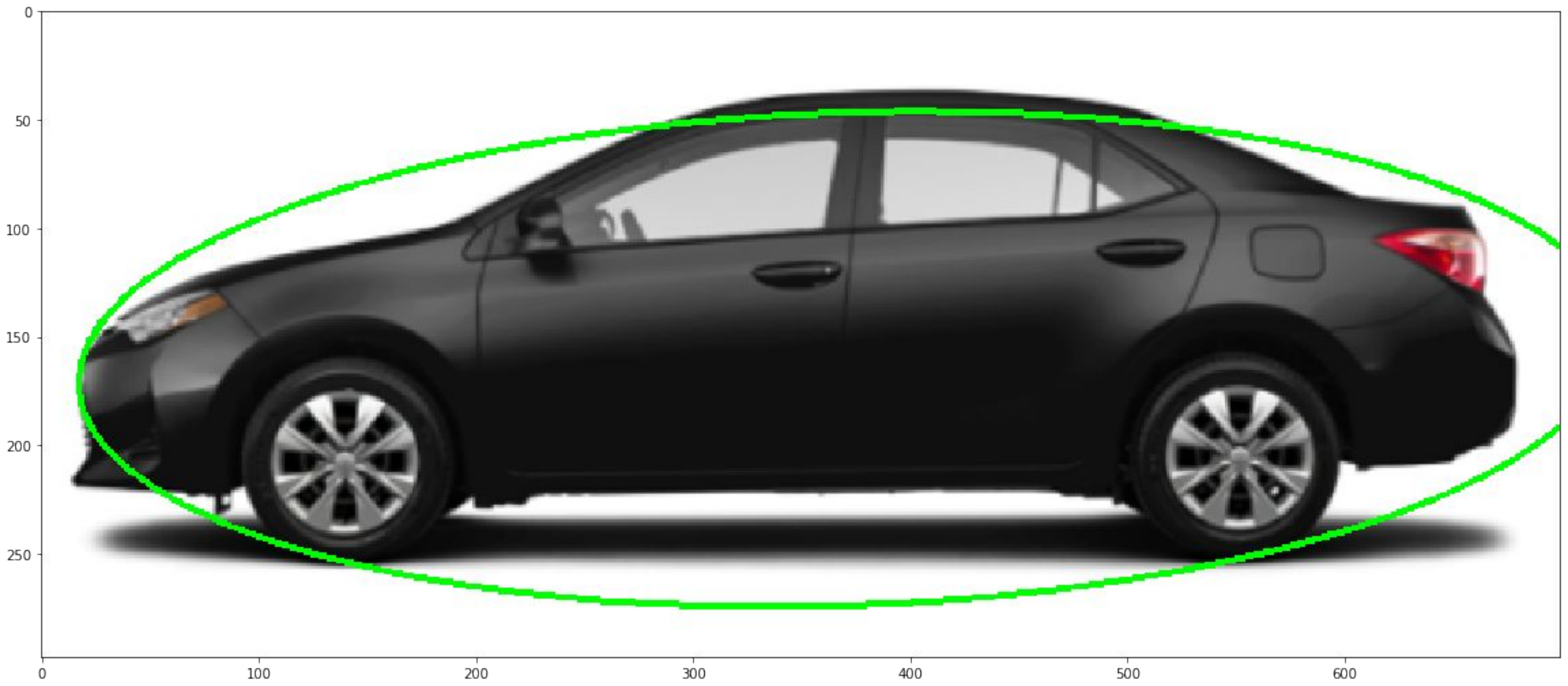
Fitting an ellipse

- To find the fitting ellipse of an object, it will rotated as needed:

  ellipse = cv2.fitEllipse(cnt)

- It also receives the object contour as an input parameter.
- Make sure it is one contour and not an array of contours, you will get an error otherwise



