# MODUL 11 – Object Detection Methods: Template Matching, Edge Detection, Corner Detection, Grid Detection, Contour Detection

## A. PURPOSE

- Students understand the concept of object detection
- Students understand several methods that can be used in the object detection process
- Students can implement several methods in the object detection process using Python on Google Colab

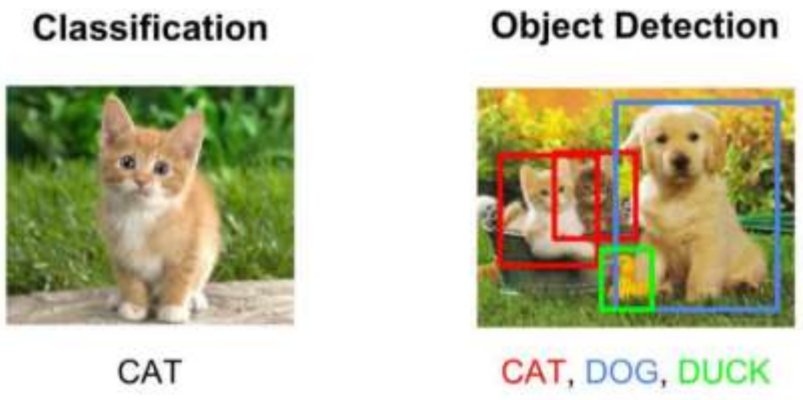
## B. TOOLS AND MATERIALS

1. PC/LAPTOP
2. Github
3. *Google Colaborator*

## C. BASIC THEORY

### C.1 Object Detection Concept

Object detection (object detection) is a computer vision technique (computer vision) that identify and find objects in an image or video. In particular, the object detection process will draw a bounding box around the detected object, so that it can be seen where the object is located in the image / video. The purpose of object detection is to detect all object instances of a known class, such as people, cars or faces in an image. Generally, there are only a few objects in an image, but there are an enormous number of possible locations and scales at which they can appear. The following is an illustration of the concept of object detection:



In a computer vision system, object detection is the first step that is carried out because the results of object detection can make it possible to obtain more information about the detected object and the location of the object. After an object is detected (for example, a face), further information can be obtained, including: (i) recognizing a specific example (for example, to identify a subject from a detected face), (ii) tracking an object in an image sequence (for example, to track a face in video), and (iii) extract more information about the object (for example, to determine the gender of the subject). Some examples of object detection implementations include:

1. Video surveillance (CCTV), an object detection model capable of tracking many people at once, in real-time.

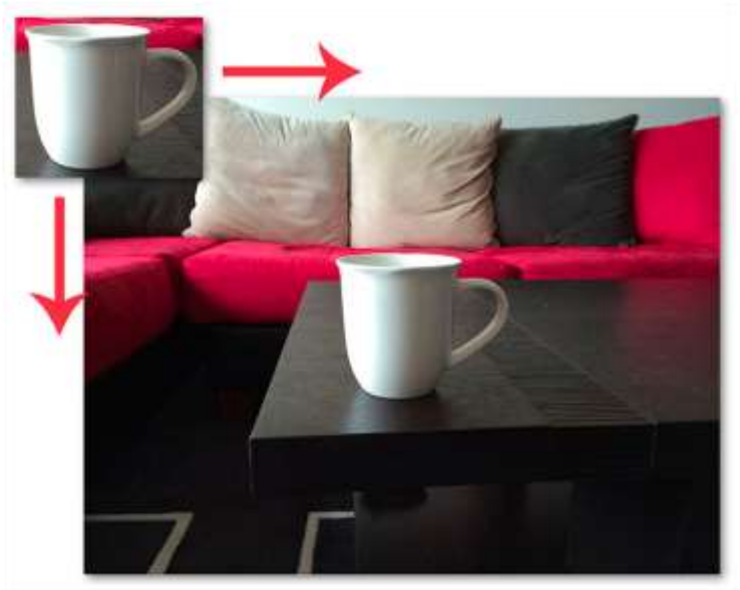
2. Crowd counting, for crowded areas such as amusement parks, malls, or city squares, object detection can help business owners and city governments measure more effectively different types of traffic, and some plans such as store hours, employee shifts, and allocating resources.
3. Anomaly detection, for example in agriculture, an object detection model can accurately identify and locate potential plant disease locations, thus enabling farmers to detect threats to their crops that cannot be seen with the naked eye.
4. Self-driving cars, object detection is a technique that underlies the creation of a driverless car. This system must be able to identify, find, and track objects in the vicinity in order to move on the road safely and efficiently.

## C.2 Template Matching

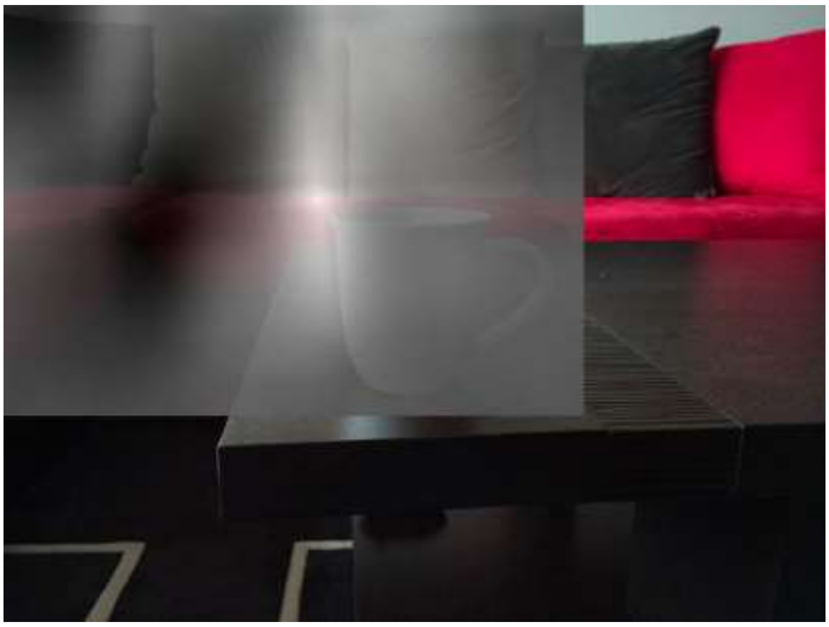
Template matching is the most basic form of object detection. Template matching is a method for locating and locating a template image on a larger image. Two images are required to implement the template matching method, namely:

- **Source image (I):** an input image that is expected to match the template
- **Template image (T):** a cut object that will be searched for in the image source.

