Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 018
Code Title:	Emerging Technologies in CpE 1 - Fundamentals of Computer Vision
1st Semester	AY 2024-2025

ACTIVITY 5.	**Line and Circle Detection**
Name	Marquez, Keith Leigh Zhen R.
Section	CPE32S3
Date Performed:	02/19/2025
Date Submitted:	02/21/2025
Instructor:	Engr. Roman M. Richard

1. Objectives

This activity aims to introduce students to openCV's APIs for Hough Transform.

2. Intended Learning Outcomes (ILOs)

After this activity, the students should be able to:

- Utilize openCV for circle and line detection.
- Analyze the use of hough Line and Circle function for finding objects in an image.

3. Procedures and Outputs

Detecting edges and contours are not only common and important tasks, they also constitute the basis for other complex operations. Lines and shape detection go hand in hand with edge and contour detection, so let's examine how OpenCV implements these.

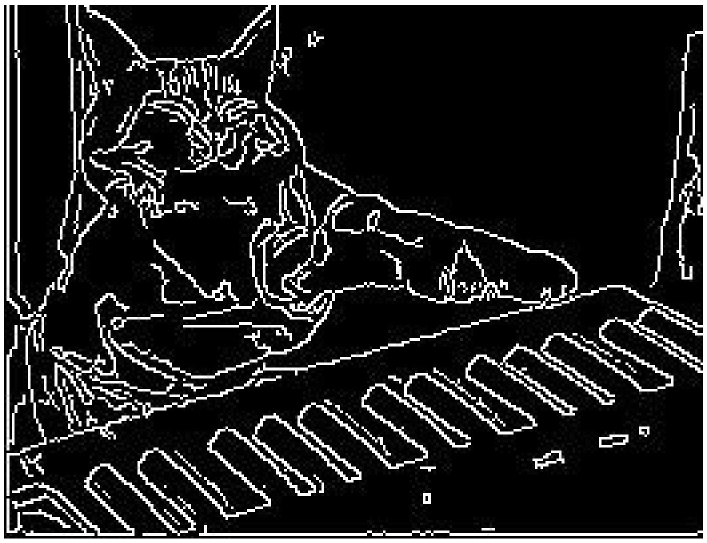
Line Detection

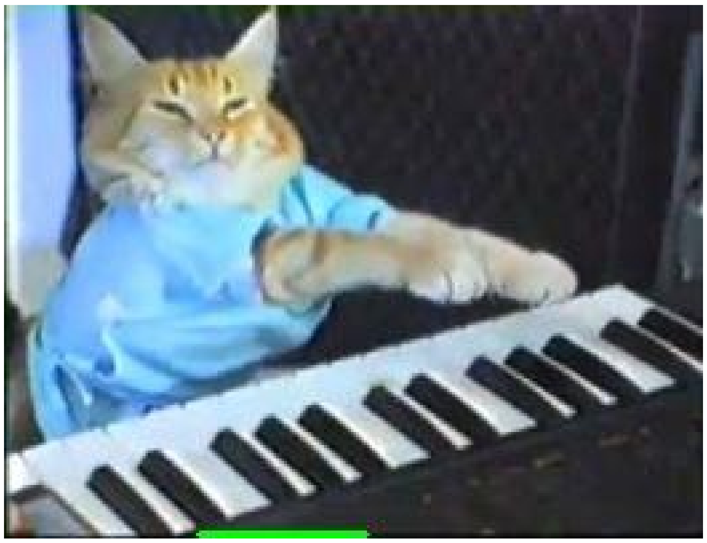
The theory behind lines and shape detection has its foundation in a technique called the Hough transform, invented by Richard Duda and Peter Hart, who extended (generalized) the work done by Paul Hough in the early 1960s.

