Project presentation

Team – 5

TEAM STRANGE

Team Members

Aditya Modi(220071)(Leader)
Akash Paijwar(220096)
Chiranshu Kataria(220315)
Himanshu(220452)
Khushi Sahu(220532)
Naman(220683)
Pavi Agarwal(220762)
Pubali Banerjee(220831)

Fourier Transform

The Fourier Transform is a mathematical tool that allows us to analyze and understand signals and functions in terms of their frequency components.

Brief:

- Fourier Transform decomposes a signal or function into a sum of sinusoidal functions, revealing the underlying frequencies.
- The transformation converts a function from the time (or spatial) domain to the frequency domain, providing a new perspective for analysis.

Mathmatical formula to calculate fourier transform

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(j\omega)e^{j\omega t}d\omega$$

$$X(j\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt$$

Applications

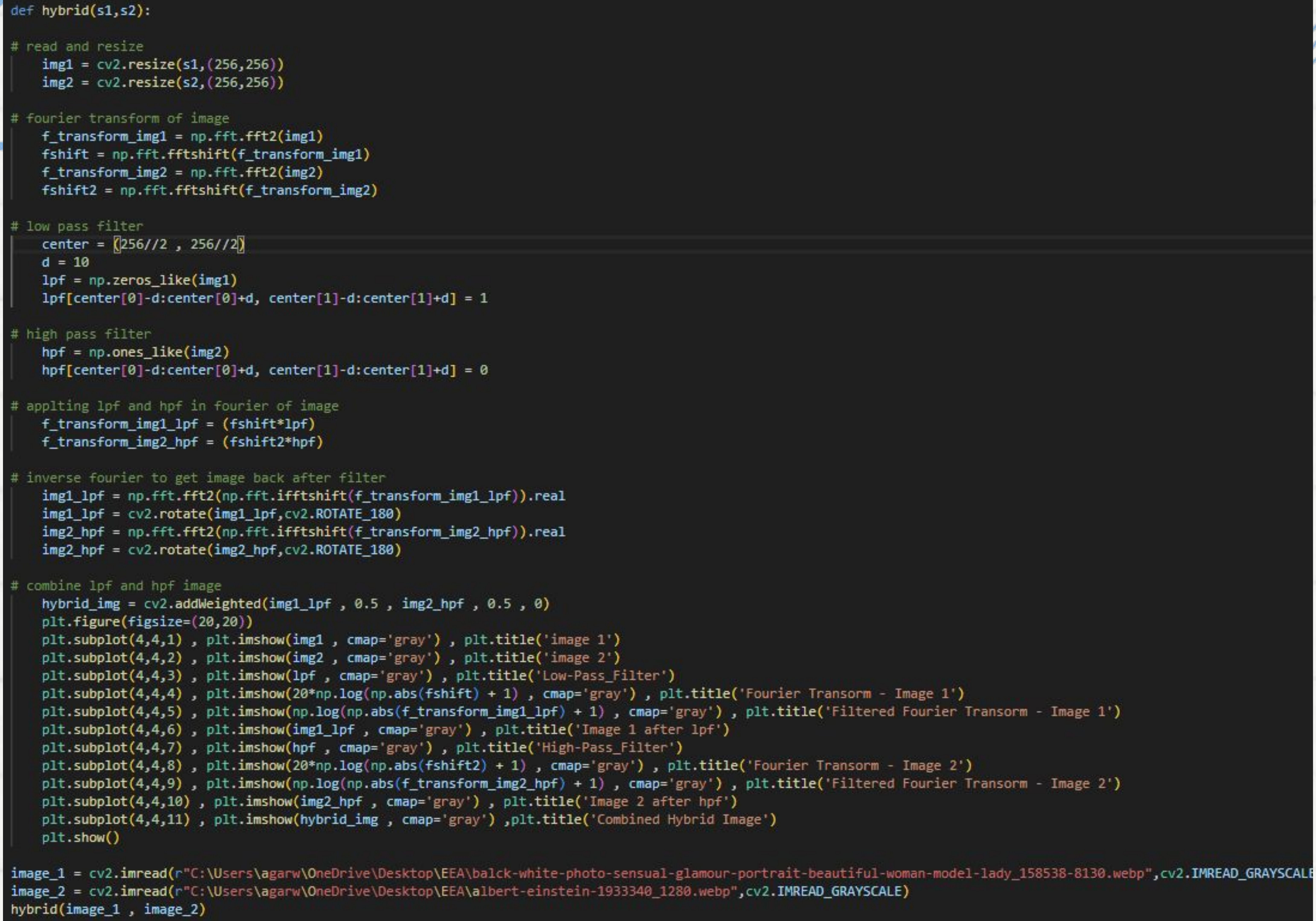
- Signal Processing
- Image Processing
- Communication Systems
- Physics
- Seismology

Hybrid Image

Hybrid images combine the low-frequency information from one image with the high-frequency information from another, creating a unique visual illusion.

Creating Hybrid Images:

- Concept: The creation of hybrid images involves the strategic combination of low-pass and high-pass filters.
- Low-Pass Filter: Used to extract low-frequency components from one image, preserving overall structure and form.
- High-Pass Filter: Extracts high-frequency components from another image, capturing fine details and textures.



```
def hybrid(s1,s2):

# read and resize
    img1 = cv2.resize(s1,(256,256))
    img2 = cv2.resize(s2,(256,256))

# fourier transform of image
    f_transform_img1 = np.fft.fft2(img1)
    fshift = np.fft.fftshift(f_transform_img1)
    f_transform_img2 = np.fft.fft2(img2)
    fshift2 = np.fft.fftshift(f_transform_img2)

# low pass filter
    center = (256//2 , 256//2)
    d = 10
    lpf = np.zeros_like(img1)
    lpf[center[0]-d:center[0]+d, center[1]-d:center[1]+d] = 1

# high pass filter
    hpf = np.ones_like(img2)
    hpf[center[0]-d:center[0]+d, center[1]-d:center[1]+d] = 0

# applting lpf and hpf in fourier of image
    f_transform_img1_lpf = (fshift*lpf)
    f_transform_img2_hpf = (fshift2*hpf)

# inverse fourier to get image back after filter
    img1_lpf = np.fft.fft2(np.fft.ifftshift(f_transform_img1_lpf)).real
    img1_lpf = cv2.rotate(img1_lpf,cv2.ROTATE_180)
    img2_hpf = np.fft.fft2(np.fft.ifftshift(f_transform_img2_hpf)).real
    img2_hpf = cv2.rotate(img2_hpf,cv2.ROTATE_180)

# combine lpf and hpf image
    hybrid_img = cv2.addWeighted(img1_lpf , 0.5 , img2_hpf , 0.5 , 0)
    plt.figure(figsize=(20,20))
    plt.subplot(4,4,1) , plt.imshow(img1 , cmap='gray') , plt.title('image 1')
    plt.subplot(4,4,2) , plt.imshow(img2 , cmap='gray') , plt.title('image 2')
    plt.subplot(4,4,3) , plt.imshow(lpf , cmap='gray') , plt.title('Low-Pass_Filter')
    plt.subplot(4,4,4) , plt.imshow(20*np.log(np.abs(fshift) + 1) , cmap='gray') , plt.title('Fourier Transorm - Image 1')
    plt.subplot(4,4,5) , plt.imshow(np.log(np.abs(f_transform_img1_lpf) + 1) , cmap='gray') , plt.title('Filtered Fourier Transorm - Image 1')
    plt.subplot(4,4,6) , plt.imshow(img1_lpf , cmap='gray') , plt.title('Image 1 after lpf')
    plt.subplot(4,4,7) , plt.imshow(hpf , cmap='gray') , plt.title('High-Pass_Filter')
    plt.subplot(4,4,8) , plt.imshow(20*np.log(np.abs(fshift2) + 1) , cmap='gray') , plt.title('Fourier Transorm - Image 2')
    plt.subplot(4,4,9) , plt.imshow(np.log(np.abs(f_transform_img2_hpf) + 1) , cmap='gray') , plt.title('Filtered Fourier Transorm - Image 2')
    plt.subplot(4,4,10) , plt.imshow(img2_hpf , cmap='gray') , plt.title('Image 2 after hpf')
    plt.subplot(4,4,11) , plt.imshow(hybrid_img , cmap='gray') ,plt.title('Combined Hybrid Image')
    plt.show()

image_1 = cv2.imread(r"C:\Users\agarw\OneDrive\Desktop\EEA\balck-white-photo-sensual-glamour-portrait-beautiful-woman-model-lady_158538-8130.webp",cv2.IMREAD_GRAYSCALE
image_2 = cv2.imread(r"C:\Users\agarw\OneDrive\Desktop\EEA\albert-einstein-1933340_1280.webp",cv2.IMREAD_GRAYSCALE)
hybrid(image_1 , image_2)
```

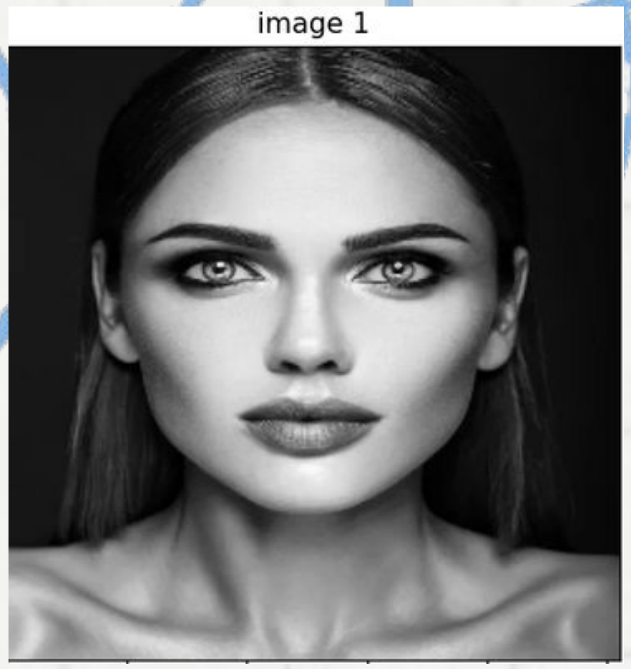
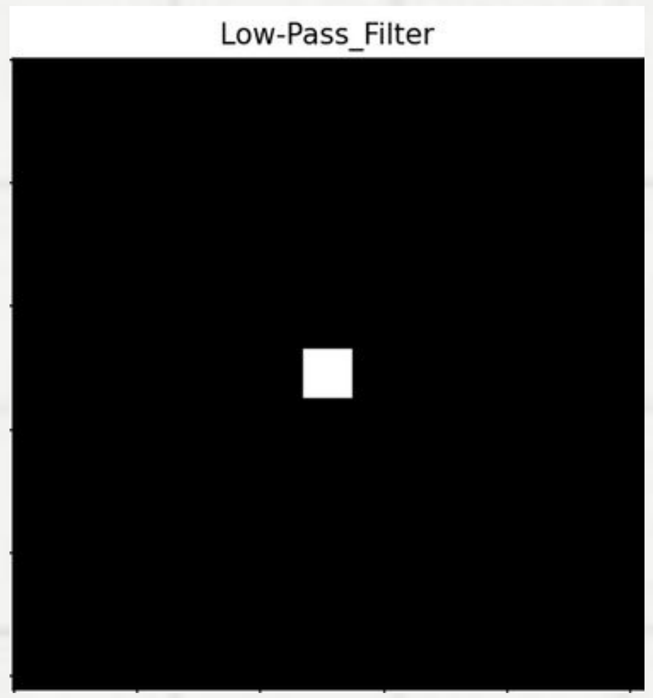
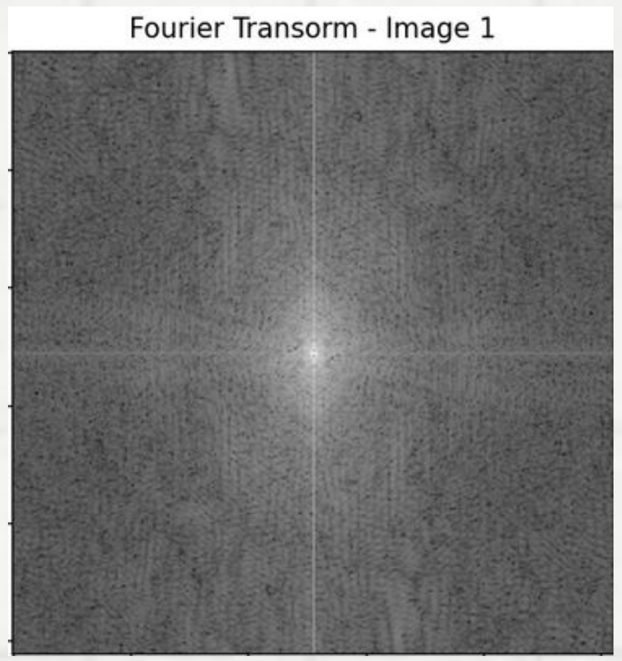
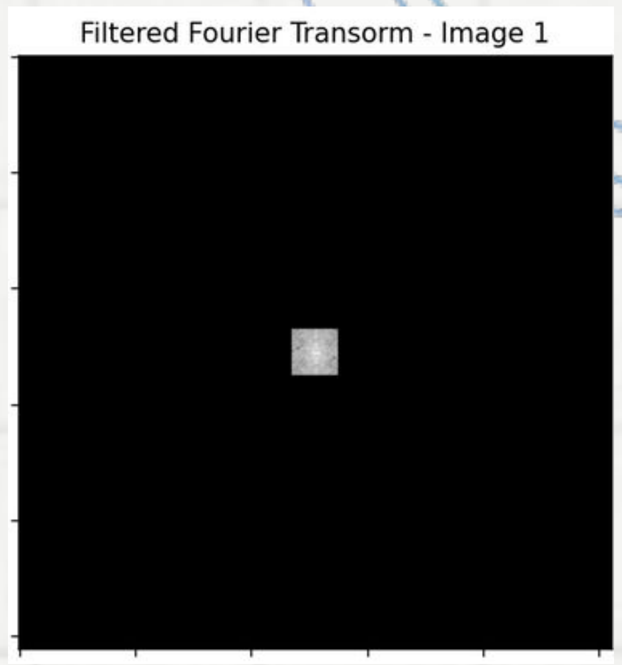