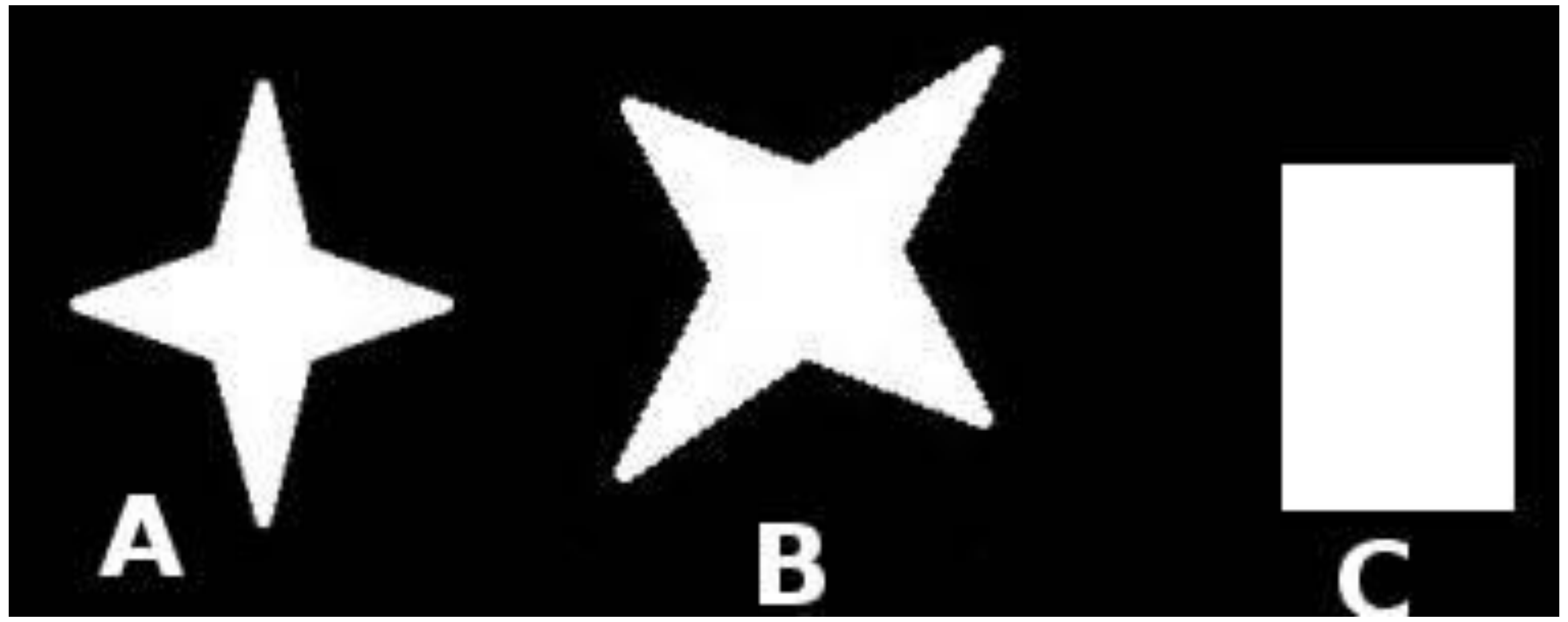


Matching contour shapes

OpenCV comes with a function *cv2.matchShapes()* which enables us to compare two shapes, or two contours and returns a metric showing the similarity.

The lower the result, the better match it is. It is calculated based on the hu-moment values.

Example: using A from the image as the template we get the following scores when comparing them



Matching Image A with itself = 0.0

Matching Image A with Image B = 0.001946

Matching Image A with Image C = 0.326911

Line Detection with Hough Transform

We can detect lines with Hough Line Transform:

- The Hough Line Transform is a transform used to detect straight lines.
- To apply the Transform, first an edge detection pre-processing is desirable

lines = cv.HoughLines(dst, 1, np.pi / 180, 150)

- lines: A vector that will store the parameters (r,θ) of the detected lines
- dst: Output of the edge detector. It should be a grayscale image (although in fact it is a binary one)
- rho : The resolution of the parameter r in pixels. We use 1 pixel.
- theta: The resolution of the parameter θ in radians. We use 1 degree (CV_PI/180)
- threshold: The minimum number of intersections to "*detect*" a line

In [ ]:

Probabilistic Hough Transform *(adds 2 new parameters)*

lines = cv2.HoughLinesP(dst,1,np.pi/180,100,minLineLength,maxLineGap)

- *minLineLength - Minimum length of line. Line segments shorter than this are rejected.*
- *maxLineGap - Maximum allowed gap between line segments to treat them as single line.*

Circle Detection

OpenCV implements another Hough Transform version to detect Circles:

cv2.HoughCircles(image, method, dp, minDist)