Let's take a look at OpenCV's API for the Hough transforms.

```python
# Image source: https://en.wikipedia.org/wiki/Keyboard_Cat

from google.colab.patches import cv2_imshow
import cv2
import numpy as np

img = cv2.imread('/content/Keyboard_cat.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

```
edges = cv2.Canny(gray,50,120)
minLineLength = 20
maxLineGap = 5
lines = cv2.HoughLinesP(edges,1,np.pi/180,100,minLineLength,
        maxLineGap)
for x1,y1,x2,y2 in lines[0]:
  cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)

cv2_imshow(edges)
cv2_imshow(img)
```





The crucial point of this simple script —aside from the HoughLines function call— is the setting of minimum line length (shorter lines will be discarded) and the maximum line gap, which is the maximum size of a gap in a line before the two segments start being considered as separate lines.

Also note that the HoughLines function takes a single channel binary image, processed through the Canny edge detection filter. Canny is not a strict requirement, however; an image that's been denoised and only represents edges, is the ideal source for a Hough transform, so you will find this to be a common practice.

The parameters of HoughLinesP are as follows:

- The image we want to process.
- The geometrical representations of the lines, rho and theta, which are usually 1 and np.pi/180.
- The threshold, which represents the threshold below which a line is discarded. The Hough transform works with a system of bins and votes, with each bin representing a line, so any line with a minimum of the votes is retained, the rest discarded.
- MinLineLength and MaxLineGap, which we mentioned previously

Questions:

1. Which line of code is responsible for setting the minimum line length?
- minLineLength = 20
1. What is the mathematical formula for Hough transform and explain how it finds lines.
- $\rho = x * \cos(\theta) + y * \sin(\theta)$ Here, ρ is the distance from the origin to the line, and θ is the angle the line makes. Each point in the image "votes" for all the possible lines that could pass through it. These votes are stored in a table, and the more votes a particular (ρ,θ) pair gets, the more likely it is a real line in the image. In the end, the algorithm picks the most voted lines and draws them back onto the image. This makes it great at detecting lines, even when the image is messy or has missing parts.