To find the template in the source image source, shift the template from left to right and from top to bottom:



At each location (x, y), a metric is calculated to represent how "good" or "bad" the match was. The calculation of the correlation coefficient is carried out to determine how similar the pixel intensity of the two image pieces is. For each location T in the first , the metric calculation result will be stored in the payoff matrix R. Each coordinate (x, y) in the source image will contain an element in the resulting matrix R :



The bright locations of the R yield matrix show the best match value, while the dark areas show very little correlation between the source and template images.

OpenCV implements template matching using the matchTemplate () function . There are 6 methods available in matchTemplate (), namely:

1. method=TM_SQDIFF

$$R(x,y)=\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2$$

2. method=TM_SQDIFF_NORMED

$$R(x,y)=\frac{\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}}$$

3. method=TM_CCORR

$$R(x,y)=\sum_{x',y'}(T(x',y')\cdot I(x+x',y+y'))$$

4. method=TM_CCORR_NORMED

$$R(x,y)=\frac{\sum_{x',y'}(T(x',y')\cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}}$$

5. method=TM_CCOEFF

$$R(x,y)=\sum_{x',y'}(T'(x',y')\cdot I'(x+x',y+y'))$$

6. method=TM_CCOEFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

Documentation of template matching in OpenCV can be seen at the following link:
https://docs.opencv.org/3.4/de/da9/tutorial_template_matching.html

## C.3 Edge Detection

Edge detection is an operation performed to detect edges that delimit two areas of an image that have different brightness levels. Edge detection of a digital image is a process to look for differences in intensity that state the boundaries of an object (sub-image) in the entire digital image in question. A point (x, y) is said to be the edge of an image if that point has a high difference with its neighbors. Edge detection is widely used to identify an object in an image. There are 3 edge detection methods provided by OpenCV, namely:

1. Sobel Edge Detection
   Sobel is one of the most commonly used edge detectors. The Sobel filter works by calculating the gradient of the image intensity at each pixel in an image. When using this filter, images can be processed in the X and Y directions separately or simultaneously. Sobel is based on the convolution of images in horizontal and vertical directions using a 3x3 filter, as follows:

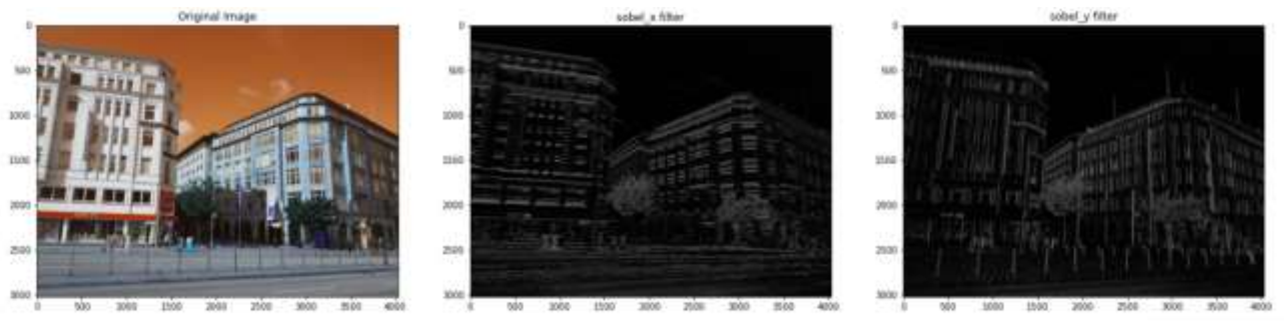
| | | |
|---|---|---|
| -1 | 0 | +1 |
| -2 | 0 | +2 |
| -1 | 0 | +1 |

Gx

| | | |
|---|---|---|
| +1 | +2 | +1 |
| 0 | 0 | 0 |
| -1 | -2 | -1 |

Gy

   Examples of results from sobel edge detection are as follows:

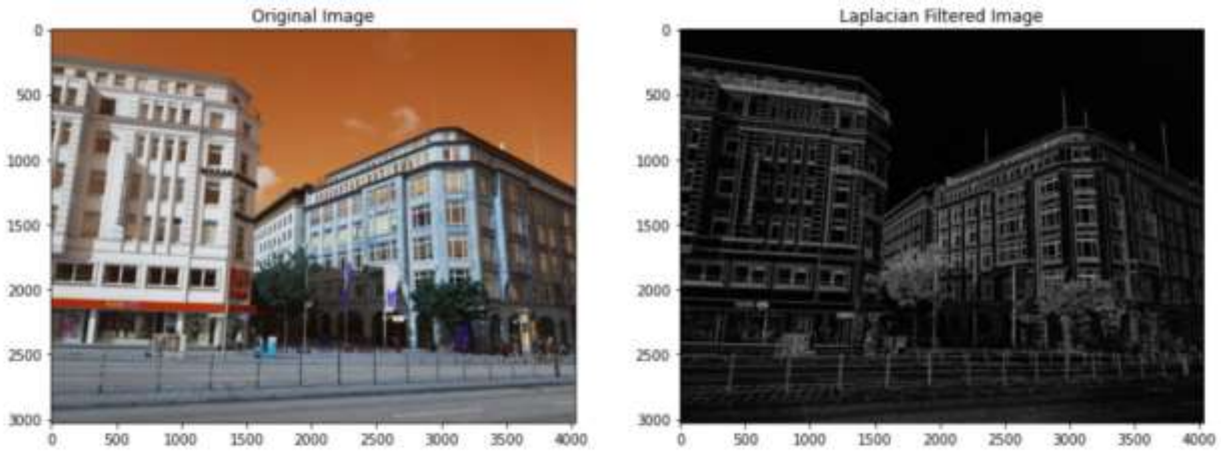


2. Laplacian Edge Detection
   The 3x3 kernels that are commonly used in Laplacian Edge Detection are as follows:

| | | |
|---|---|---|
| 0 | −1 | 0 |
| −1 | 4 | −1 |
| 0 | −1 | 0 |

| | | |
|---|---|---|
| −1 | −1 | −1 |
| −1 | 8 | −1 |
| −1 | −1 | −1 |

This method is very sensitive to noise, therefore to fix it, an image will generally be smoothed first using a Gaussian filter before applying Laplacian. Examples of results from laplacian edge detection are as follows:



3. Canny Edge Detection
   Canny Edge Detection is the most commonly used and most effective method. This method works in 4 stages, namely:
   a. Smooth the image with a Gaussian filter to reduce noise
   b. Calculate the gradient using one of the Sobel gradient operators
   c. Extracting edge point: Non-maximum suppression
   d. Linking and thresholding: Hysteresis

Canny edge detection is based on the concept that the intensity of the image will be highly valued at the edges. The problem with this concept is that if the image has noise, the noise will also be detected as an edge. So the first step in Canny edge detection is noise removal, one way is to apply a Gaussian filter to smooth the image before continuing processing.