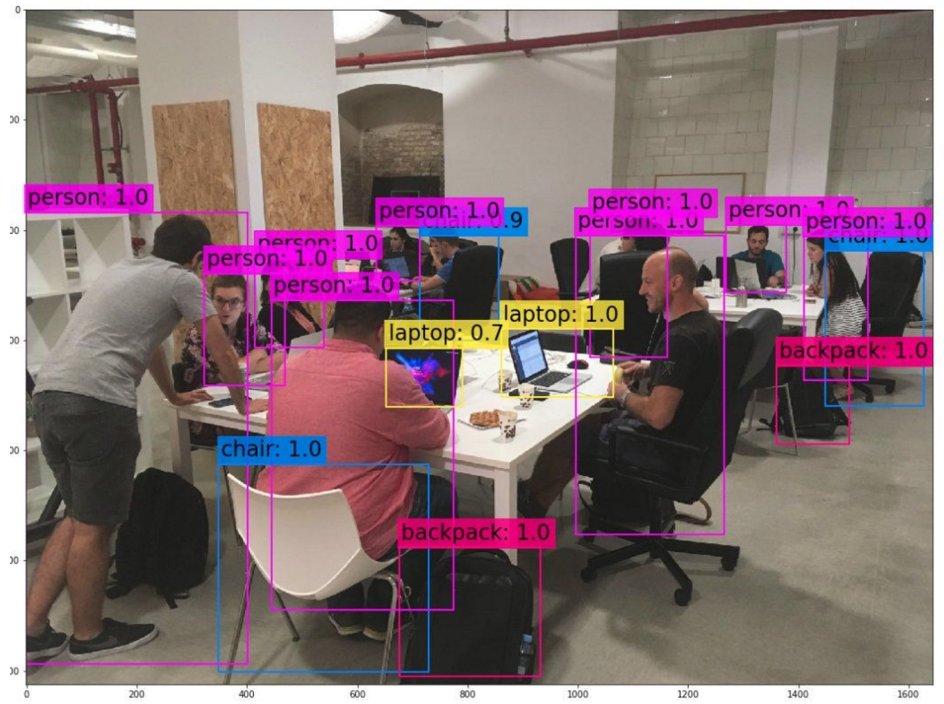
- *src_gray: Input image (grayscale)*
- *circles: A vector that stores sets of 3 values: x_{c}, y_{c}, r for each detected circle.*
- *method: Define the detection method. Currently CV_HOUGH_GRADIENT this is the only* In [ ]: *one available in OpenCV*
- *dp = 1: The inverse ratio of resolution*
- *min_dist = src_gray.rows/8: Minimum distance between detected centers*
- *param_1 = 200: Upper threshold for the internal Canny edge detector*
- *param_2 = 100*: Threshold for center detection.*
- *min_radius = 0: Minimum radio to be detected. If unknown, put zero as default.*
- *max_radius = 0: Maximum radius to be detected. If unknown, put zero as default*

You may also want to check

- Blob detection
- Fitting a line
- Fitting with a rotated rectangle

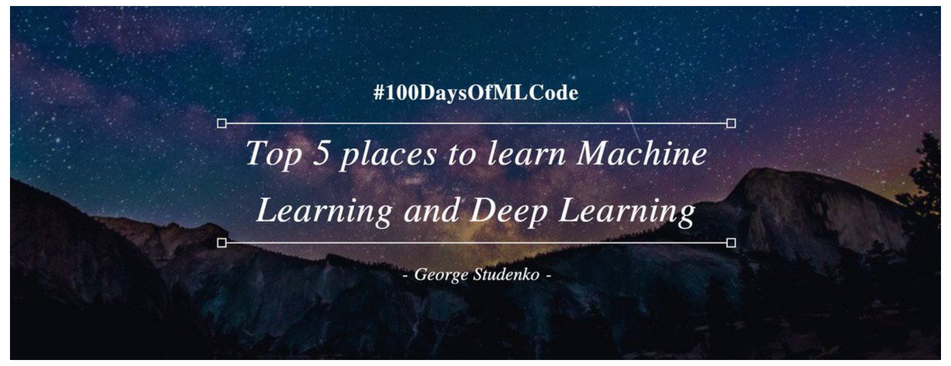
```
In [ ]:
```

About me...



#100DaysOfMLCode

By George Studenko


#100DaysOfMLCode

Top 5 places to learn Machine Learning and Deep Learning

- George Studenko -

George Studenko

Software Engineer and AI Enthusiast

georgestudenko

georgestudenko

george-studenko