image 1


Low-Pass_Filter


Fourier Transorm - Image 1


Filtered Fourier Transorm - Image 1

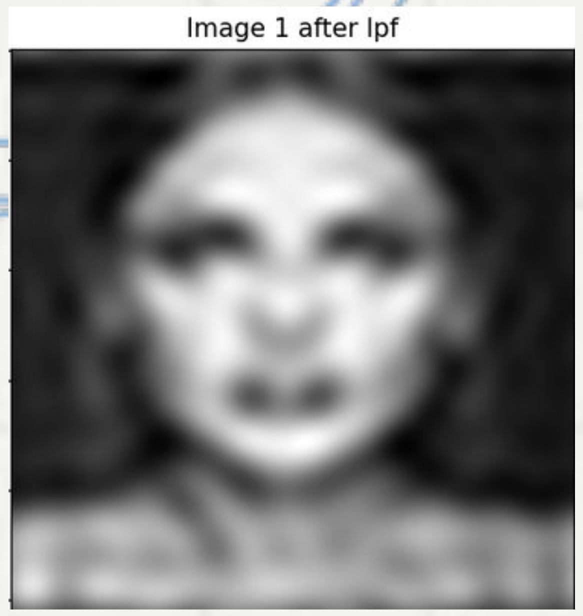


Image 1 after lpf

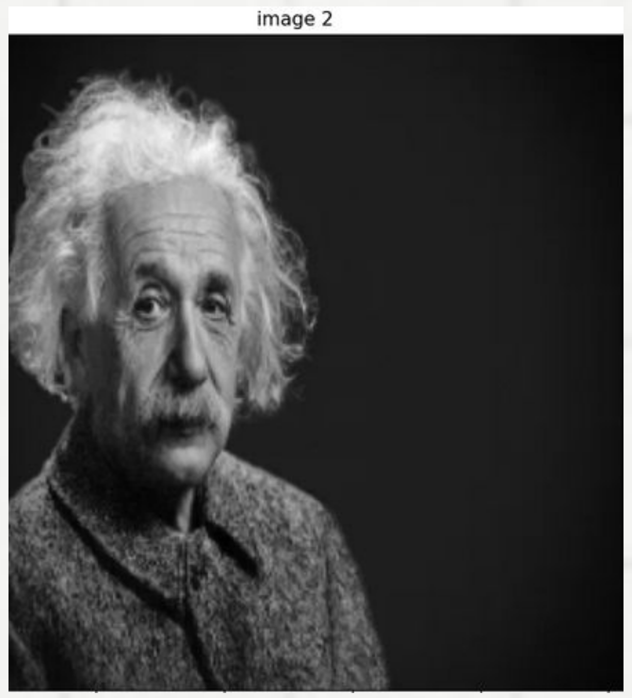
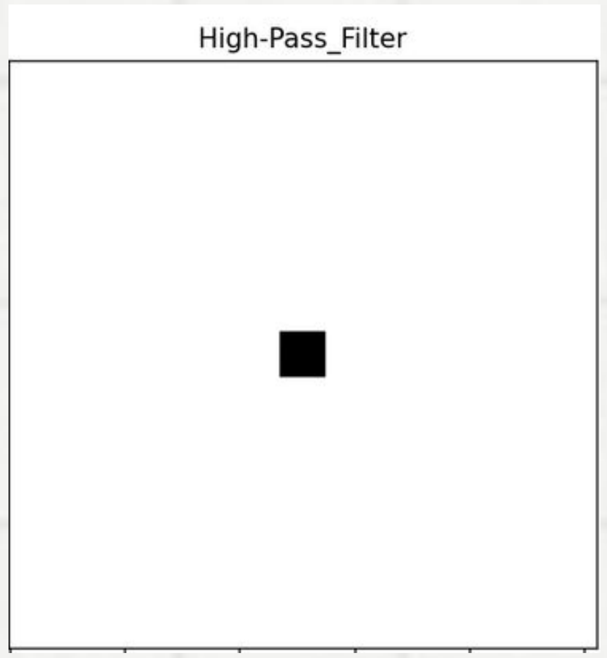
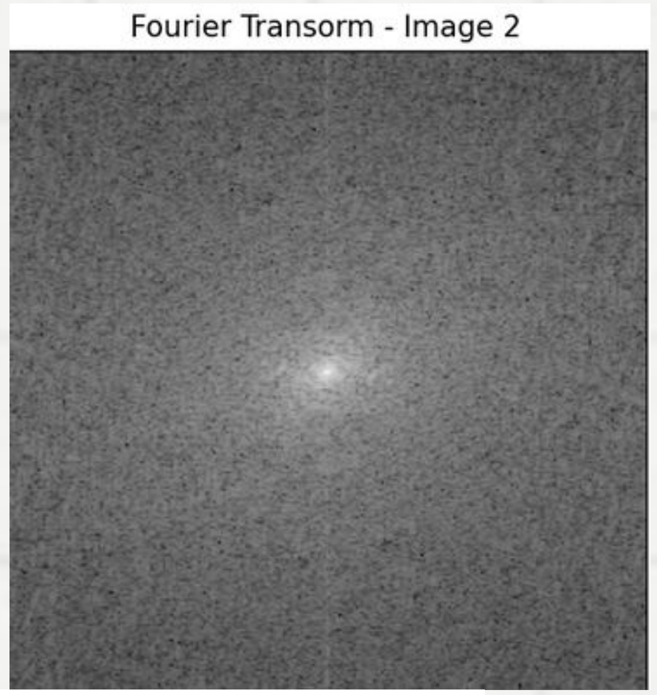
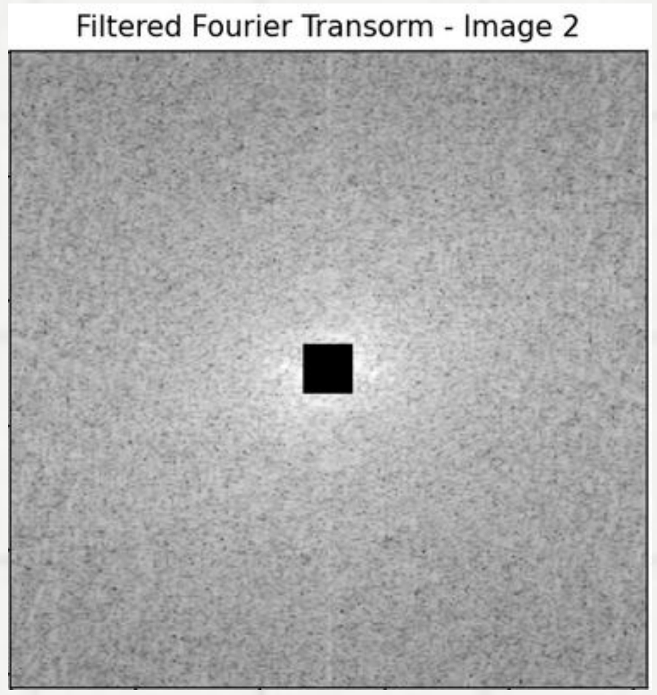


image 2


High-Pass_Filter


Fourier Transorm - Image 2


Filtered Fourier Transorm - Image 2

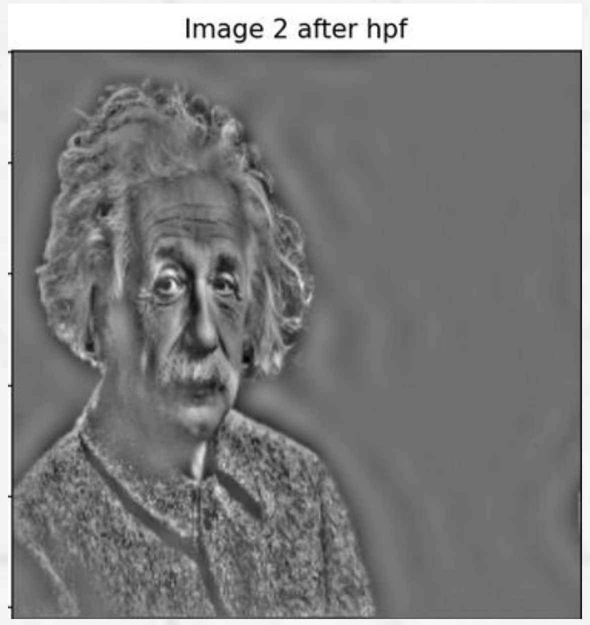
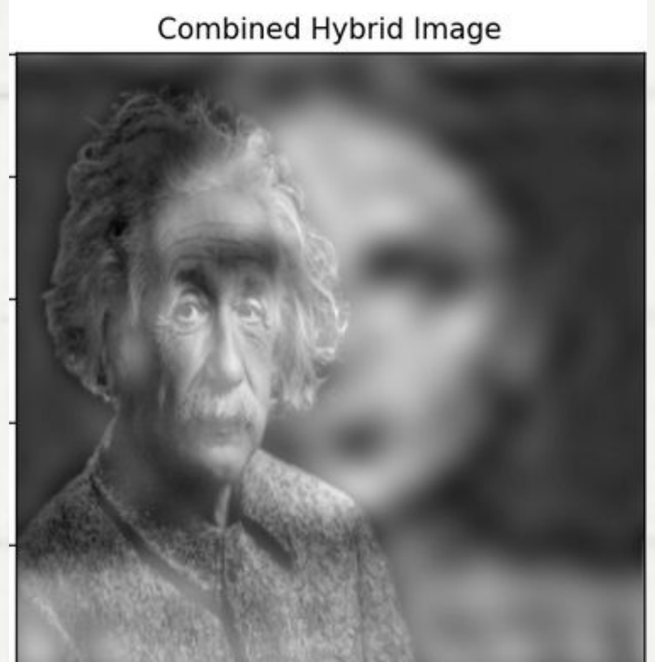


Image 2 after hpf


Combined Hybrid Image

Edge Detection

It is an image processing technique that identifies boundaries by detecting points with significant intensity changes, often corresponding to objects or regions, such as dark object boundaries.Its main goals are

- Localization : involves locating edges in an image precisely
- Detection : identification of significant intensity changes that indicate object boundaries.

There are various methods for edge detection and here will discuss two of them .

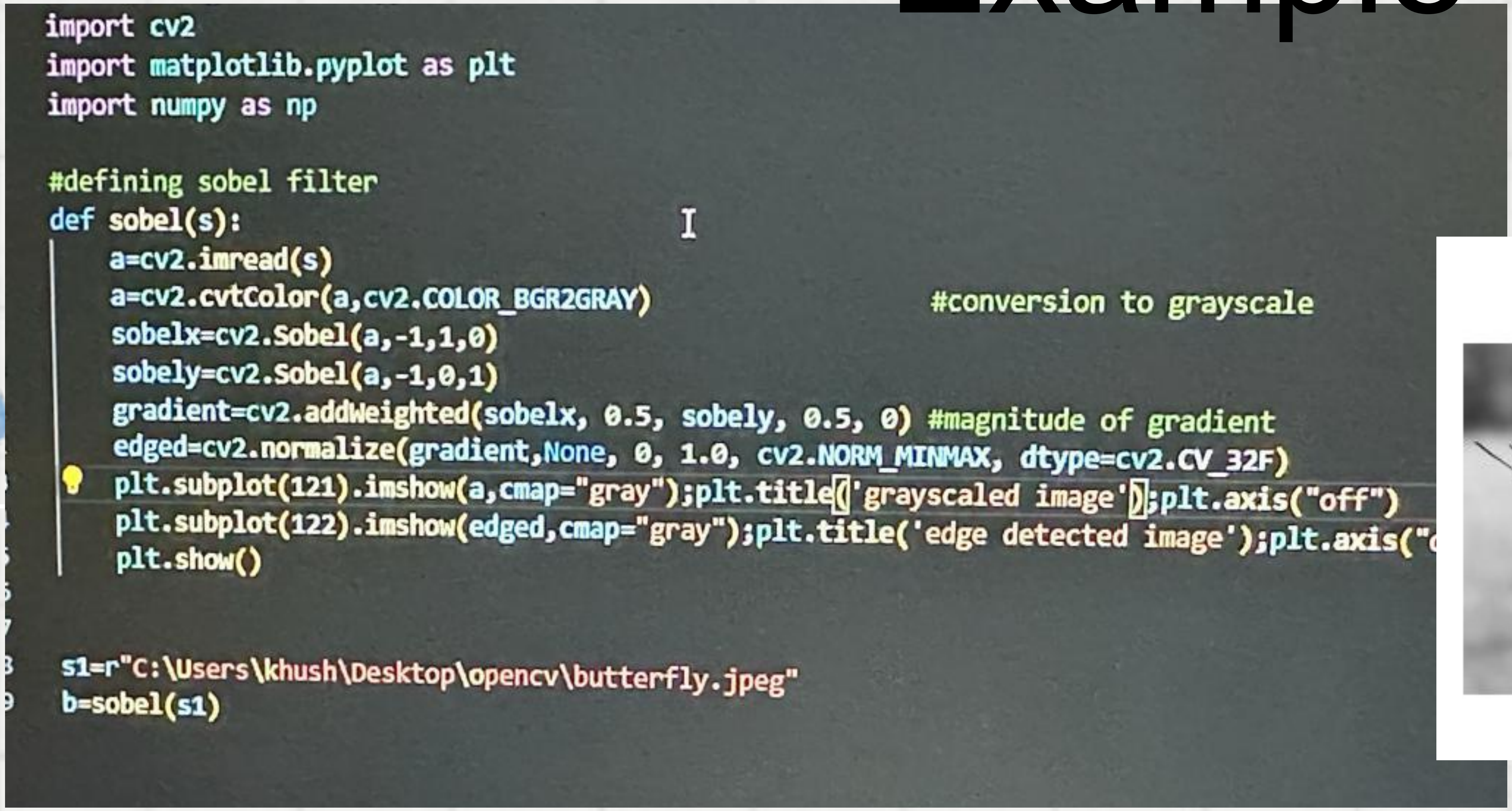
- Sobel Operator
- Canny Edge detection

Sobel Operator

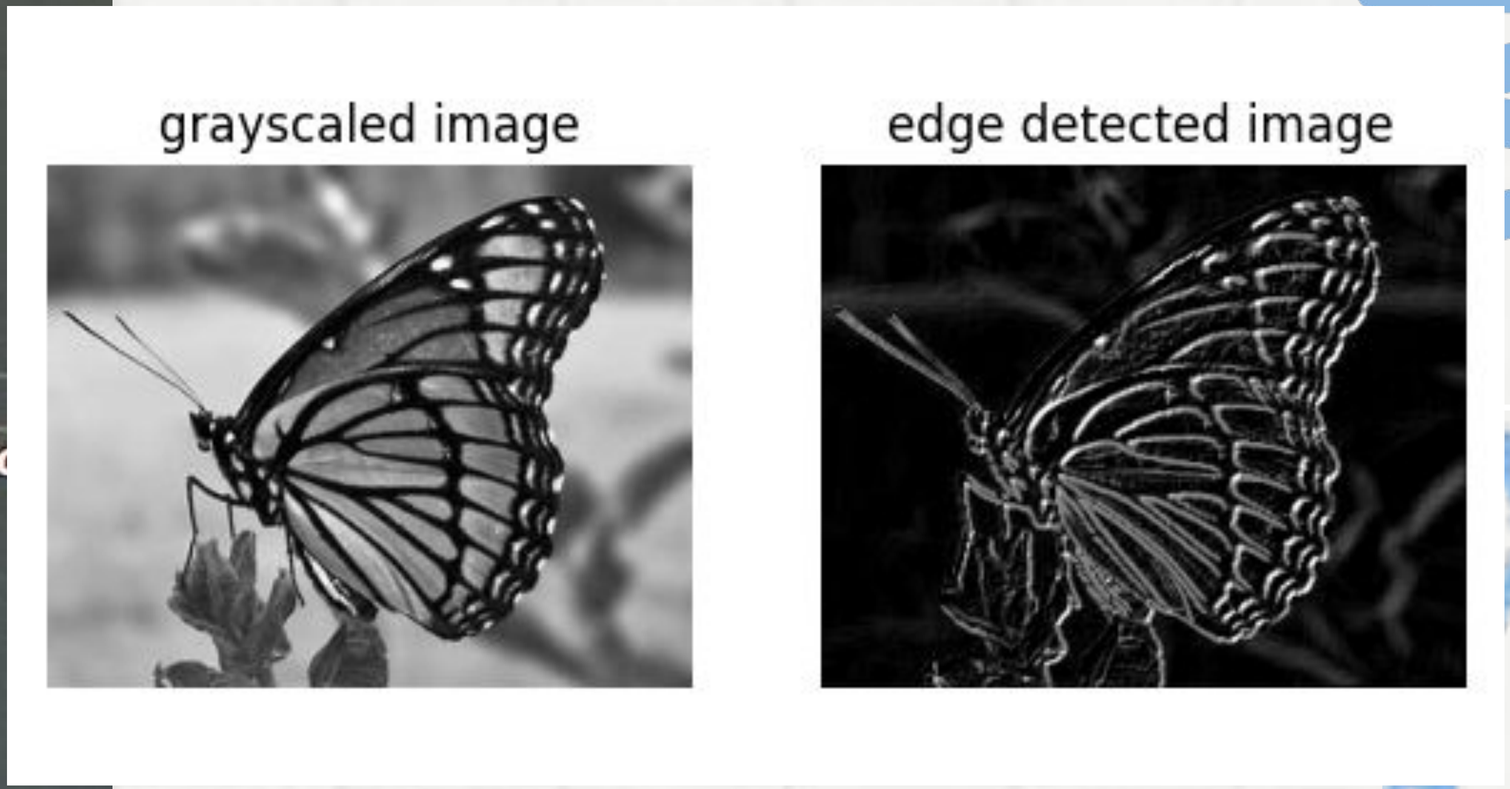
The process involves calculating the gradient of image intensity in both horizontal and vertical directions, identifying significant edges.Its working can explained in following steps :