The second step in Canny's edge detection process is gradient computation, which is done by calculating the degree of change in intensity (gradient) in the image along the direction of the gradient. The image that has been refined in the first step is then filtered with the Sobel kernel, both horizontally and vertically, to obtain the first derivative in the horizontal (Gx) and vertical (Gy) direction. From the two images, you can search for the edge and direction gradient for each pixel as follows:

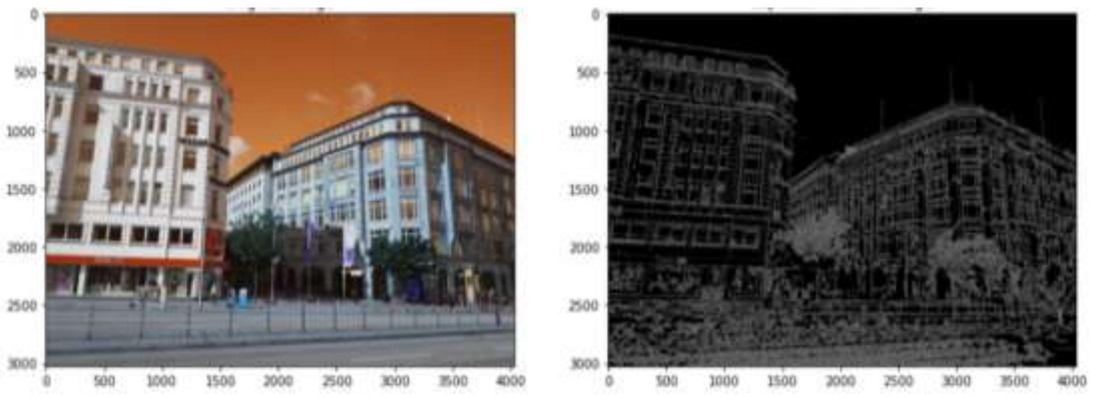
$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

The intensity of the image is at its highest point at the edges, but in reality, the intensity does not peak at a single pixel; conversely, there are contiguous pixels of high intensity. At each pixel location, Canny edge detection will compare the pixels and select the local maximum in the 3X3 neighbor in the direction of the gradient. This process is known as Non-maximum suppression.

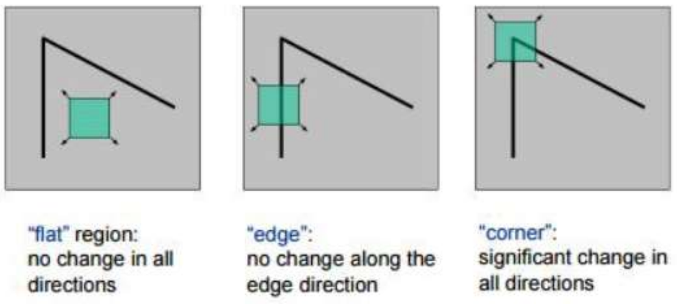
The third step creates a thin but broken outline. The final step is to repair / connect the broken edges using the hysteresis thresholding technique. In hysteresis thresholding, there are two thresholds: a high threshold and a low threshold. Any pixel with a gradient value higher than the high threshold is automatically saved as an edge. Meanwhile, pixels whose gradient is between the high threshold and low threshold will be handled in two ways. The pixels will be checked for possible edge connection; pixels will be saved if connected and will be discarded / ignored otherwise. Pixels with a gradient lower than the low threshold are discarded automatically.

Examples of results from canny edge detection are as follows:



## C.4 Corner Detection

An angle can be interpreted as the intersection of two sides. A corner cannot be defined on a single pixel, because there is only one gradient at each point. Gradient is the direction of change in the intensity of brightness in an image. Angle is an important feature of an image, and is generally referred to as an immutable interest point when translated, rotated, and illuminated.


"flat" region:
no change in all directions
"edge":
no change along the edge direction
"corner":
significant change in all directions

There are 2 corner detection methods provided by OpenCV, namely:

1. Harris corner detection
   Harris Corner Detection is an angle detection system that is often used because it is able to produce consistent values on images that experience rotation, scaling, lighting variations, or have a lot of noise in the image. Angle detection with the Harris method is based on variations in signal intensity. A large variation in intensity indicates an angle in the image. How the Harris

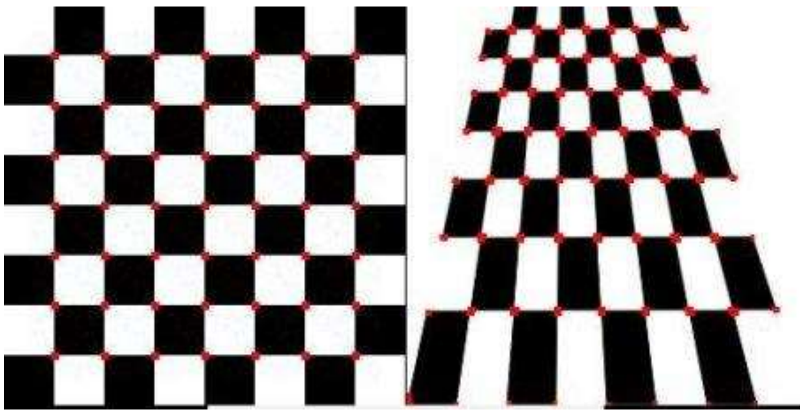
method works is to find the difference in intensity for displacement (u, v) in all directions, with the following equation:

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \, [\underbrace{I(x+u,y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

The window function can be a Gaussian filter that gives weight to the pixels below it. This function of E (u, v) should be maximized for angle detection. In OpenCV, the methods used are:
cv2.cornerHarris (src, dest, blockSize, kSize, freeParameter, borderType)
with parameters:

- Src: input image
- Dest: image to store Harris detector results, the size is the same as the input image
- blockSize: neighbor measure
- ksize: kernel size for the Sobel operator
- freeParameter: free parameter for Harris detector
- borderType: pixel extrapolation method

Examples of Harris corner detection results are as follows:



Complete documentation can be read at the following link:

https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html

2. Shi-Tomasi Corner Detection

Shi-Tomasi Corner Detection uses the basis that angles can be detected by looking for significant changes in any direction. For example, there is a small window in the image then it scans the entire image for angles. Shifting this small window in any direction will result in a large change in the appearance, if it is located at a corner.