Circle Detection

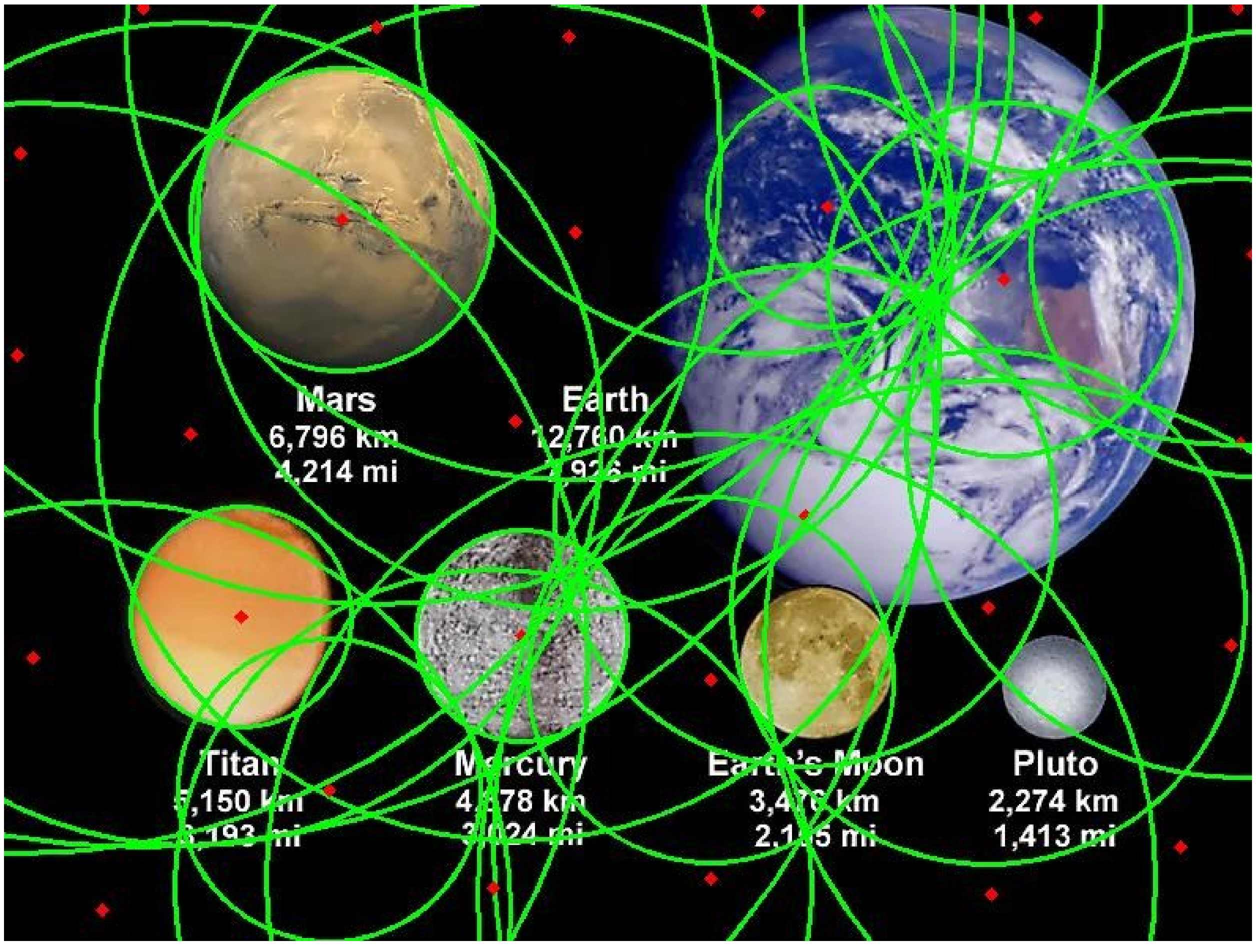
OpenCV also has a function for detecting circles, called HoughCircles. It works in a very similar fashion to HoughLines, but where minLineLength and maxLineGap were the parameters to discard or retain lines, HoughCircles has a minimum distance between circles' centers, minimum, and maximum radius of the circles. Here's the obligatory example:

Before going into the sample code, check first: **What is the HoughCircles function and what are its parameters?**

The HoughCircles function in OpenCV detects circles in an image using the Hough Transform. It works by identifying circular patterns based on edge detection. The function takes parameters like dp (resolution scale), minDist (minimum distance between circles), param1 (Canny edge threshold), param2 (circle detection threshold), and minRadius/maxRadius (circle size limits). It is useful for detecting objects like coins, bubbles, or eyes in an image.

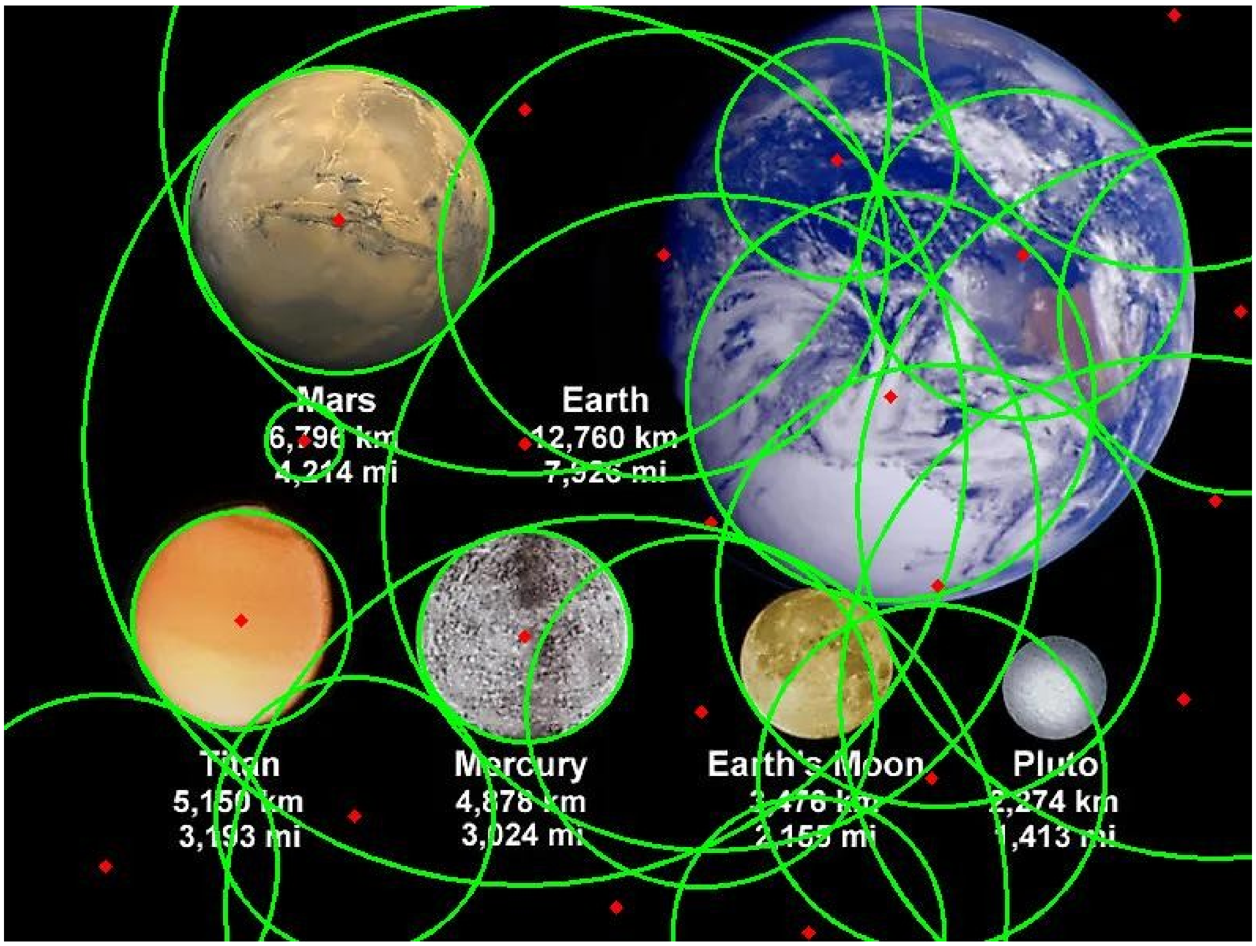
```python
import cv2
import numpy as np