1. Grayscale conversion – works on gray scaled images.
2. Convolution – use of kernels to measure change in respective directions.It involves sliding the kernels over an image, calculating the product of kernel elements and pixel values, and summarizing the results for each pixel.
3. Gradient calculation – The filter generates two gradient images, for horizontal and vertical changes, and calculates the gradient magnitude at each pixel.
4. Normalising – To ensure the output image is within the valid range, we normalize it in case any pixel value exceeds 255 or falls below 0.

Example



```
import cv2
import matplotlib.pyplot as plt
import numpy as np

#defining sobel filter
def sobel(s):
    a=cv2.imread(s)
    a=cv2.cvtColor(a,cv2.COLOR_BGR2GRAY)                    #conversion to grayscale
    sobelx=cv2.Sobel(a,-1,1,0)
    sobely=cv2.Sobel(a,-1,0,1)
    gradient=cv2.addWeighted(sobelx, 0.5, sobely, 0.5, 0) #magnitude of gradient
    edged=cv2.normalize(gradient,None, 0, 1.0, cv2.NORM_MINMAX, dtype=cv2.CV_32F)
    plt.subplot(121).imshow(a,cmap="gray");plt.title('grayscaled image');plt.axis("off")
    plt.subplot(122).imshow(edged,cmap="gray");plt.title('edge detected image');plt.axis("
    plt.show()


s1=r"C:\Users\khush\Desktop\opencv\butterfly.jpeg"
b=sobel(s1)
```

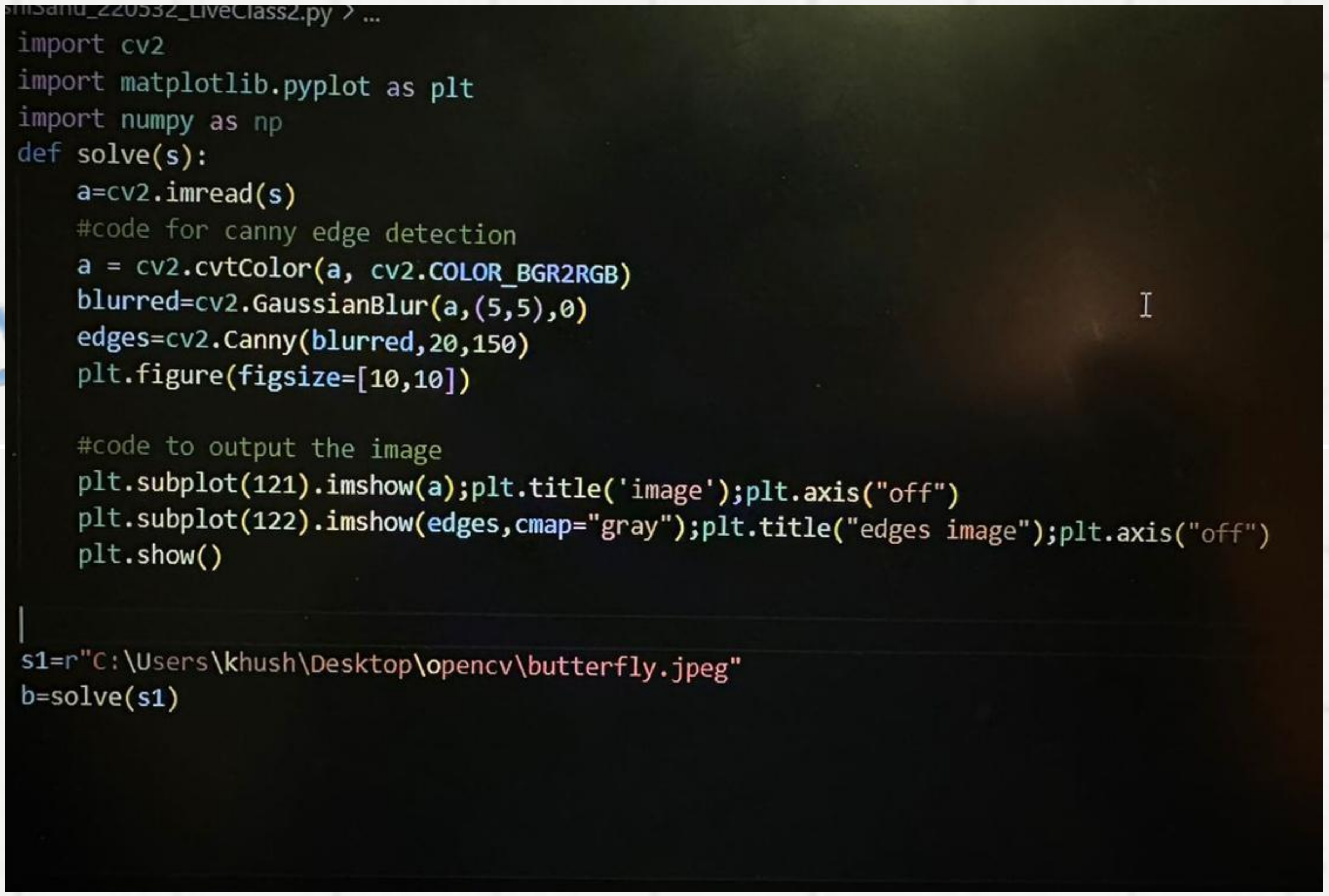


Canny Edge detection

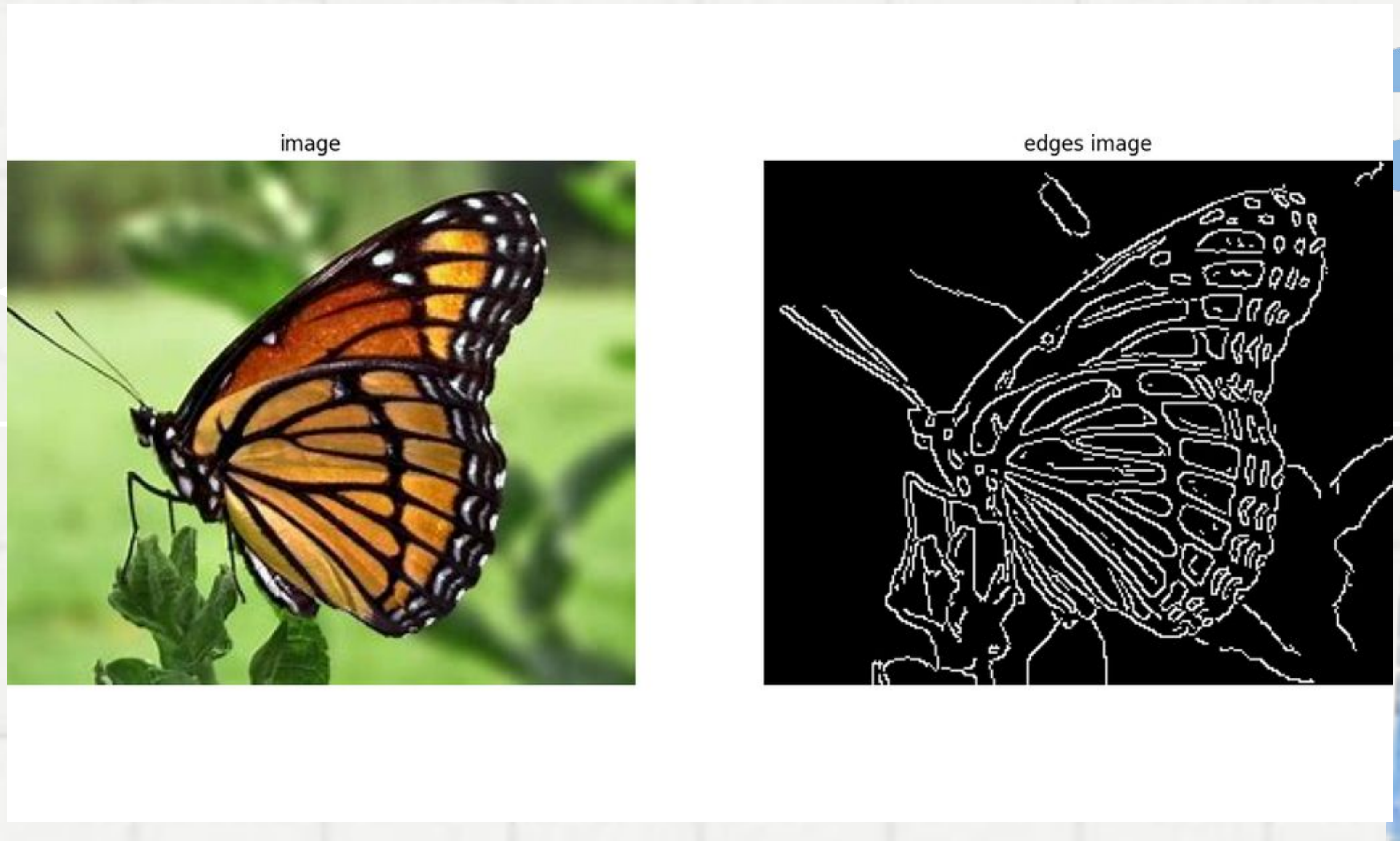
The Canny edge detector is a multi-stage algorithm that accurately identifies edges in images while suppressing noise.The main steps involved are :

1. Smoothing - By applying gaussian blur we can reduce noise and unwanted details in the image.
2. Gradient Calculation - same as it is done for sobel opeartor.
3. Double thresholding - to classify pixels into three categories: strong edges, weak edges, and non-edges.
4. Edge tracking - used to connect weak edges to strong edges which gives continuous and complete edges.

Example



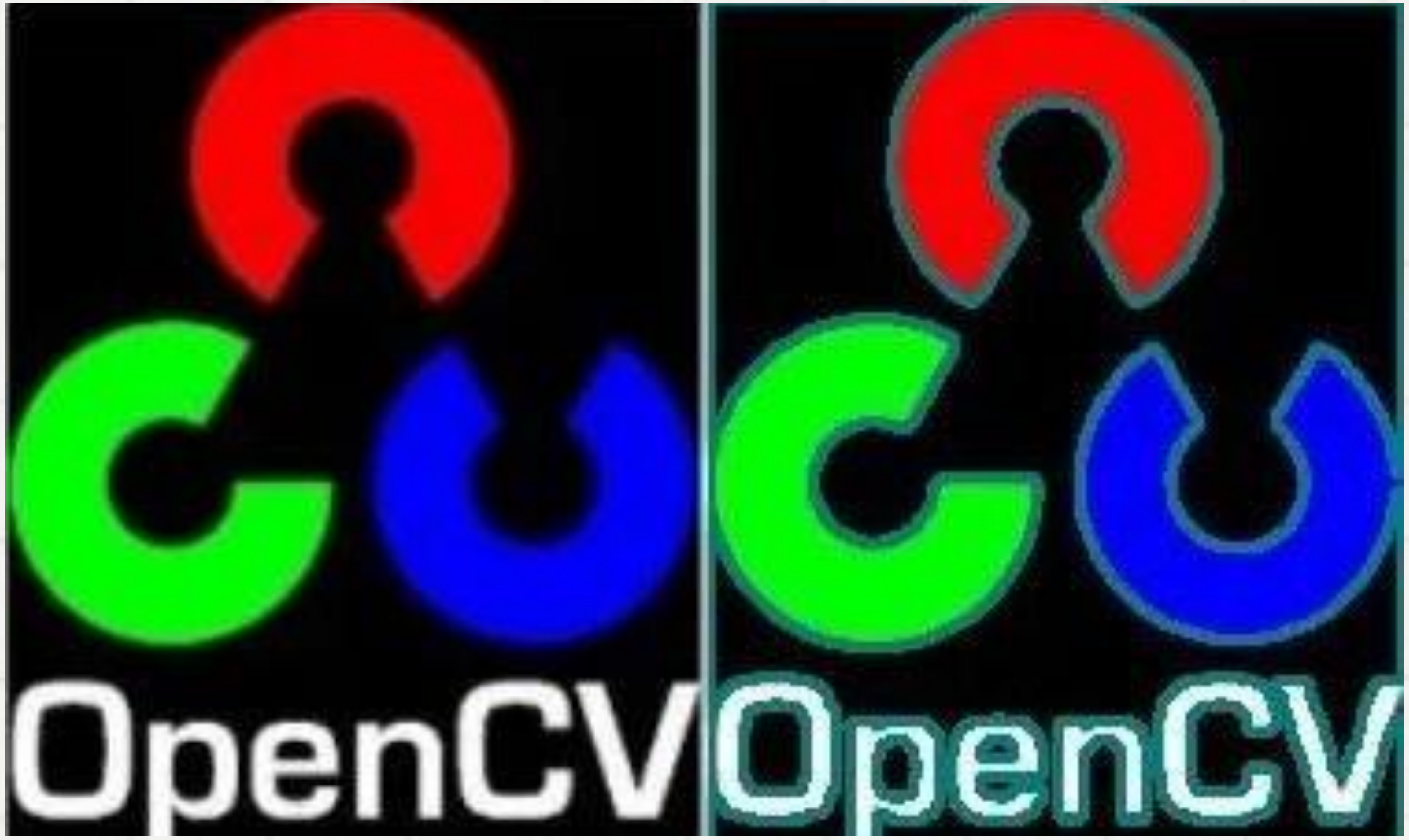
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
def solve(s):
    a=cv2.imread(s)
    #code for canny edge detection
    a = cv2.cvtColor(a, cv2.COLOR_BGR2RGB)
    blurred=cv2.GaussianBlur(a,(5,5),0)
    edges=cv2.Canny(blurred,20,150)
    plt.figure(figsize=[10,10])