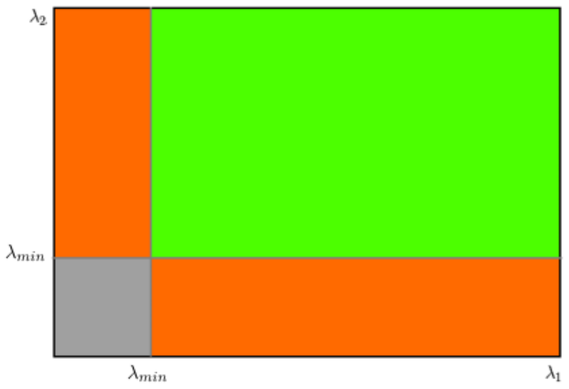
In the Harris method, the assessment of a point is said to be an angle or not done by giving an R value

$$R = det(M) - k(\mathrm{Tr}(M))^2$$

Shi-Tomasi came up with the idea of assessing R by simply determining the Eigenvalue of M alone

$$R = \min(\lambda_1, \lambda_2)$$

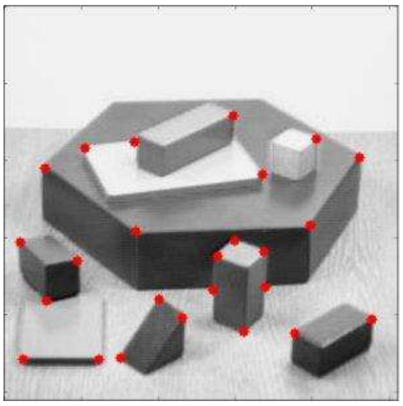
If the value is greater than the threshold value, it is considered an angle. If plotted in space λ1 - λ2, it will be like this:



From the figure it can be seen that only if $\lambda_1$ and $\lambda_2$ are above the minimum value, $\lambda_{min}$ is considered an angle (the area is green).

OpenCV has a function, cv.goodFeaturesToTrack (). This function will find the strongest N angles in the image using the Shi-Tomasi method. The input image must be a grayscale image. The settings / parameters of this method include: (a) the number of angles to be found, (b) the quality level, which is a value between 0-1, indicating the minimum quality of the angle, and (c) the minimum euclidean distance between detected angles . With these parameters, the function will find the angles in the image. All corners below the quality level will be rejected. The remaining corners will be sorted by quality in descending order. Then the function will take the strongest angle first, discard all the closest corners within the minimum distance range and return the N strongest angle.

Examples of Harris corner detection results are as follows:



Complete documentation can be read at the following link:

https://docs.opencv.org/master/d4/d8c/tutorial_py_shi_tomasi.html

## C.5 Grid Detection

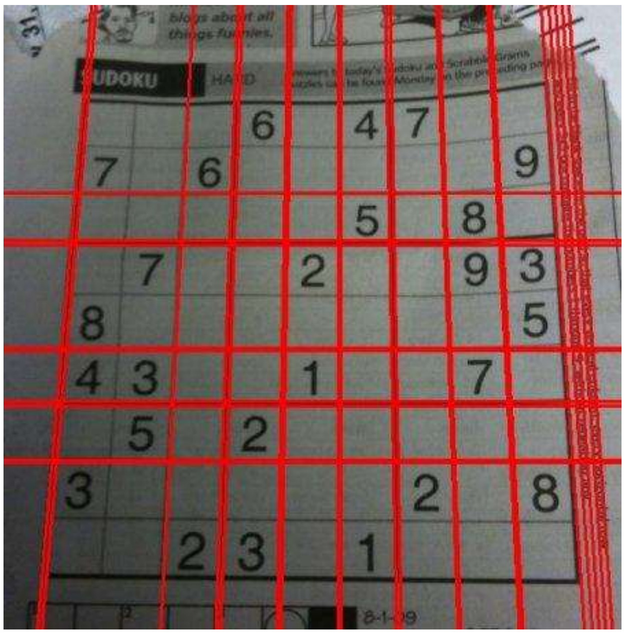
Grid detection can be done using the Hough Transform method. Hough Transform is a popular technique for detecting any shape, if it can be represented in mathematical form. Hough Transform can detect shapes even if they are damaged or slightly distorted.

The stages for conducting grid detection are as follows:

1. Implement Canny Edge Detection
2. Image edge dilation (Canny can find the two dividing edges on the grid as different edges, dilation can make these two edges join again)

3. Erosion (after the dilation process, the borders become thicker so Hough will detect many lines)
4. Implement HoughLines with the cv2.HoughLines () function in OpenCV
5. Join lines that are similar (similar)

An example of the results of the Hough Transform is as follows:



Complete documentation about HoughTransform can be read at the following link:

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html

## C.6 Contour Detection

Contours are closed curves that connect all continuous points having some color or intensity. Contours represent the shapes of objects found in the image. Contour detection is a useful technique for shape analysis as well as object detection and recognition. Contours are abstract collections of points and segments that match the shape of the objects in the image. We can manipulate the contours in our program such as counting the number of contours, using them to categorize object shapes, or cropping objects from the image (image segmentation).

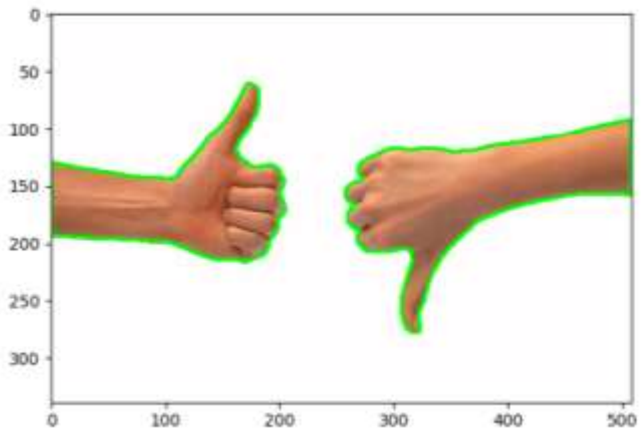
For better accuracy, here are the steps for detecting contours in an image:

1. Convert the input image to a binary image (can be obtained by implementing thresholding or canny edge detection).
2. Find contours using the findContours () function in OpenCV.
3. Draw a contour on the input image using the drawContours () function.

The findContours function accepts three parameters, namely:

- The first parameter is the Image which will be analyzed its contours
- The second parameter indicates the retrieval method. The retrieval method can be used to see the hierarchy of the contours in an image. Hierarchy denotes parent-child relationship. Parents is a shape / contour that is outside the child or in other words, parents are a shape / contour that surrounds the child.
- The third parameter shows the method used to process the shape approximation. The method can be the cv2.CHAIN_APPROX_SIMPLE method or the cv2.CHAIN_APPROX_NONE method. In the cv2.CHAIN_APPROX_SIMPLE flag, the class cv2.findContours will return the minimum number of coordinates representing contours while in the cv2.CHAIN_APPROX_SIMPLE flag the cv2.findContours class will return as many coordinates as possible representing the contours.