# Our testing value
n = 5

planets = cv2.imread('/content/planets.webp')
```

```
gray_img = cv2.cvtColor(planets, cv2.COLOR_BGR2GRAY)
img = cv2.medianBlur(gray_img, n) # We will change this value passed
as parameter and observe results
cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,120,
                            param1=100,param2=30,minRadius=0,
                                maxRadius=0)
circles = np.uint16(np.around(circles))

for i in circles[0,:]:
  # draw the outer circle
  cv2.circle(planets,(i[0],i[1]),i[2],(0,255,0),2)
  # draw the center of the circle
  cv2.circle(planets,(i[0],i[1]),2,(0,0,255),3)

cv2.imwrite("planets_circles.jpg", planets)
cv2_imshow(planets)
```



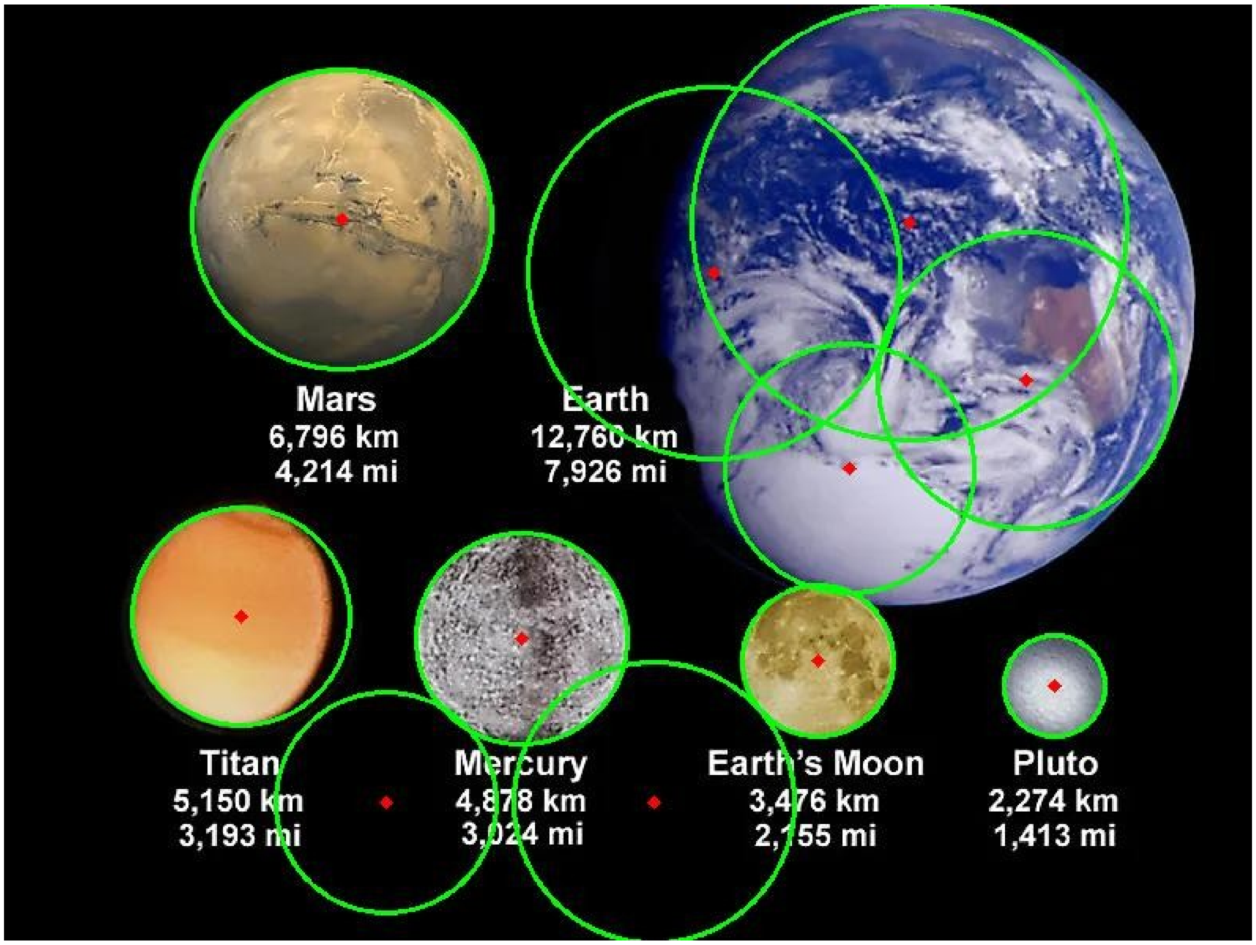
What happens to the code once you run **and the value of n is 5**?

```
import cv2
import numpy as np

# Our testing value
n = 9

planets = cv2.imread('/content/planets.webp')
gray_img = cv2.cvtColor(planets, cv2.COLOR_BGR2GRAY)
img = cv2.medianBlur(gray_img, n) # We will change this value passed
as parameter and observe results
cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,120,
                            param1=100,param2=30,minRadius=0,
                                maxRadius=0)
circles = np.uint16(np.around(circles))

for i in circles[0,:]:
  # draw the outer circle
  cv2.circle(planets,(i[0],i[1]),i[2],(0,255,0),2)
  # draw the center of the circle
  cv2.circle(planets,(i[0],i[1]),2,(0,0,255),3)