    #code to output the image
    plt.subplot(121).imshow(a);plt.title('image');plt.axis("off")
    plt.subplot(122).imshow(edges,cmap="gray");plt.title("edges image");plt.axis("off")
    plt.show()


s1=r"C:\Users\khush\Desktop\opencv\butterfly.jpeg"
b=solve(s1)
```



Contour

Contouring refers to the process of outlining or delineating the boundaries of objects or regions in an image. In image processing or computer vision, contouring is often used to identify and represent the shapes of objects within an image. It can be done manually by experts or automatically using algorithms. It can be used for various purposes, such as object recognition, shape analysis, and image segmentation.

Example:



Input

Output

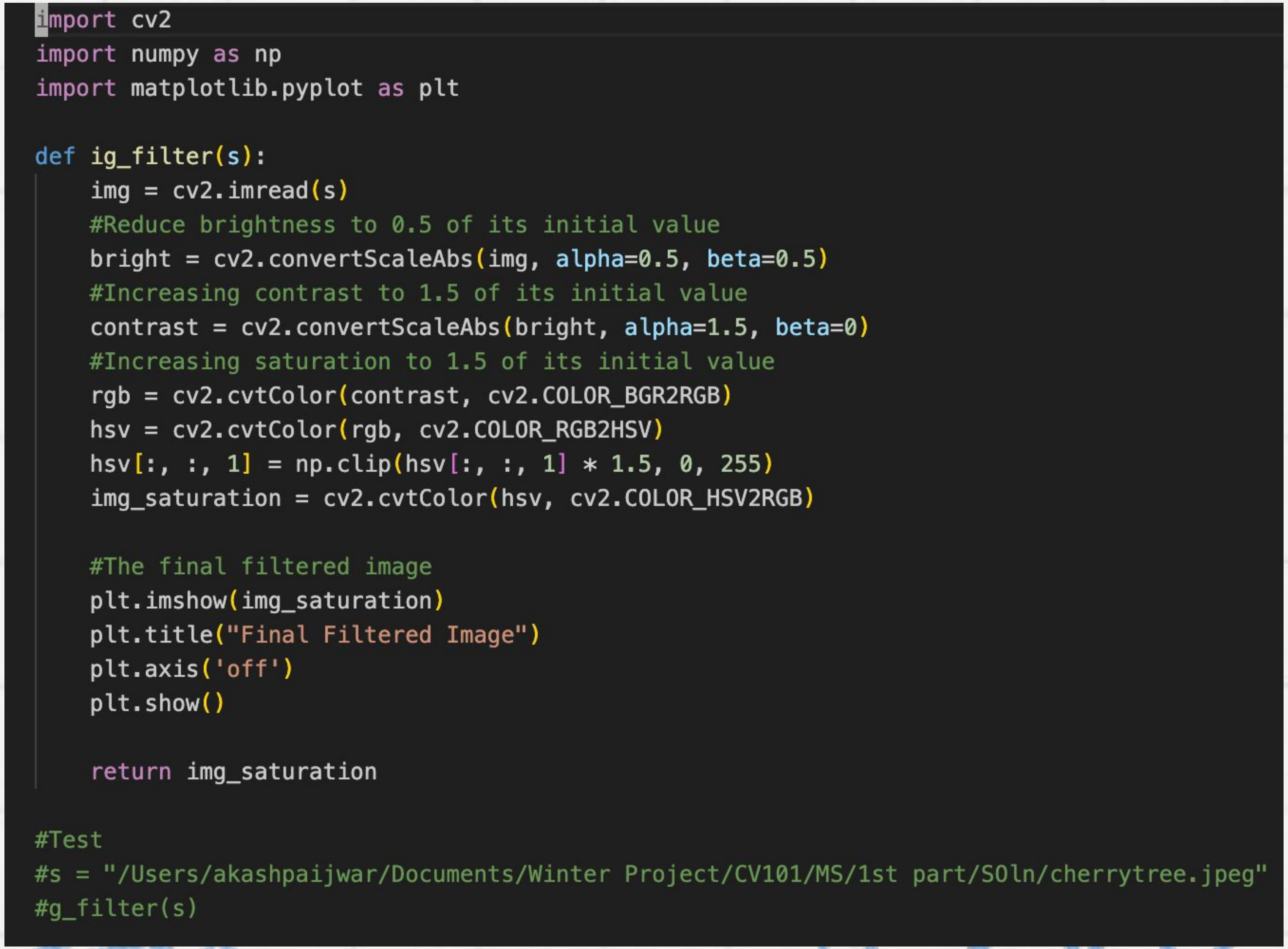
Image_Filter

Brightness refers to the overall level of light in an image. Low brightness means that the image is relatively dark, and the details might be harder to discern. It is a measure of how much light is being emitted. The cv2.convertScaleAbs function with alpha as the scaling factor and beta as the offset is commonly used.

Contrast refers to the sharpness of an image. Increasing the contrast level will result in brighter highlights and darker shadows. While decreasing the contrast will bring the highlights down and the shadows up. The cv2.convertScaleAbs function is also used for contrast adjustment, where alpha is the scaling factor.

Saturation refers to the intensity of colors in an image. A high saturation level indicates that the colors are very vivid and distinct. Decreasing saturation removes color from the image, creating a more muted or grayscale appearance The cv2.cvtColor function is used to convert between

Example

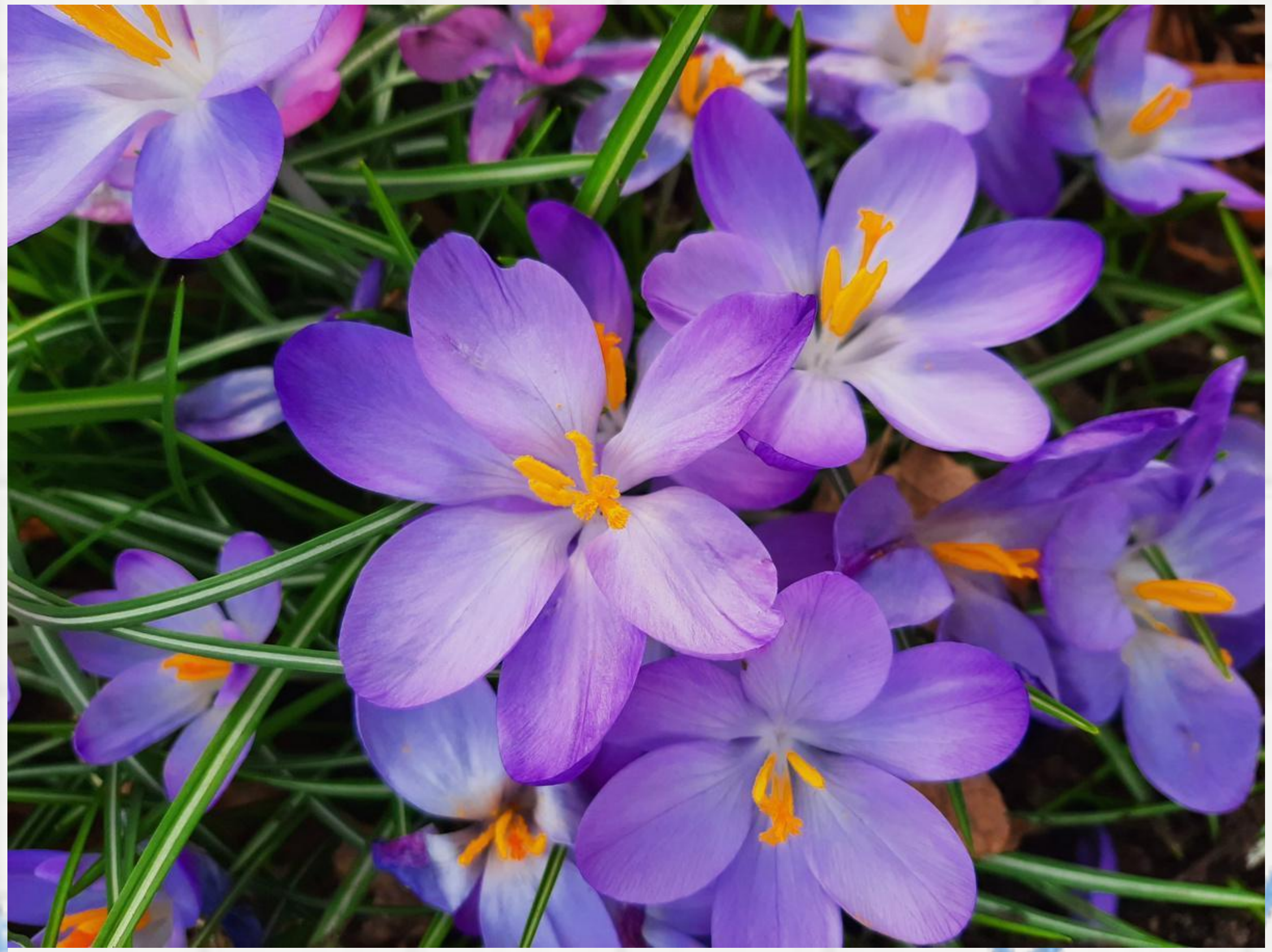


```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

def ig_filter(s):
    img = cv2.imread(s)
    #Reduce brightness to 0.5 of its initial value
    bright = cv2.convertScaleAbs(img, alpha=0.5, beta=0.5)
    #Increasing contrast to 1.5 of its initial value
    contrast = cv2.convertScaleAbs(bright, alpha=1.5, beta=0)
    #Increasing saturation to 1.5 of its initial value
    rgb = cv2.cvtColor(contrast, cv2.COLOR_BGR2RGB)
    hsv = cv2.cvtColor(rgb, cv2.COLOR_RGB2HSV)
    hsv[:, :, 1] = np.clip(hsv[:, :, 1] * 1.5, 0, 255)
    img_saturation = cv2.cvtColor(hsv, cv2.COLOR_HSV2RGB)

    #The final filtered image
    plt.imshow(img_saturation)
    plt.title("Final Filtered Image")
    plt.axis('off')
    plt.show()