Examples of contour detection results are as follows:

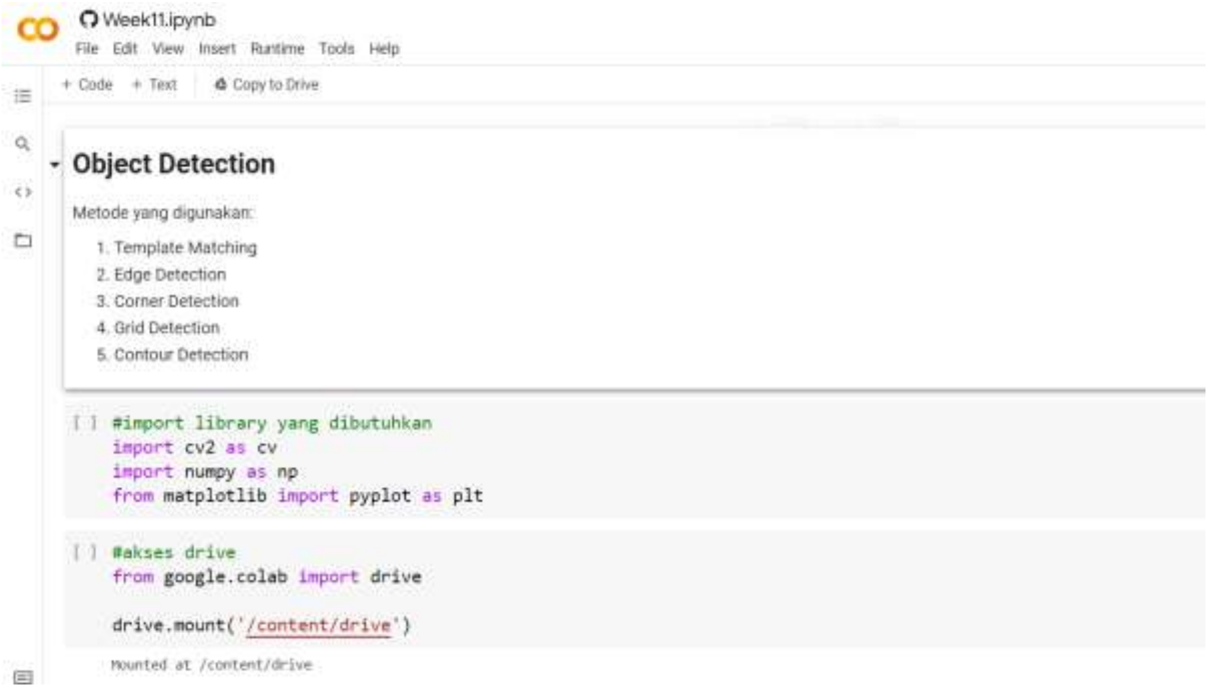


Complete documentation can be read at the following link:

https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html

## D. PRACTICUM

1. Go to https://colab.research.google.com/. After making sure that Google Colab is connected to your Github, create a new notebook and name it "Week11.ipynb". Then import some libraries and access the folders on your Drive as follows.



### Object Detection

Metode yang digunakan:

1. Template Matching
2. Edge Detection
3. Corner Detection
4. Grid Detection
5. Contour Detection

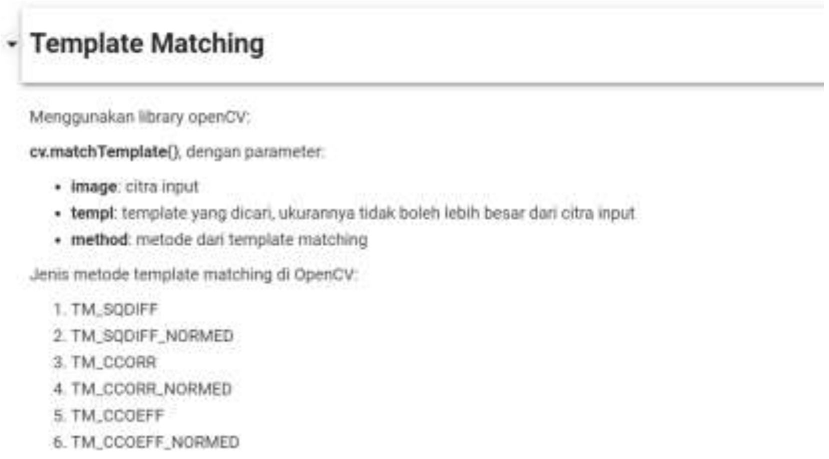
```
#import library yang dibutuhkan
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
```

```
#akses drive
from google.colab import drive

drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

2. Implement 6 template matching methods in OpenCV using the cats_and_bunnies.jpg and cat2_templatejpg.jpg images as templates.



### Template Matching

Menggunakan library openCV:

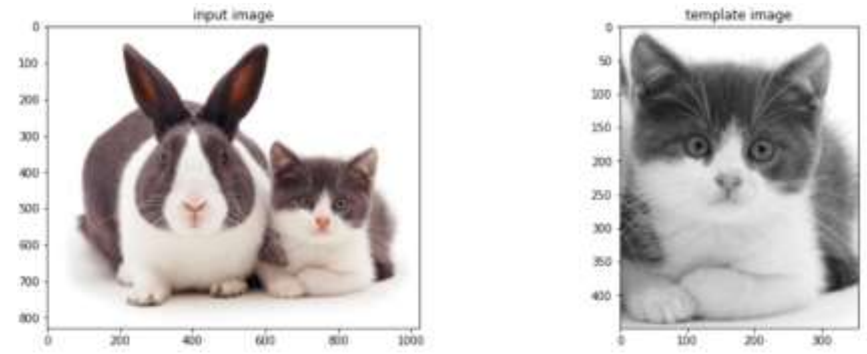
**cv.matchTemplate()**, dengan parameter:

- **image**: citra input
- **templ**: template yang dicari, ukurannya tidak boleh lebih besar dari citra input
- **method**: metode dari template matching

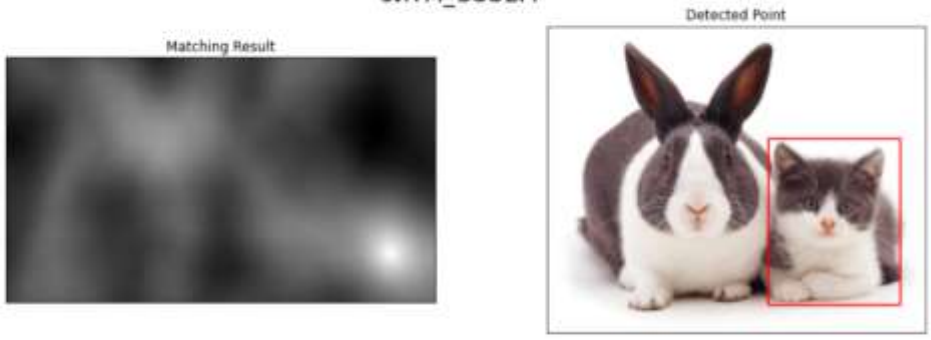