cv2.imwrite("planets_circles.jpg", planets)
cv2_imshow(planets)
```



Change the value to 9, **what happens to the image**?

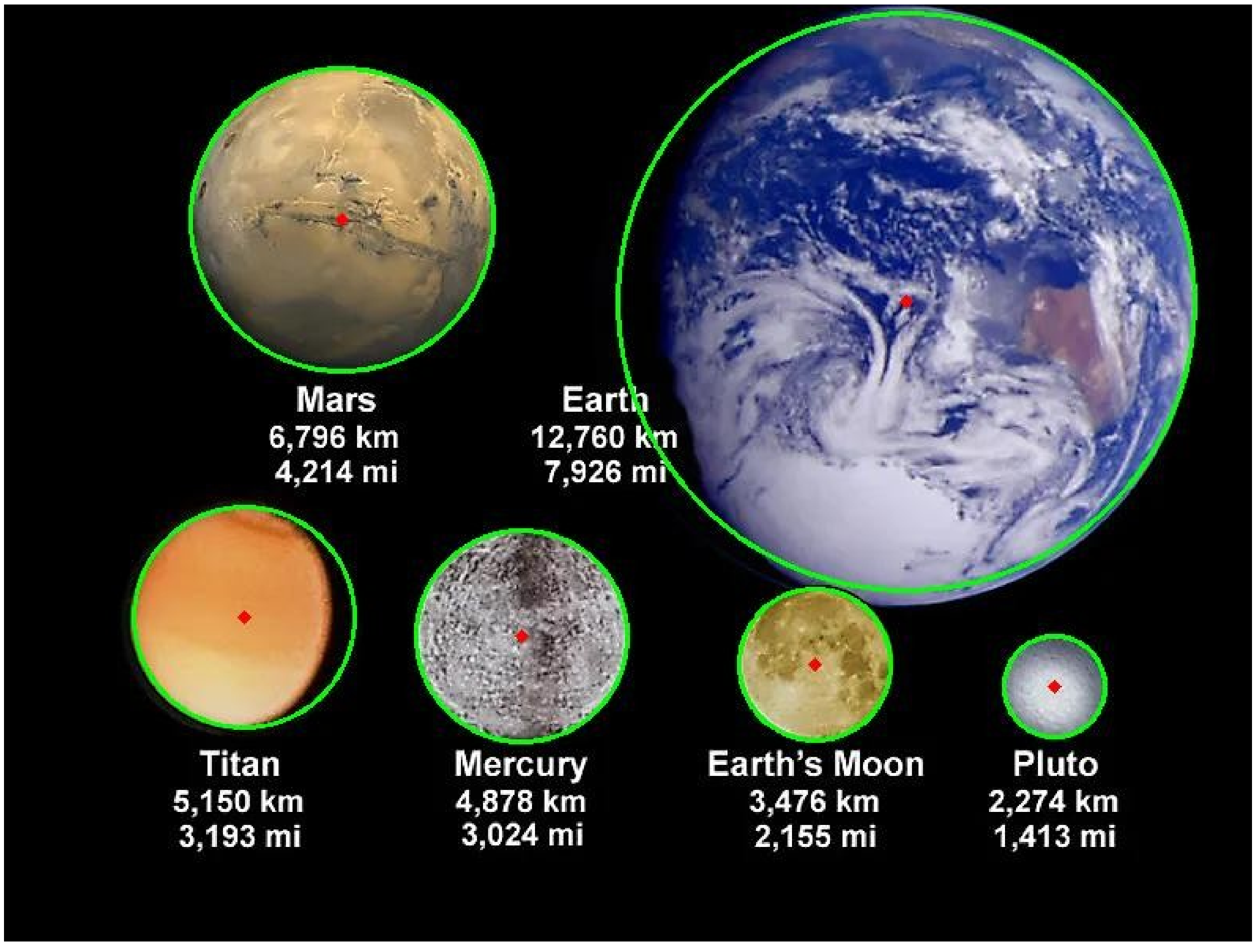
```
import cv2
import numpy as np

# Our testing value
n = 15

planets = cv2.imread('/content/planets.webp')
gray_img = cv2.cvtColor(planets, cv2.COLOR_BGR2GRAY)
img = cv2.medianBlur(gray_img, n) # We will change this value passed
as parameter and observe results
cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,120,
                            param1=100,param2=30,minRadius=0,
                                maxRadius=0)
circles = np.uint16(np.around(circles))

for i in circles[0,:]:
  # draw the outer circle
  cv2.circle(planets,(i[0],i[1]),i[2],(0,255,0),2)
  # draw the center of the circle
  cv2.circle(planets,(i[0],i[1]),2,(0,0,255),3)

cv2.imwrite("planets_circles.jpg", planets)
cv2_imshow(planets)
```



Lastly, change the value to 15, **what can you say about the resulting image?**

```
import cv2
import numpy as np

# Our testing value
n = 21

planets = cv2.imread('/content/planets.webp')
gray_img = cv2.cvtColor(planets, cv2.COLOR_BGR2GRAY)
img = cv2.medianBlur(gray_img, n) # We will change this value passed
as parameter and observe results
cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,120,
                            param1=100,param2=30,minRadius=0,
                                maxRadius=0)
```