    return img_saturation

#Test
#s = "/Users/akashpaijwar/Documents/Winter Project/CV101/MS/1st part/SOln/cherrytree.jpeg"
#g_filter(s)
```

It follows these steps:

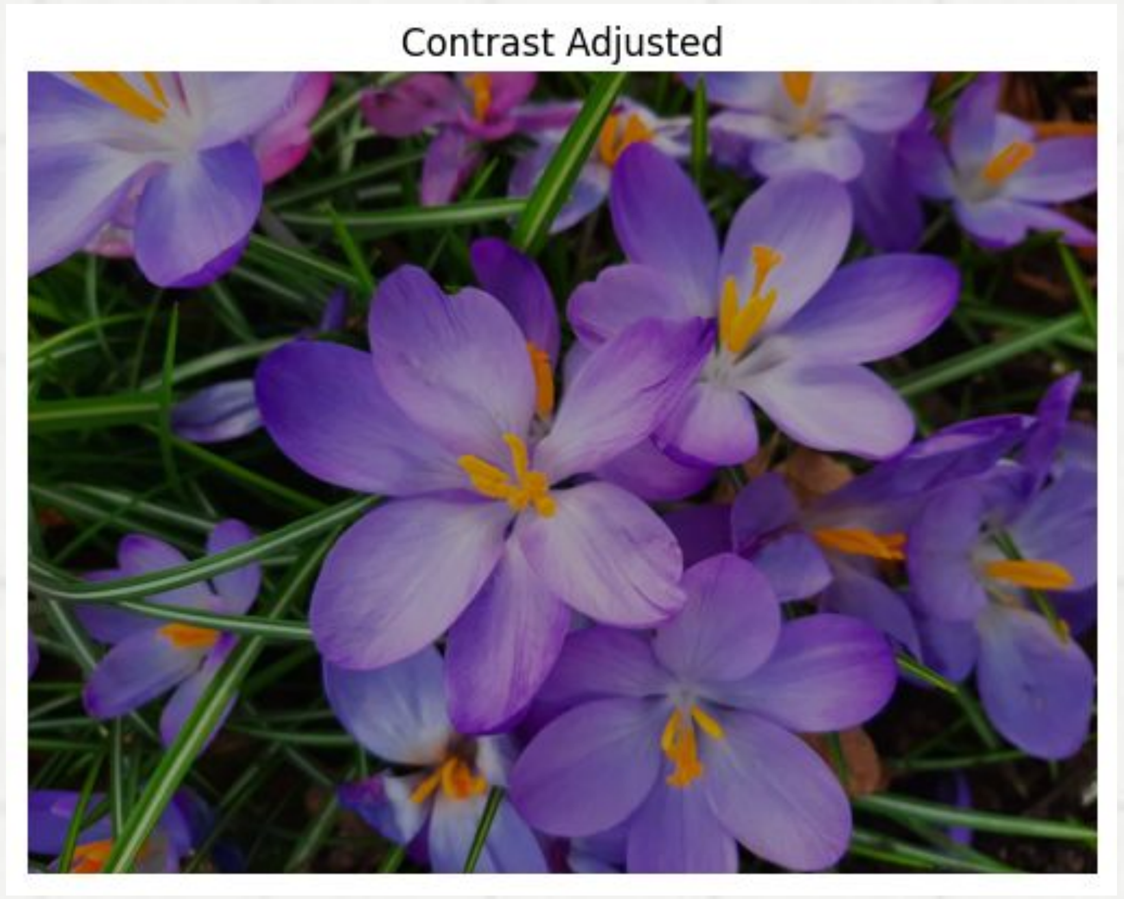
1. Reducing the brightness to 0.5 of its initial value
2. Increasing the contrast to 1.5 of its initial value
3. Increasing the saturation of the image to 1.5 of its initial value

Input



Output

Brightness Adjusted


Contrast Adjusted

Final Filtered Image

Hough Transform

- Elegant method for direct object recognition
- Edges need not be connected
- Complete object need not be visible
- Key Idea: Edges VOTE for the possible model

The Motivation for Polar Coordinates

- Vertical lines cannot be mapped to the (m,c) space, since:

  m = infinte

  b = ?
- Vertical lines can be described using polar coordinates

  x*cos(theta)+y*sin(theta) = r

The Parameter Plane

- Each point in the (x,y) space(image plane) is mapped to a straight line in the (m,c) space (parameter plane).
- A straight line in the (x,y) space (image plane) is mapped to the intersection point of the lines corresponding to its points, in the (m,c) space (parameter plane).

flowchart

input Edge image

Mapping edge points to parametric space

Creating ACUMULATOR

Find the point with maximum voting

Detection of Line

The Accumulator Concept

- The (m,c) space (parameter plane) is subdivided in cells.
- Each pixel (x,y) int he original image vote in the (m,c) space for each line passing through it.
- The votes are summed in an Accumulator

- It is widely used in the feature extraction.
- It can extract the lines even through noice data.
- By Hough transform, we can detect the missing. lines which are broken while edge detection.
- We can use it in lane detection of the road.
- Another big advantage is used for the building-edge extraction.

The Straight Line

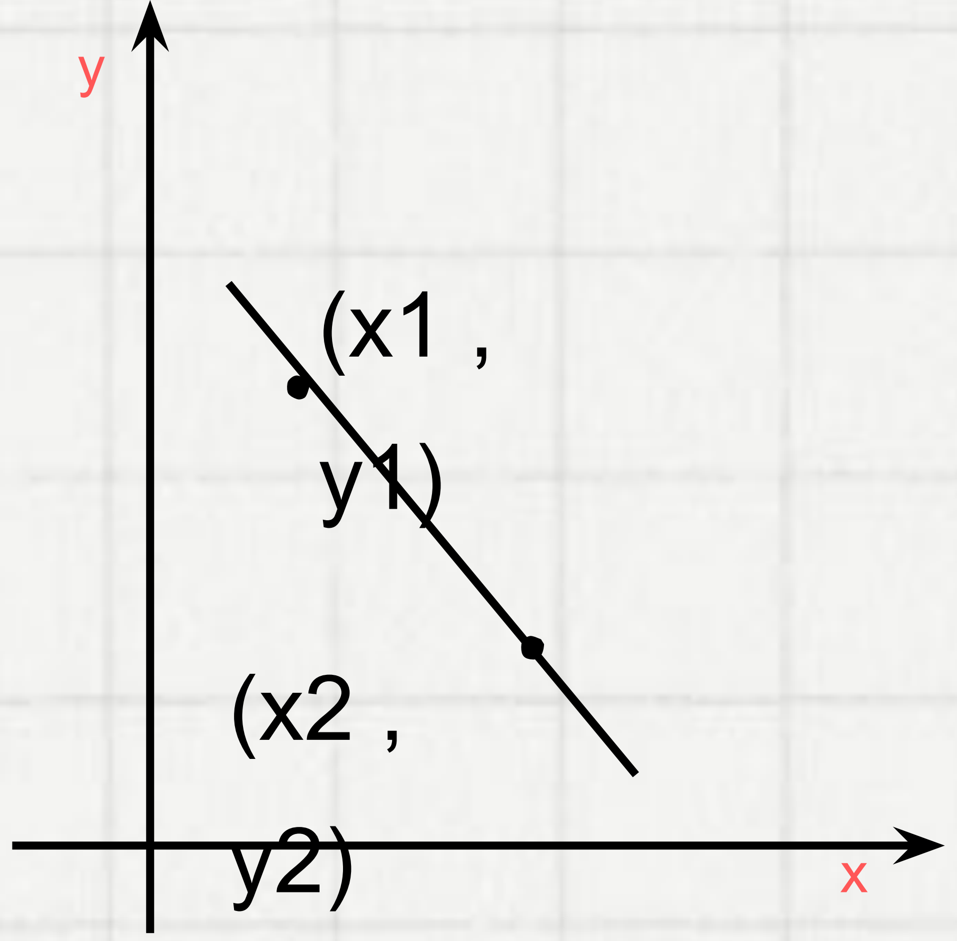
- For each points (x, y) in the line the following equation applies:
- Therefore:

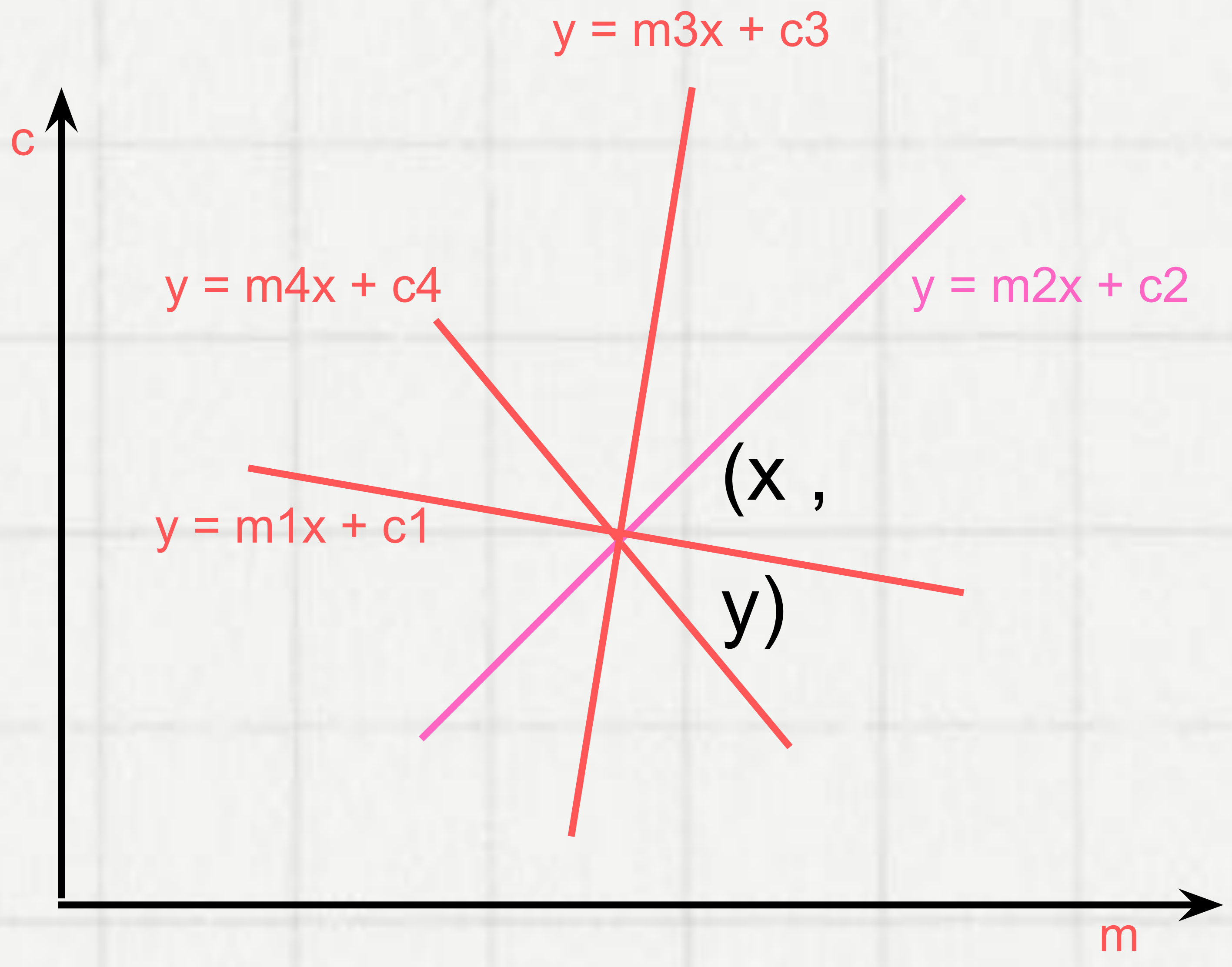
  y = m*x + c

Multiple lines over a single point

- Each pair(m,c) defines a distinct straight line containing the points (x,y)

Your paragraph text

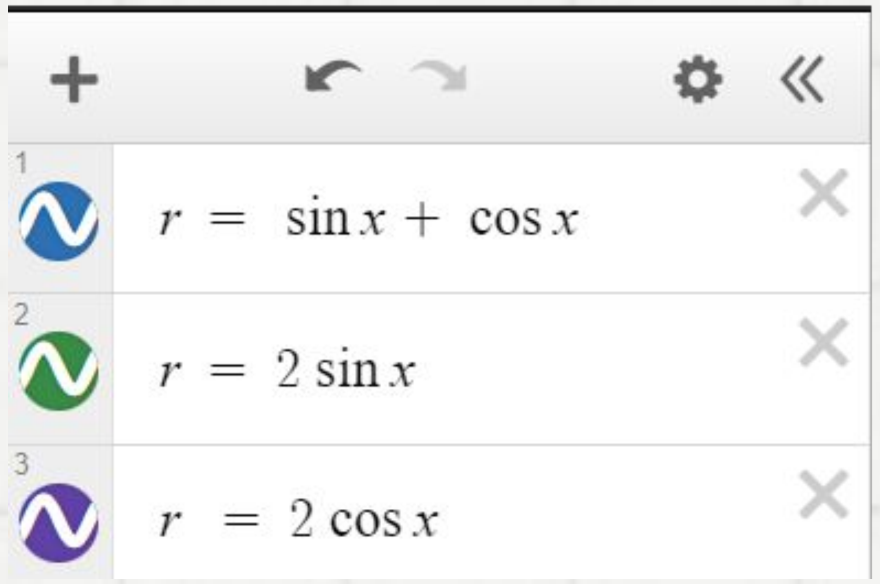




Using Polar Coordinates

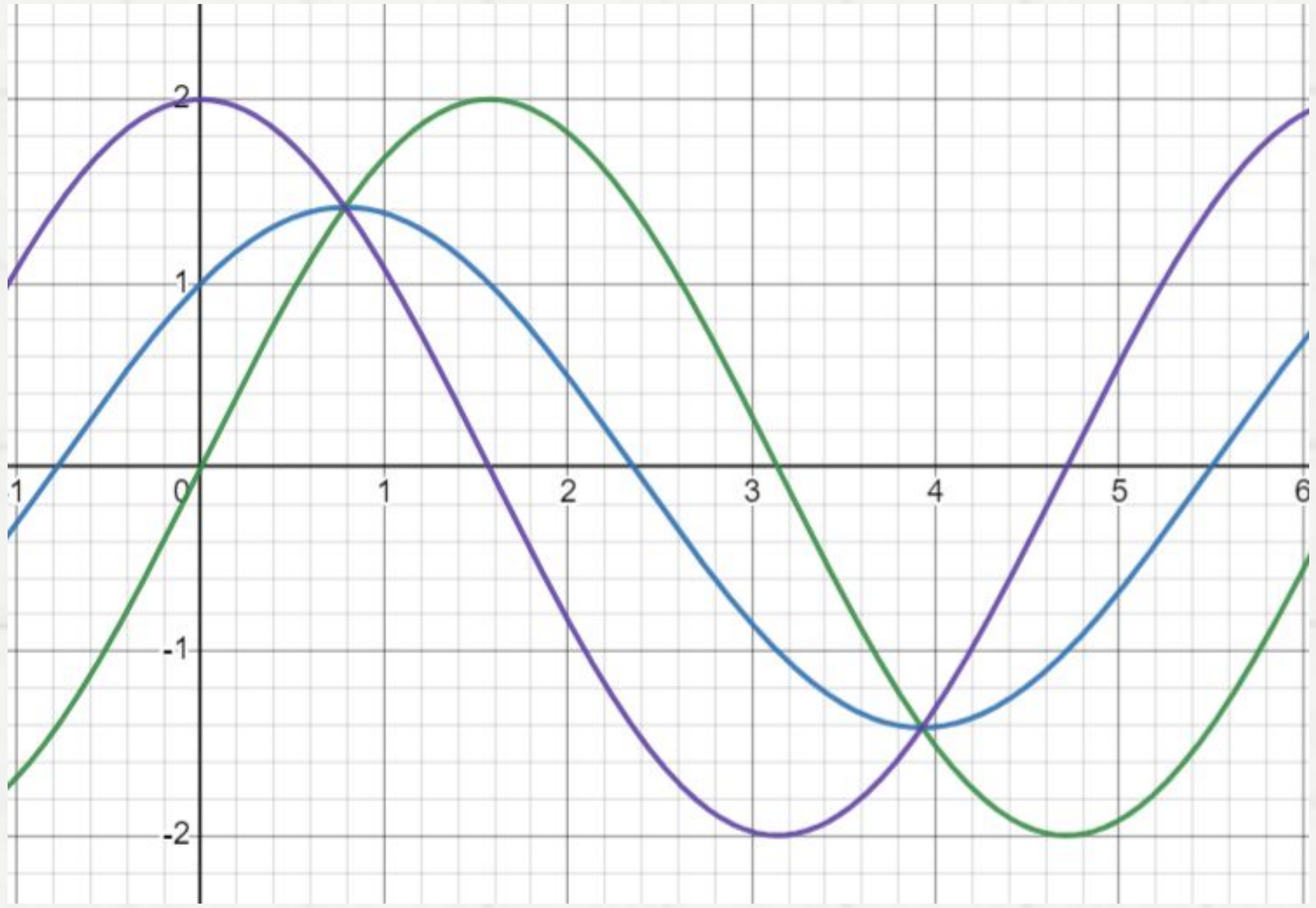
- For each point (x , y) in the line the following equation applies:
- In particular:

  r = xcos(thetha) + ysin(thetha)

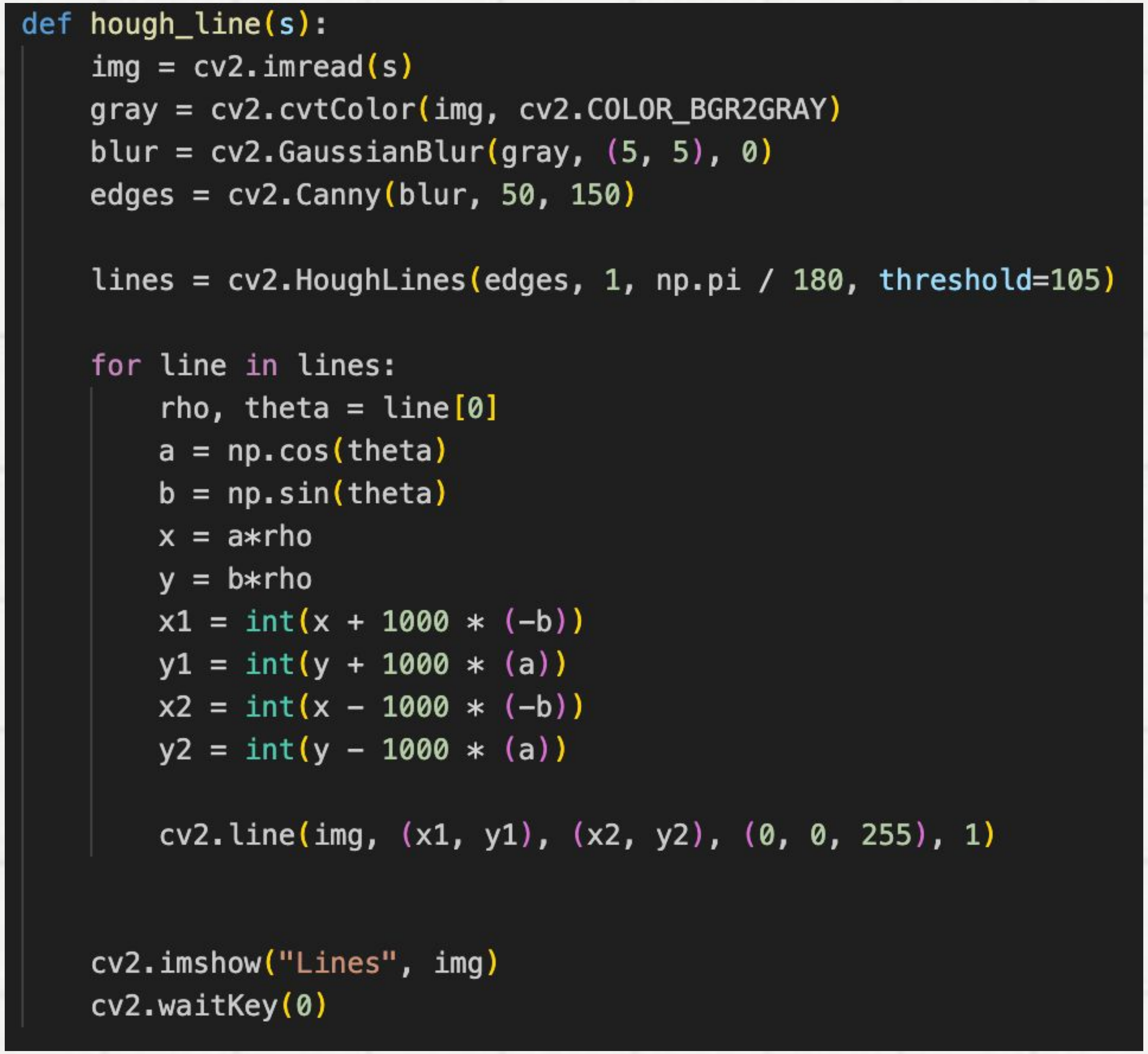

1 $r = \sin x + \cos x$

2 $r = 2\sin x$

3 $r = 2\cos x$



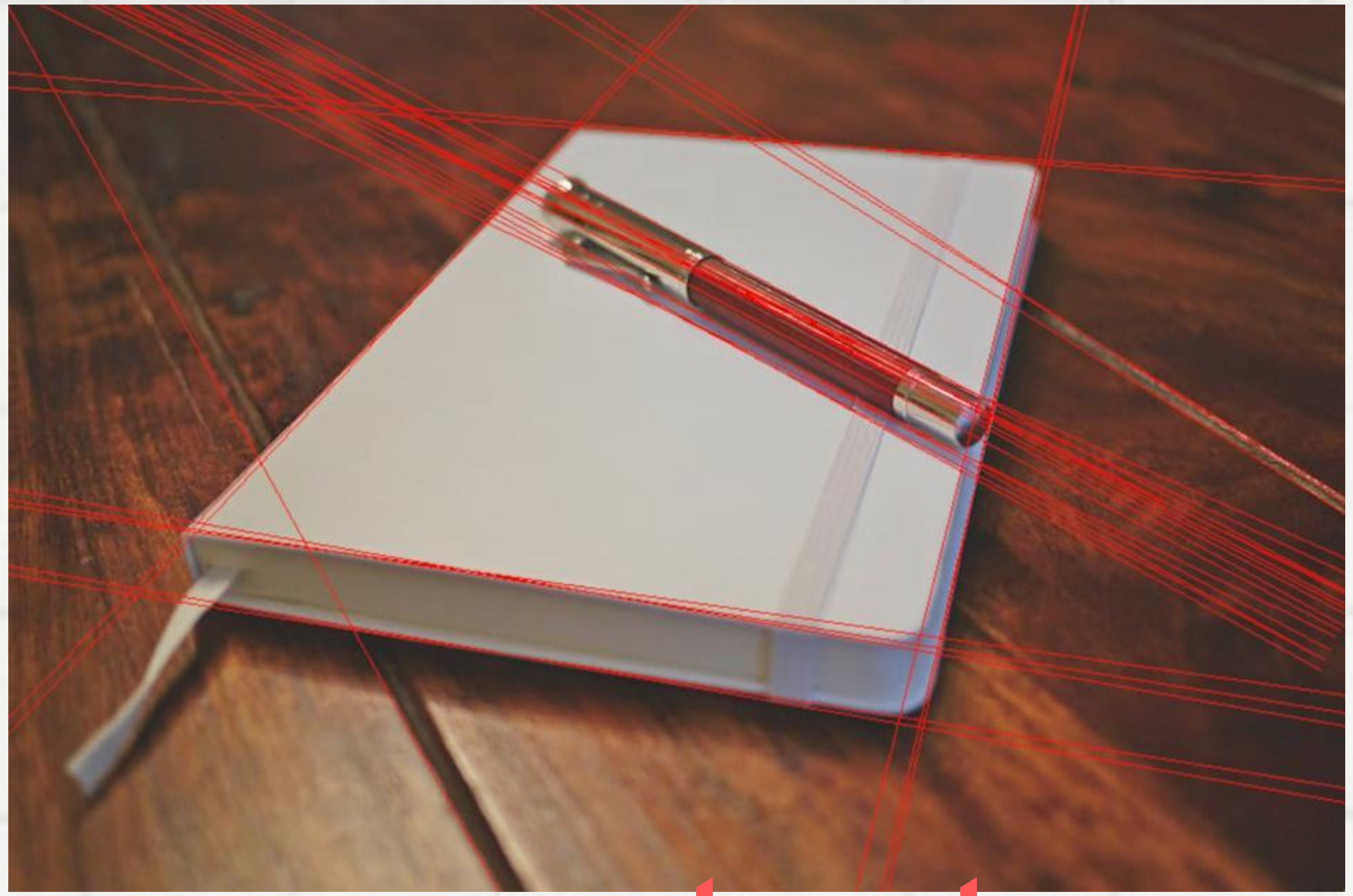
Your paragraph text