Jenis metode template matching di OpenCV:

1. TM_SQDIFF
2. TM_SQDIFF_NORMED
3. TM_CCORR
4. TM_CCORR_NORMED
5. TM_CCOEFF
6. TM_CCOEFF_NORMED

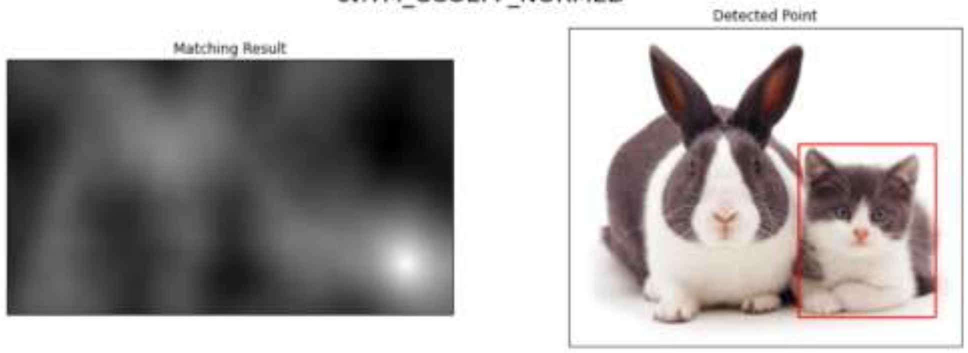
So as to produce the following output:



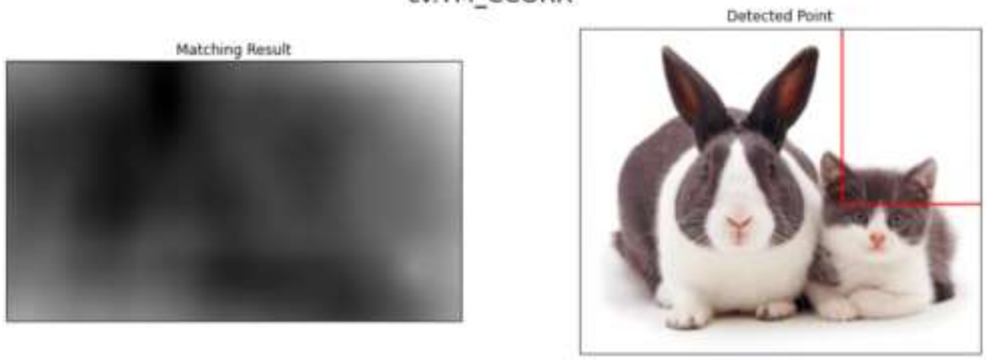
cv.TM_CCOEFF

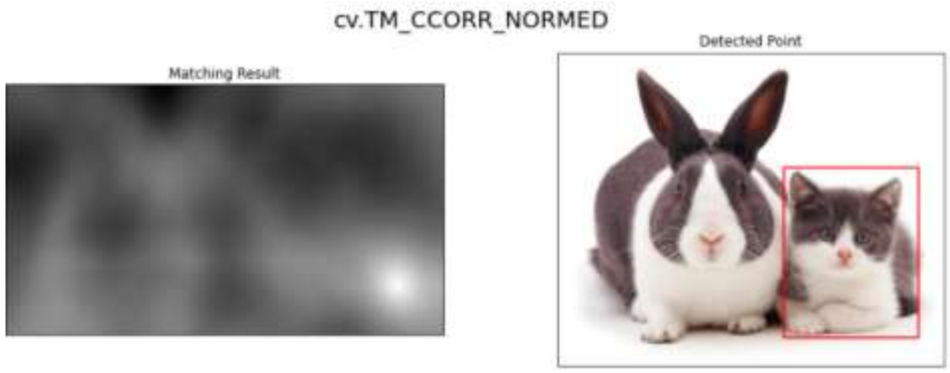


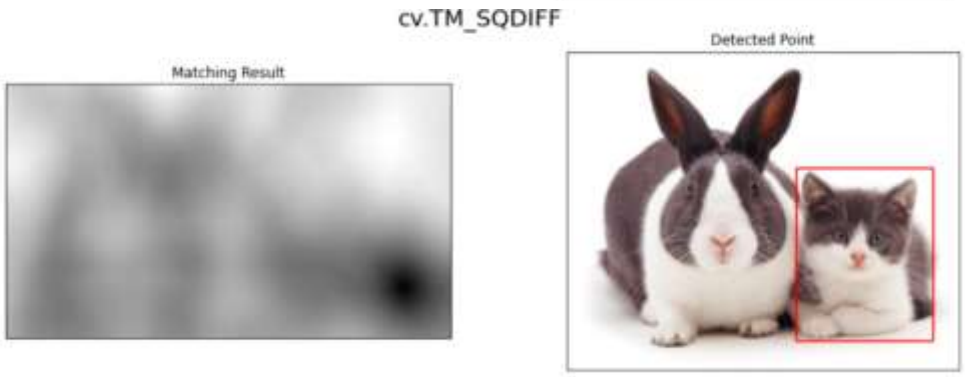
cv.TM_CCOEFF_NORMED

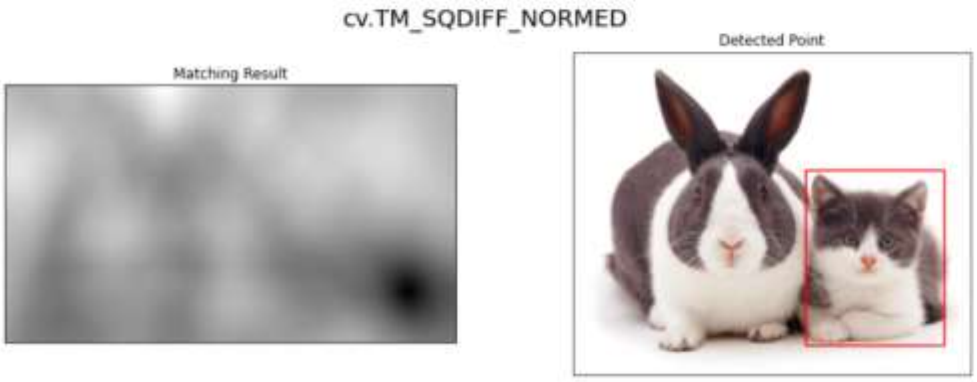


cv.TM_CCORR

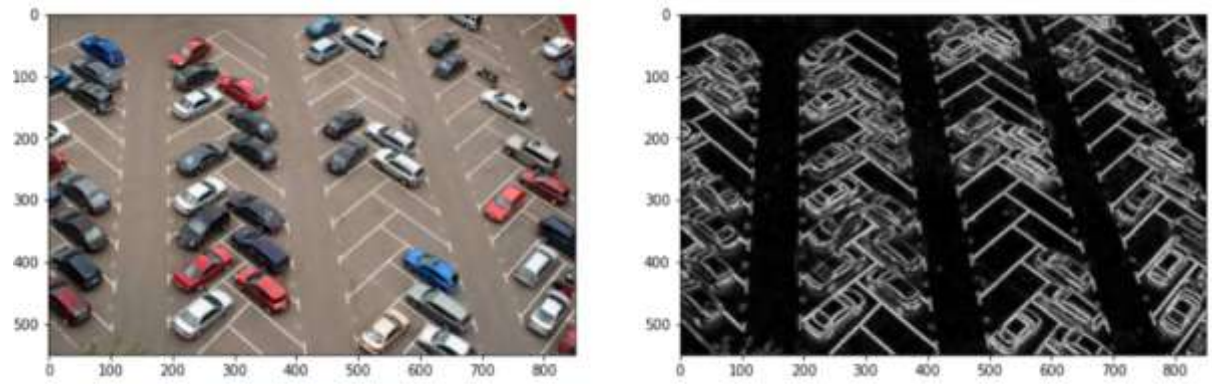




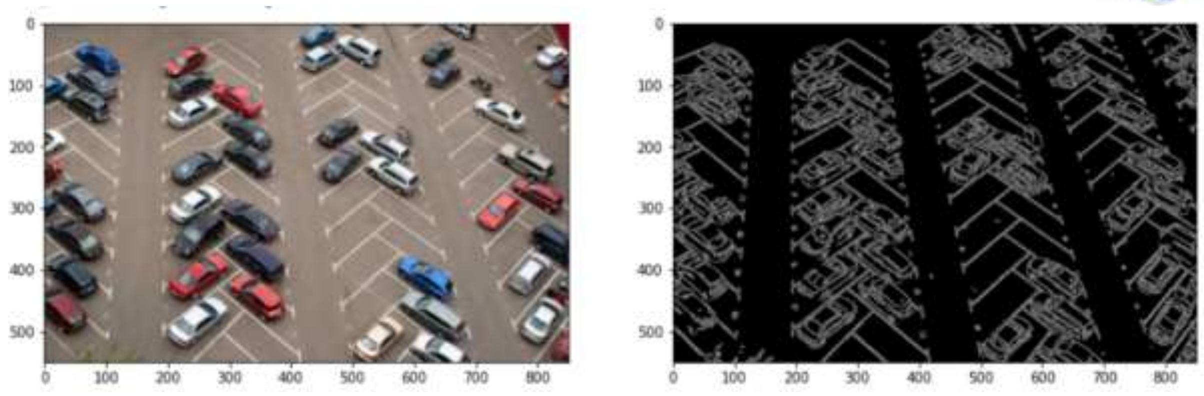




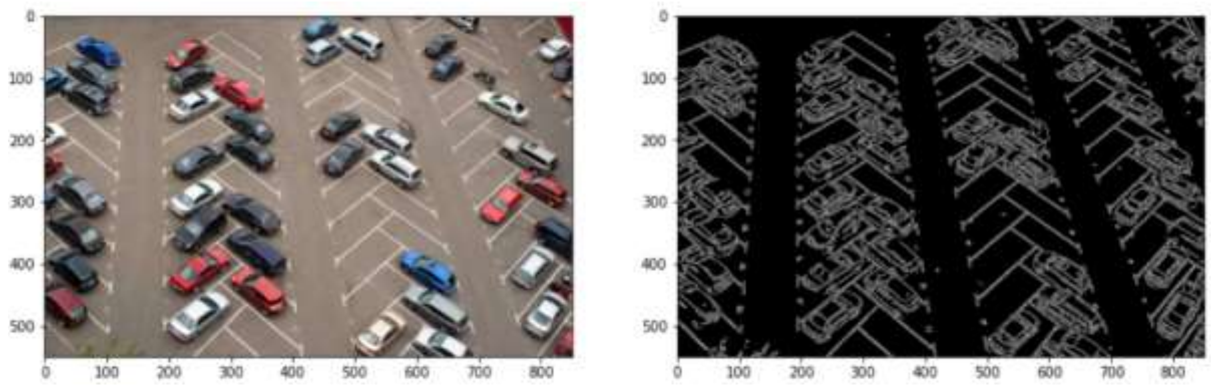
3. Implement the Sobel Edge Detection, Canny Edge Detection, and Laplacian Edge Detection methods in OpenCV using the parking-lot-cars.jpg image, resulting in the following output:
    a. Sobel Edge Detection



    b. Canny Edge Detection