```
circles = np.uint16(np.around(circles))

for i in circles[0,:]:
  # draw the outer circle
  cv2.circle(planets,(i[0],i[1]),i[2],(0,255,0),2)
  # draw the center of the circle
  cv2.circle(planets,(i[0],i[1]),2,(0,0,255),3)

cv2.imwrite("planets_circles.jpg", planets)
cv2_imshow(planets)
```

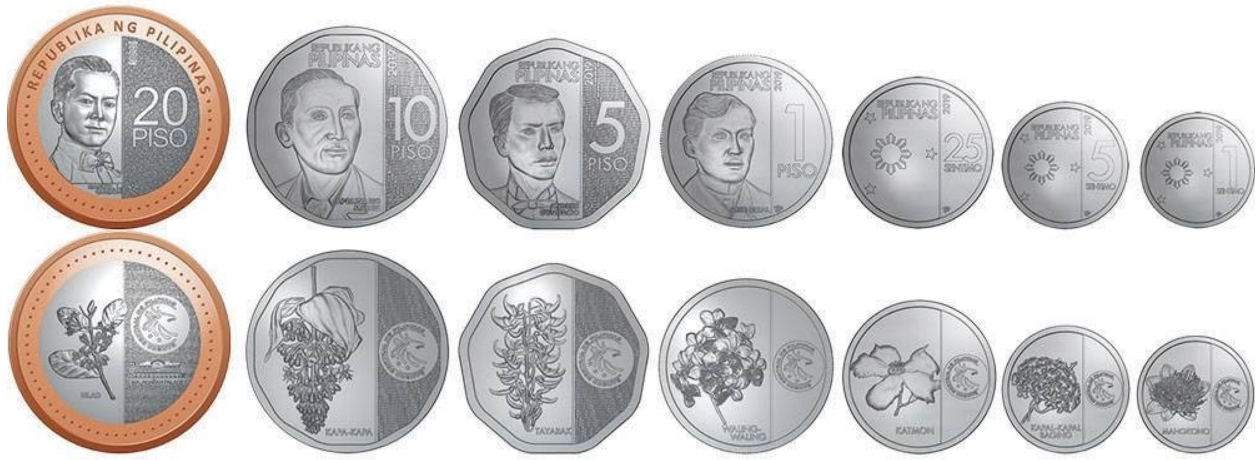


Provide an analysis of the output so far. How does the code help the changes in the resulting image?

Increasing n enhances blurring, reducing noise and improving detection of larger circles. However, too much blurring can erase details, making smaller circles harder to detect. The best n value depends on the image and needs some trial and error.

4. Supplementary Activity

The attached image contains coins used in the Philippines.



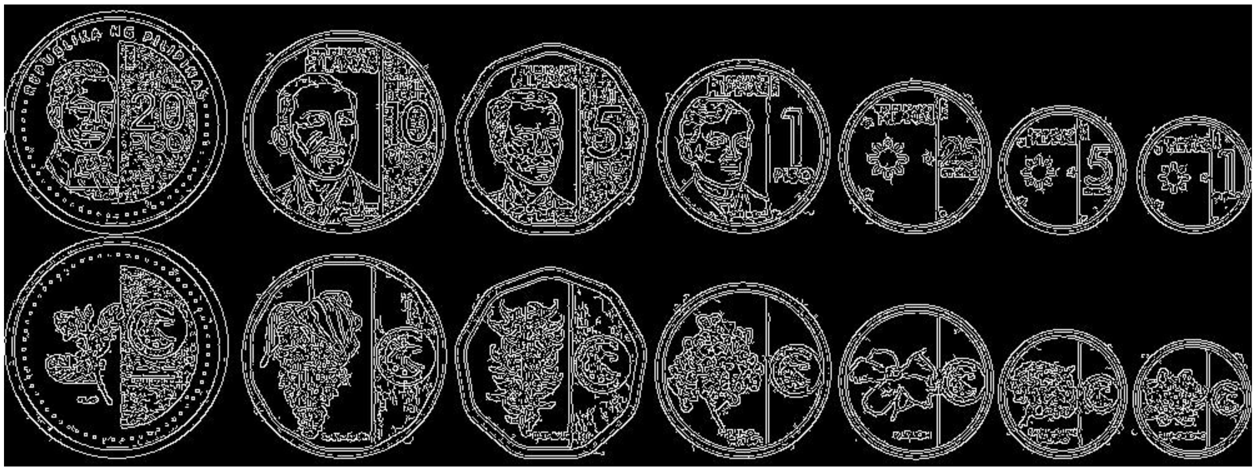
Your job is to count the amount of coins (denomation not included, no sum of prices; just the amount of coins present) through either line detection or circle detection.

- Create a function using line detection and pass this image as parameter, what is the output? Can you use houghlines to count circles?
- Create a function using circle detection and pass this image as parameter, show the output? Can you use houghcircles to count the circles?

Create a function using line detection and pass this image as parameter, what is the output?

Output