```python
def hough_line(s):
    img = cv2.imread(s)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blur, 50, 150)

    lines = cv2.HoughLines(edges, 1, np.pi / 180, threshold=105)

    for line in lines:
        rho, theta = line[0]
        a = np.cos(theta)
        b = np.sin(theta)
        x = a*rho
        y = b*rho
        x1 = int(x + 1000 * (-b))
        y1 = int(y + 1000 * (a))
        x2 = int(x - 1000 * (-b))
        y2 = int(y - 1000 * (a))

        cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 1)


    cv2.imshow("Lines", img)
    cv2.waitKey(0)
```

Output

input


output

Shape Detection

Shape detection is a technique used in image processing and computer vision, that enables us to identify and classify various shapes present within an image. By analyzing the geometric attributes of objects, shape detection forms the basis for numerous applications, including object recognition, industrial automation, and quality assessment.

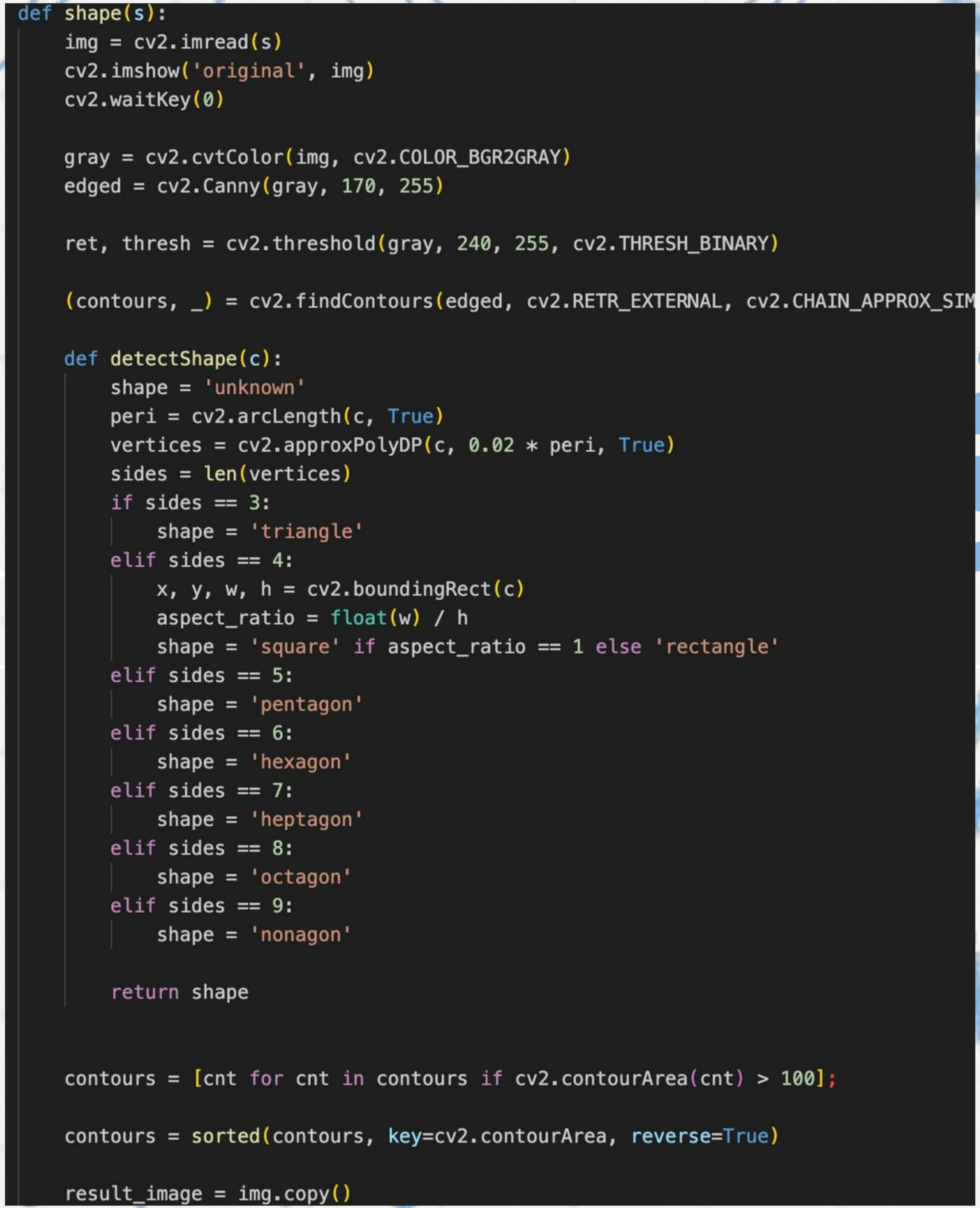
We will use the following functions to implement shape detction -

cv.findContours( ) is the function that will find out all the contours present in the grayscale image based on the thresholds.

cv.approxPolyDP() function returns all the polygons curve based on the contour with precision. Both the True parameters specify the close contour and curve.

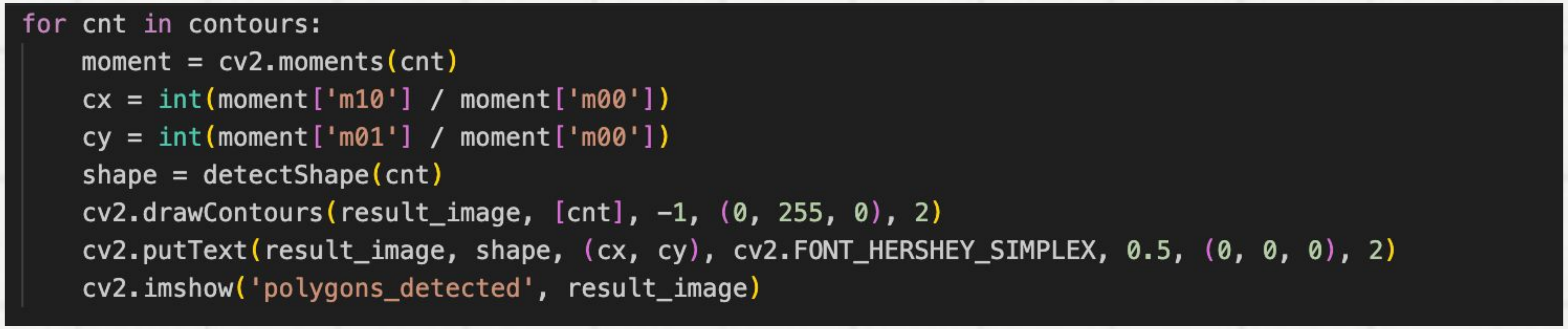
cv2.drawContours() This draws a contour. It can also draw a shape if you provide boundary points.

Function to detect shape with sides less than 10



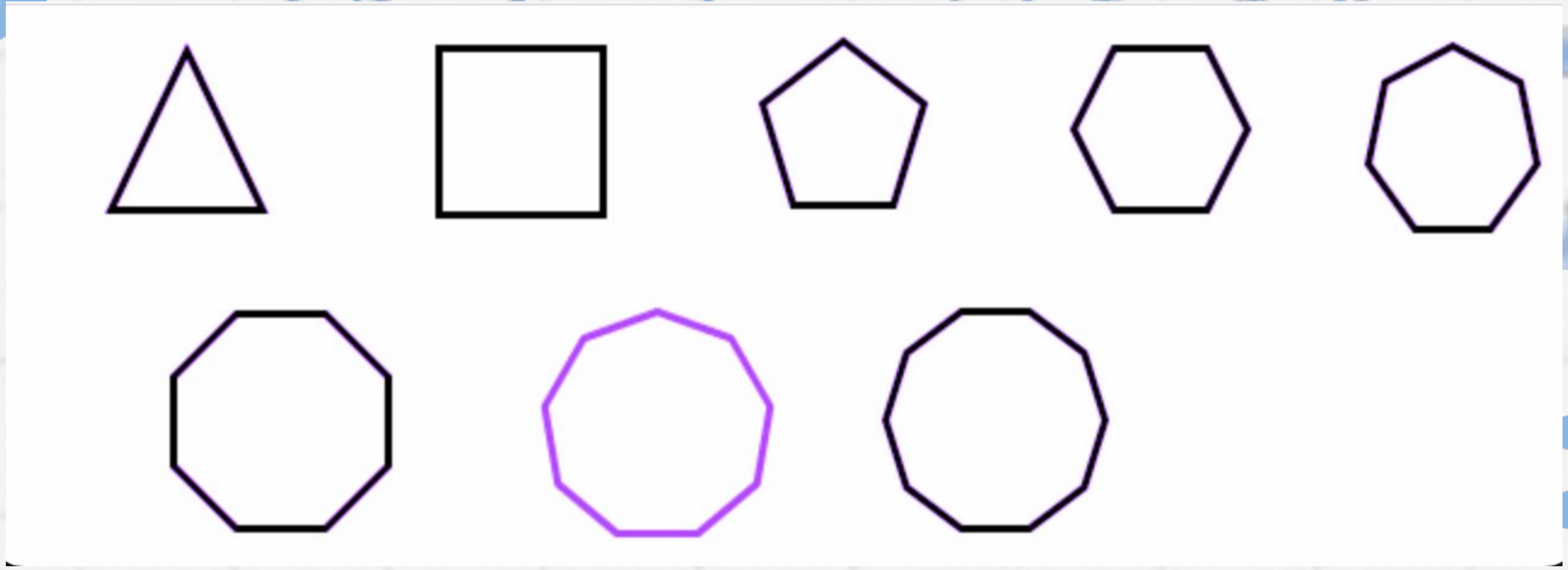
```
def shape(s):
    img = cv2.imread(s)
    cv2.imshow('original', img)
    cv2.waitKey(0)