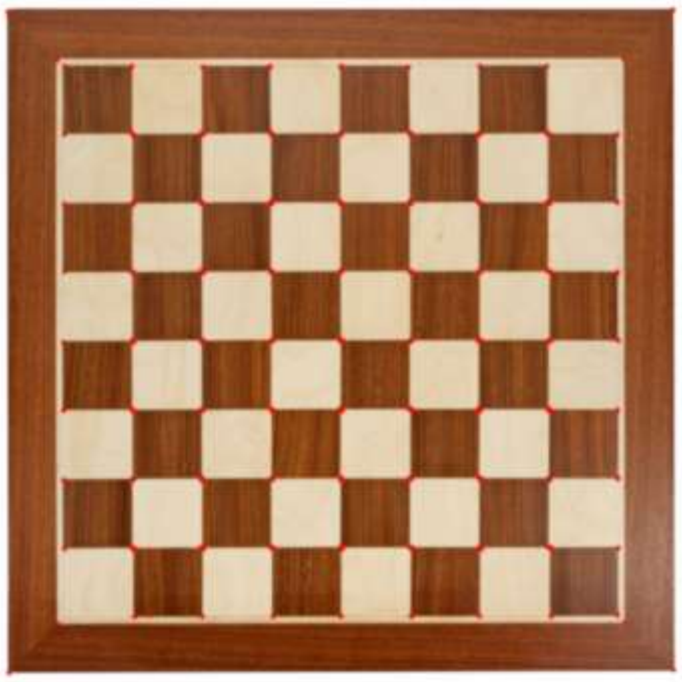


c. Laplacian Edge Detection

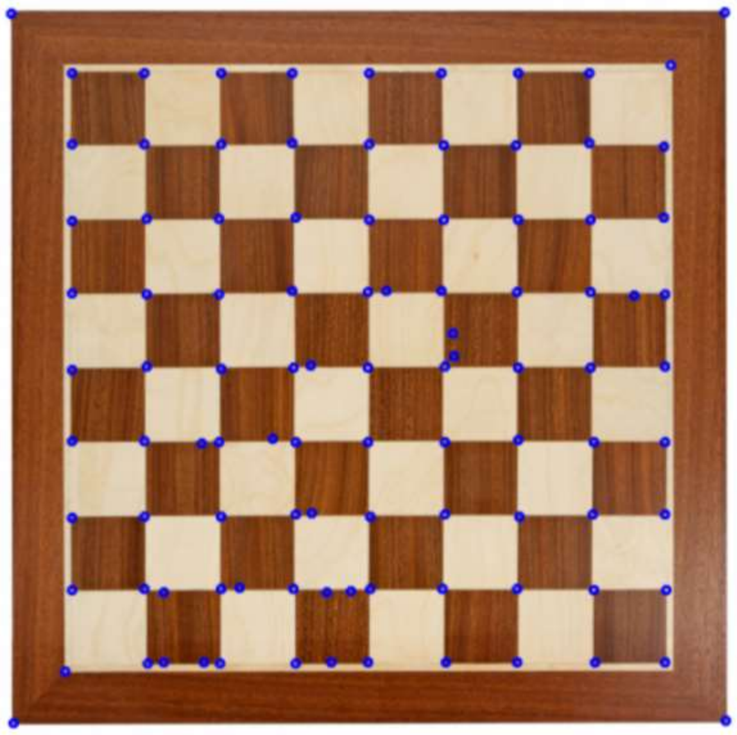


4. Implement the Sobel Edge Detection, Canny Edge Detection, and Laplacian Edge Detection methods in OpenCV using the parking-lot-cars.jpg image, resulting in the following output:

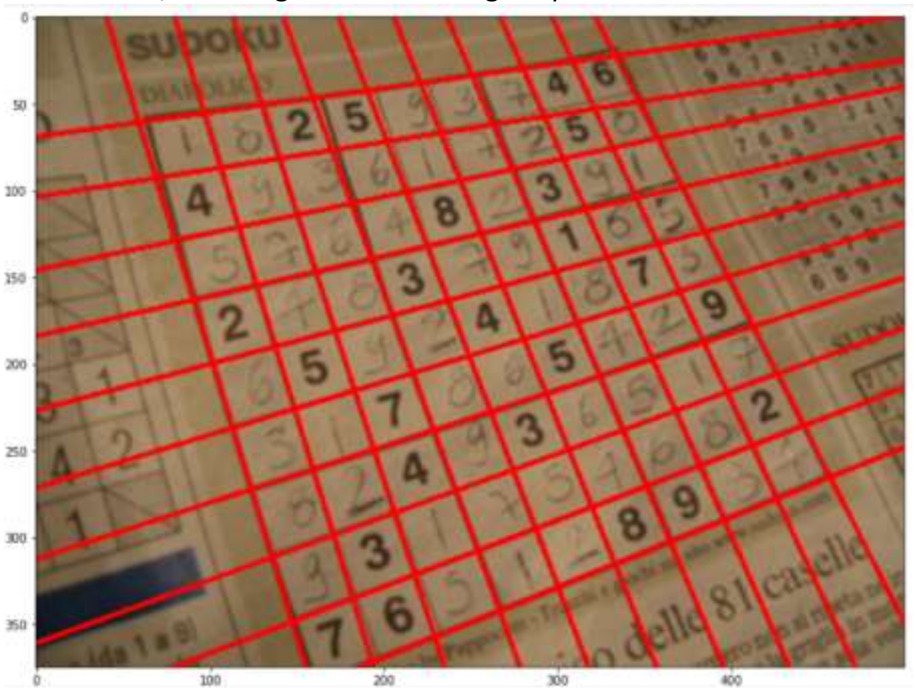
   a. Harris Corner Detection



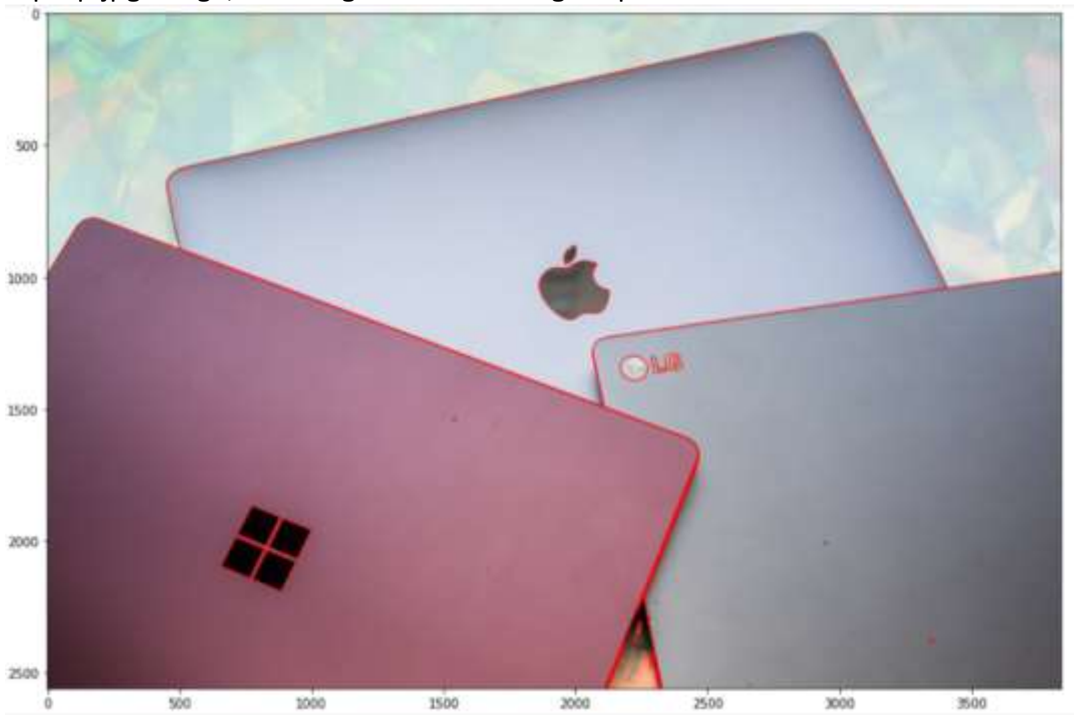
b. Shi-Tomasi Detection



5. Implement the Hough Transform method in OpenCV using the sudoku.jpg image. The stages of the grid detection process are in accordance with those contained in the theoretical review, resulting in the following output:



6. Implement the findContours () function in OpenCV for contour detection using the laptop.jpg image, resulting in the following output:



**--- Good Luck ---**