```
img_coins = cv2.igray = cv2.cvtColor(img_coins,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,120)
mread('/content/coins.jpg')
minLineLength = 20
maxLineGap = 5
lines = cv2.HoughLinesP(edges,1,np.pi/180,100,minLineLength,
        maxLineGap)
for x1,y1,x2,y2 in lines[0]:
  cv2.line(img_coins,(x1,y1),(x2,y2),(0,255,0),2)

cv2_imshow(edges)
```



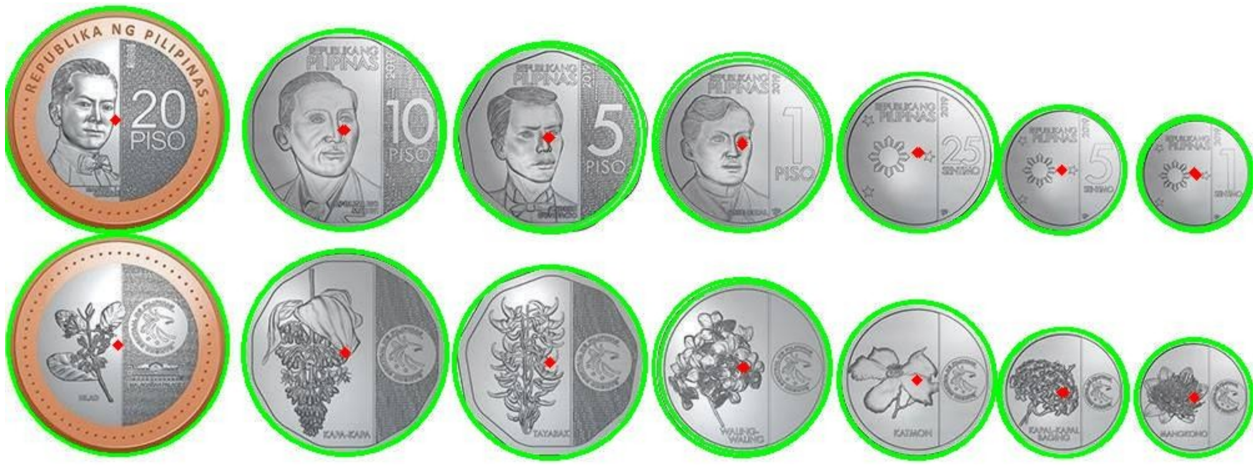
Can you use houghlines to count circles?

No, HoughLines and HoughLinesP are specifically designed for detecting lines in images. They cannot be used directly to count circles.

Create a function using circle detection and pass this image as parameter, show the output?

Output:

```
n = 23

ph_coins = cv2.imread('/content/coins.jpg')
gray_img = cv2.cvtColor(ph_coins, cv2.COLOR_BGR2GRAY)
img = cv2.medianBlur(gray_img, n) # We will change this value passed
as parameter and observe results
cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,100,
                            param1=100,param2=30,minRadius=0,
                                maxRadius=0)
circles = np.uint16(np.around(circles))

for i in circles[0,:]:
  # draw the outer circle
  cv2.circle(ph_coins,(i[0],i[1]),i[2],(0,255,0),2)
  # draw the center of the circle
  cv2.circle(ph_coins,(i[0],i[1]),2,(0,0,255),3)

cv2.imwrite("Coins.jpg", ph_coins)
cv2_imshow(ph_coins)
```



```
num_circles = len(circles[0])
print("Number of circles detected:", num_circles)

Number of circles detected: 14
```

Can you use houghcircles to count the circles?

Yes, you can use HoughCircles to count the circles in an image.

5. Summary, Conclusions and Lessons Learned

In this activity, I explored how to use OpenCV's Hough Transform to detect lines and circles in images. By adjusting the parameters of HoughLinesP and HoughCircles, I realized how important it is to set a right value for minimum line length, maximum line gap, and blur level for better accuracy. Applying medianBlur helped reduce noise and improved the detection of larger circles, but too much blurring made smaller circles harder to find. I also learned that finding the right parameter values requires experimentation and careful analysis of the results.