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edged = cv2.Canny(gray, 170, 255)

    ret, thresh = cv2.threshold(gray, 240, 255, cv2.THRESH_BINARY)

    (contours, _) = cv2.findContours(edged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIM

    def detectShape(c):
        shape = 'unknown'
        peri = cv2.arcLength(c, True)
        vertices = cv2.approxPolyDP(c, 0.02 * peri, True)
        sides = len(vertices)
        if sides == 3:
            shape = 'triangle'
        elif sides == 4:
            x, y, w, h = cv2.boundingRect(c)
            aspect_ratio = float(w) / h
            shape = 'square' if aspect_ratio == 1 else 'rectangle'
        elif sides == 5:
            shape = 'pentagon'
        elif sides == 6:
            shape = 'hexagon'
        elif sides == 7:
            shape = 'heptagon'
        elif sides == 8:
            shape = 'octagon'
        elif sides == 9:
            shape = 'nonagon'

        return shape


    contours = [cnt for cnt in contours if cv2.contourArea(cnt) > 100];

    contours = sorted(contours, key=cv2.contourArea, reverse=True)

    result_image = img.copy()
```

Now we will detect centre of the shapes and display their names.

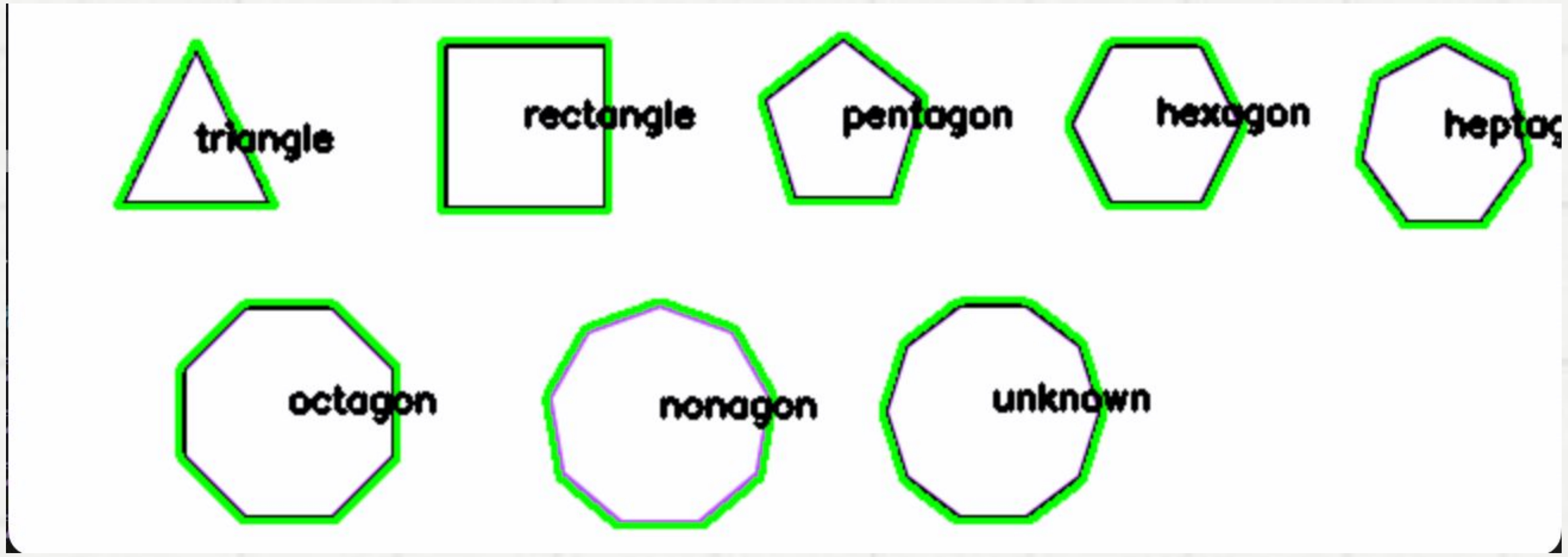


```python
for cnt in contours:
    moment = cv2.moments(cnt)
    cx = int(moment['m10'] / moment['m00'])
    cy = int(moment['m01'] / moment['m00'])
    shape = detectShape(cnt)
    cv2.drawContours(result_image, [cnt], -1, (0, 255, 0), 2)
    cv2.putText(result_image, shape, (cx, cy), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2)
    cv2.imshow('polygons_detected', result_image)
```

Input



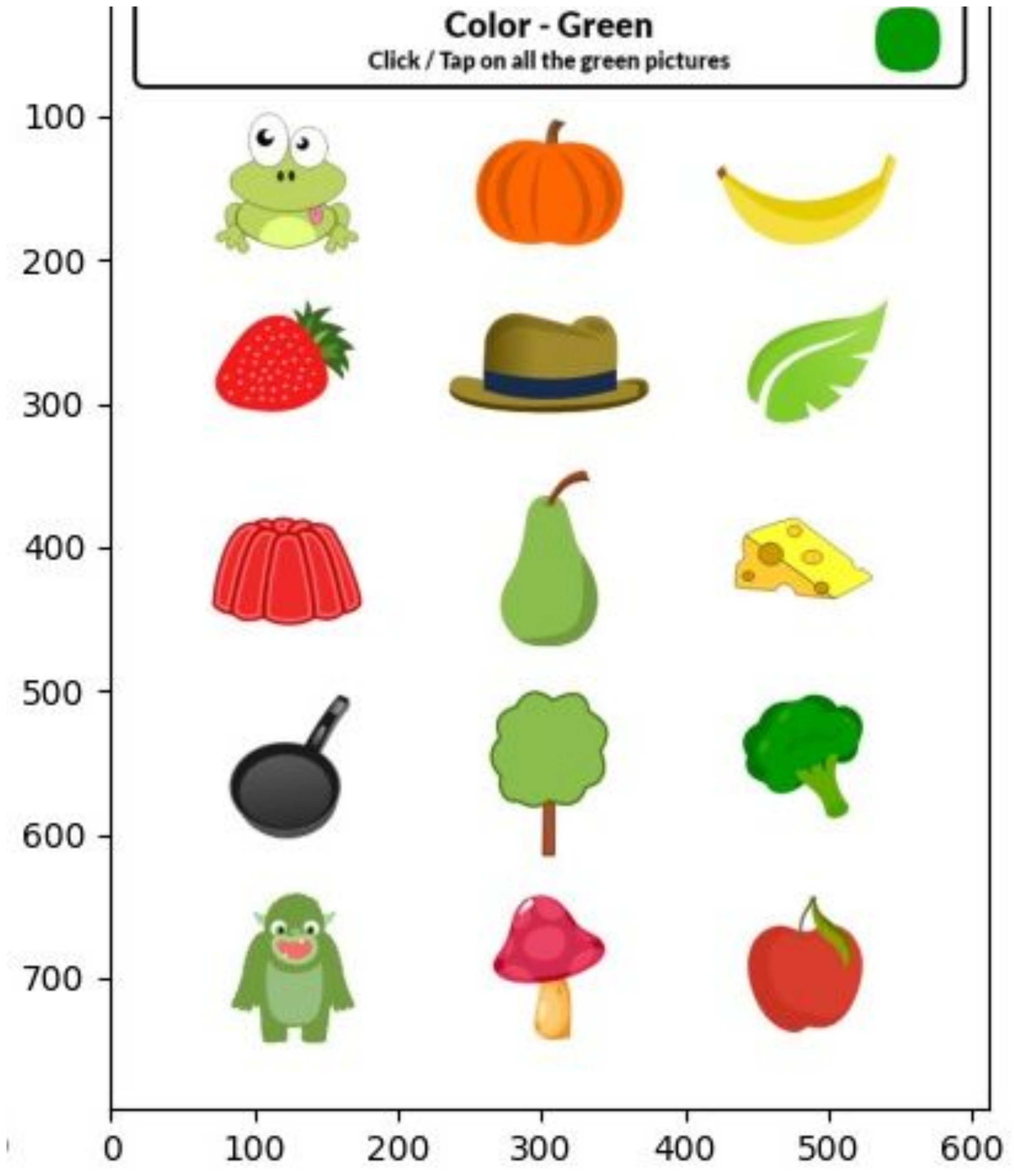
Output

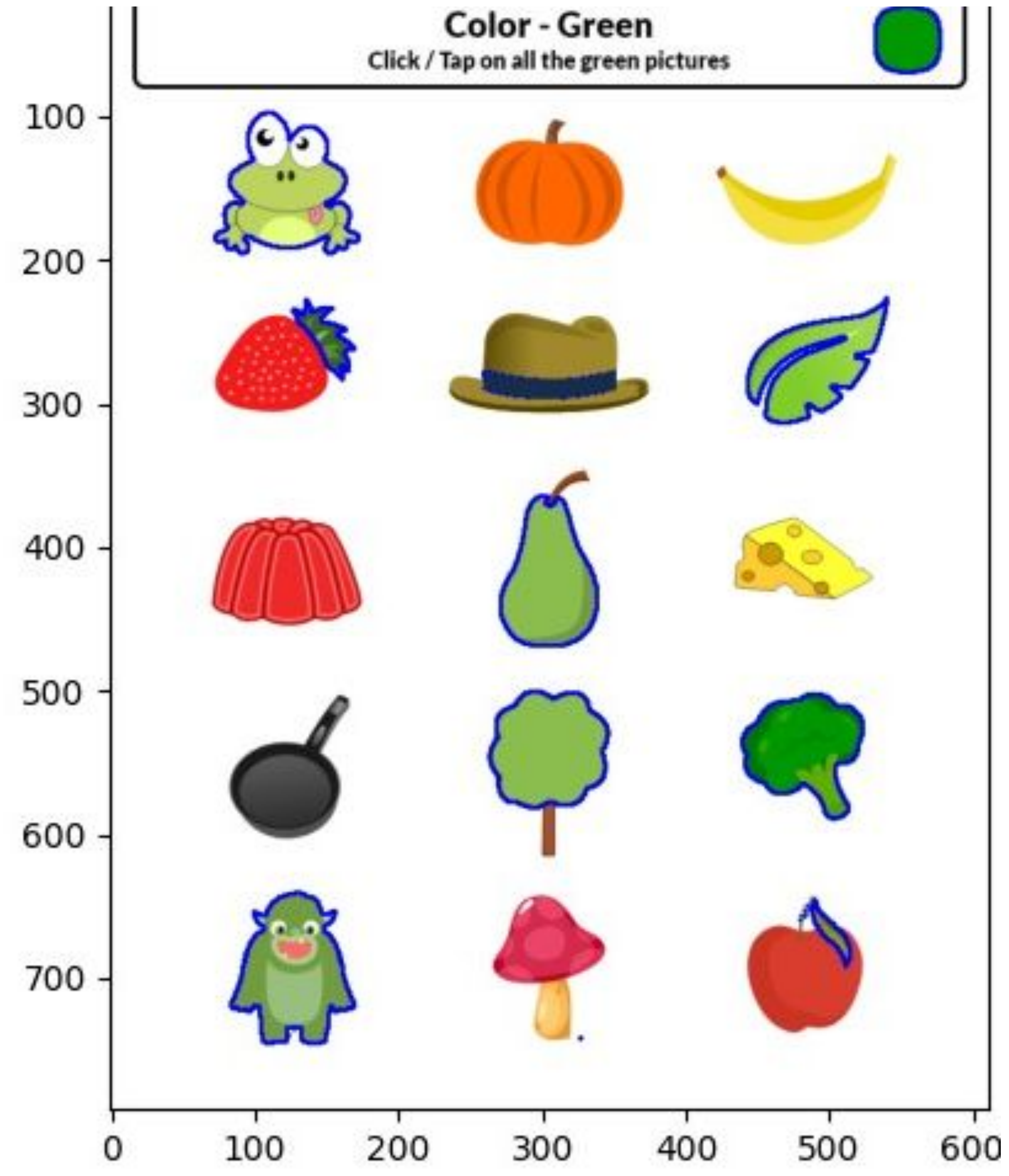


Color Detection

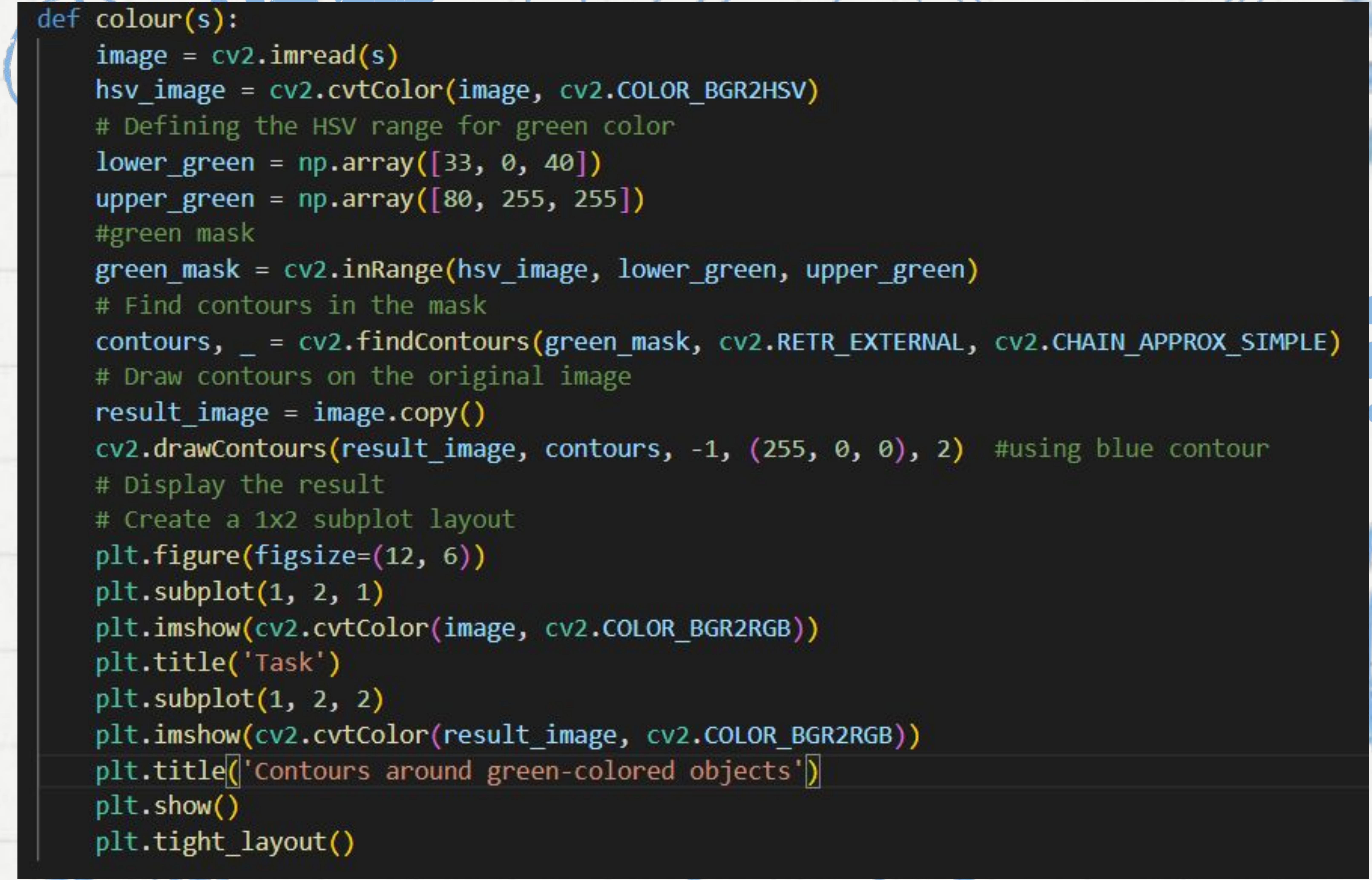
Color detection is a fundamental task in computer vision that involves identifying and extracting specific colors or color ranges from an image or video. The process typically involves converting the image from one color space to another, we converted it to HSV (Hue, Saturation, Value), and then setting a range of values to isolate the desired color or colors.
In the assignment particularly we detected a given color and made contours of another color around it. I have shown the input and output for it in the next slide.

COLOR DETECTION





C O D E



```python
def colour(s):
    image = cv2.imread(s)
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # Defining the HSV range for green color
    lower_green = np.array([33, 0, 40])
    upper_green = np.array([80, 255, 255])
    #green mask
    green_mask = cv2.inRange(hsv_image, lower_green, upper_green)
    # Find contours in the mask
    contours, _ = cv2.findContours(green_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    # Draw contours on the original image
    result_image = image.copy()
    cv2.drawContours(result_image, contours, -1, (255, 0, 0), 2)  #using blue contour
    # Display the result
    # Create a 1x2 subplot layout
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title('Task')
    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(result_image, cv2.COLOR_BGR2RGB))
    plt.title('Contours around green-colored objects')
    plt.show()
    plt.tight_layout()
```

Masking:

- Convert the image from the default BGR (Blue, Green, Red) color space to the HSV (Hue, Saturation, Value) color space.
- Use the cv2.inRange function to create a binary mask that extracts only the pixels within the specified color range.
- When implementing color detection algorithms, it's important to choose a color space and threshold values carefully based on the characteristics of the images or videos being processed.
- The cv2.inRange function creates a binary mask based on the specified color range. Pixels in the mask that fall within the specified range are set to white (255), and pixels outside the range are set to black (0). The resulting mask is a binary image where white pixels correspond to the regions with the detected color.
- The mask now contains information about where in the image the desired color is present. This binary mask can be used to isolate or highlight specific regions(desired colors) in the

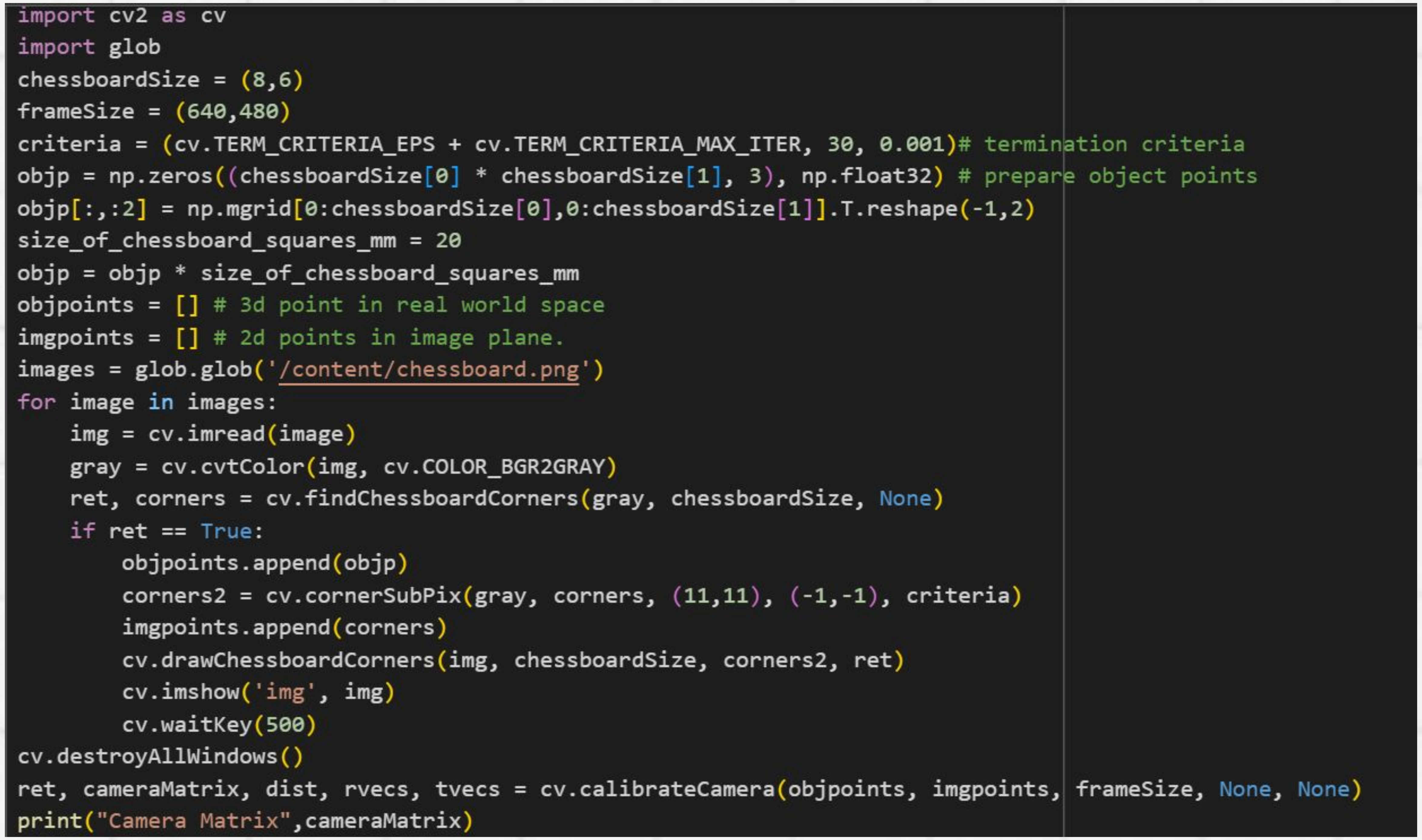
Calibration

Calibration is a process of adjusting or verifying the accuracy and precision of a measurement instrument, system, or device. The goal of calibration is to ensure that the measurements or outputs of the system are reliable and accurate, often by comparing them to a known standard. Calibration is crucial in various fields, including engineering, science, manufacturing, and technology.

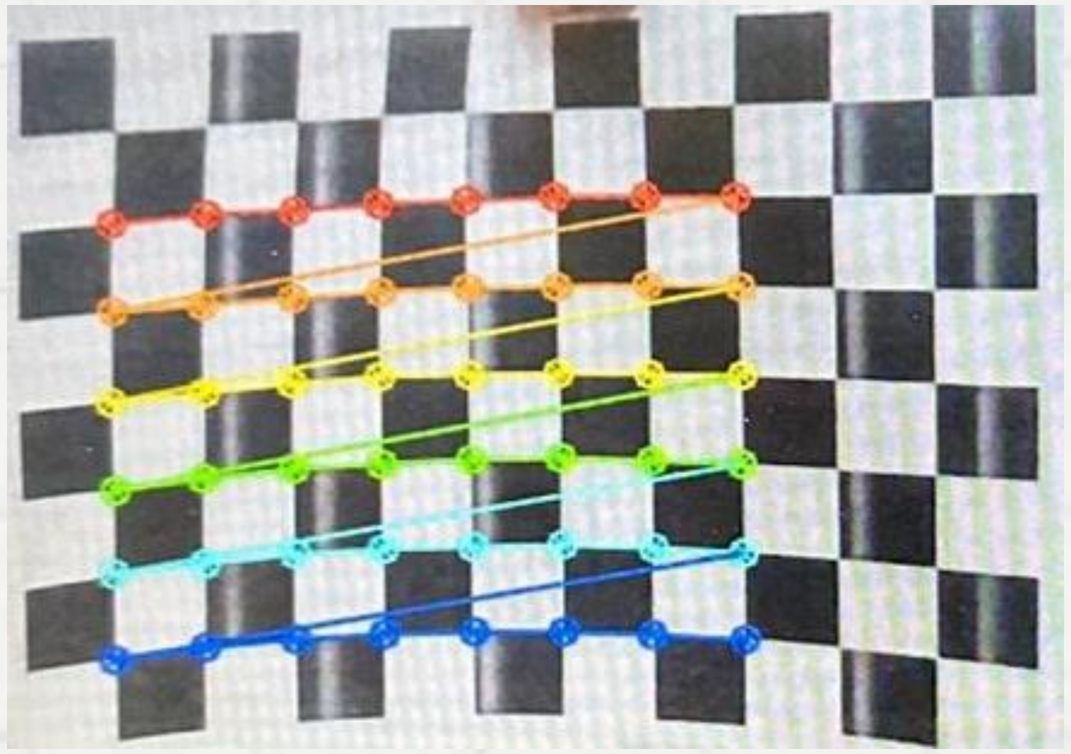
Camera Calibration:

Camera calibration is a crucial step in computer vision and image processing to correct distortions and obtain accurate measurements from images. The calibration process involves determining the intrinsic and extrinsic parameters of a camera, which describe its internal characteristics and its position and orientation in the 3D world, respectively.

Example:



```
import cv2 as cv
import glob
chessboardSize = (8,6)
frameSize = (640,480)
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)# termination criteria
objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3), np.float32) # prepare object points
objp[:,:2] = np.mgrid[0:chessboardSize[0],0:chessboardSize[1]].T.reshape(-1,2)
size_of_chessboard_squares_mm = 20
objp = objp * size_of_chessboard_squares_mm
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.
images = glob.glob('/content/chessboard.png')
for image in images:
    img = cv.imread(image)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    ret, corners = cv.findChessboardCorners(gray, chessboardSize, None)
    if ret == True:
        objpoints.append(objp)
        corners2 = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners)
        cv.drawChessboardCorners(img, chessboardSize, corners2, ret)
        cv.imshow('img', img)
        cv.waitKey(500)
cv.destroyAllWindows()
ret, cameraMatrix, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, frameSize, None, None)
print("Camera Matrix",cameraMatrix)
```



Output

Undistortion of Image

For correction of distortion in an image using OpenCV, we can use the camera calibration and distortion correction functions. The most common distortion in images captured by cameras is radial distortion, which can be corrected using calibration parameters.

The calibration process finds the camera matrix (mtx) and distortion coefficients (dist). The cv2.undistort function then uses these parameters to correct the distortion in the image.

Types of Distortion

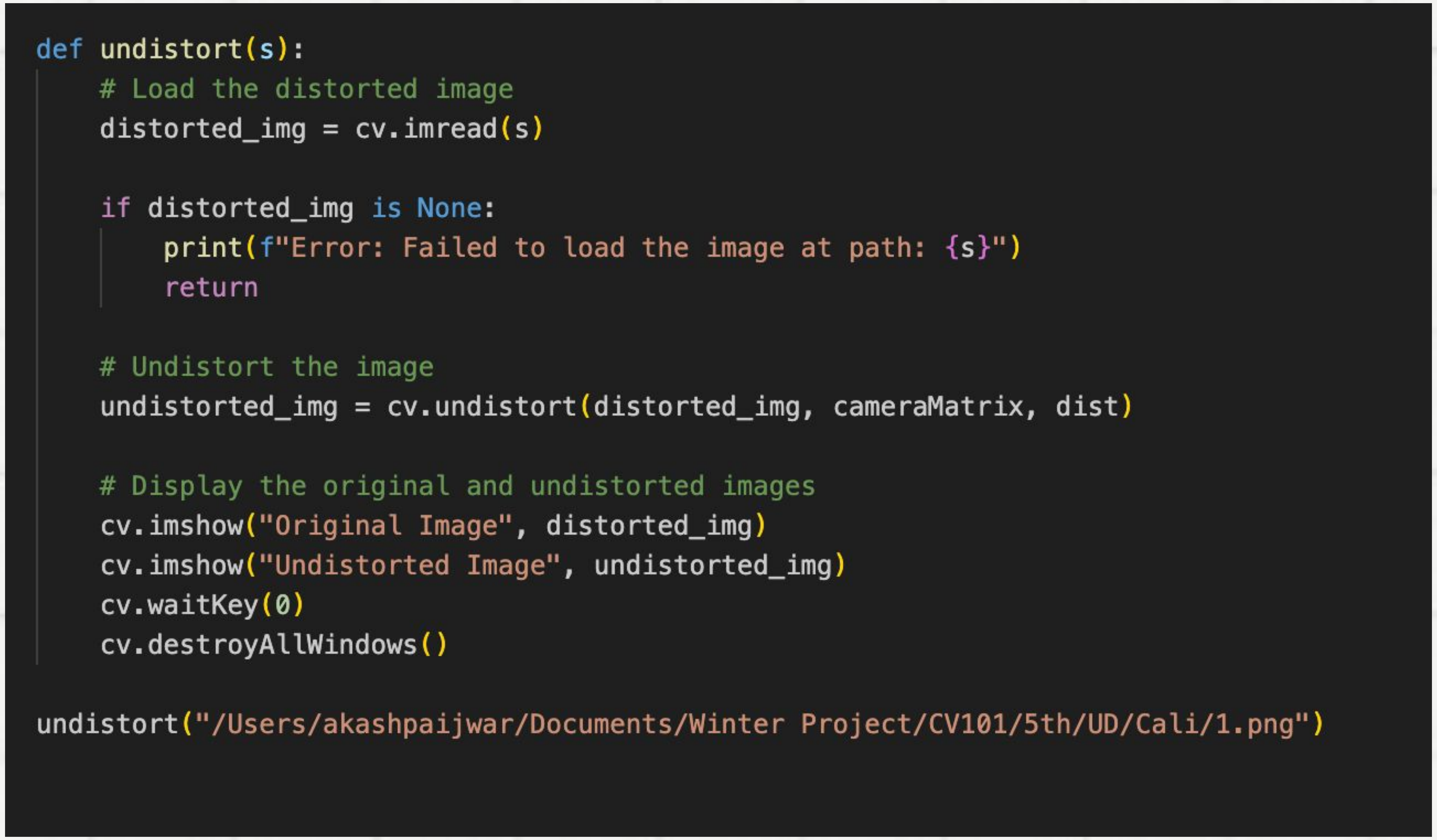
Radial Distortion: Radial Distortion is the most common type that affects the images, In which when a camera captured pictures of straight lines appeared slightly curved or bent

Tangential distortion: Tangential distortion occurs mainly because the lens is not parallely aligned to the imaging plane, that makes the image to be extended a little while longer or tilted, it makes the objects appear farther away or even closer than they actually are.

So, In order to reduce the distortion, luckily this distortion can be captured by five numbers called Distortion Coefficients, whose values reflect the amount of radial and tangential distortion in an image.

$$\text{Distortion}_{\text{coefficients}} = (k_1 \; k_2 \; p_1 \; p_2 \; k_3)$$

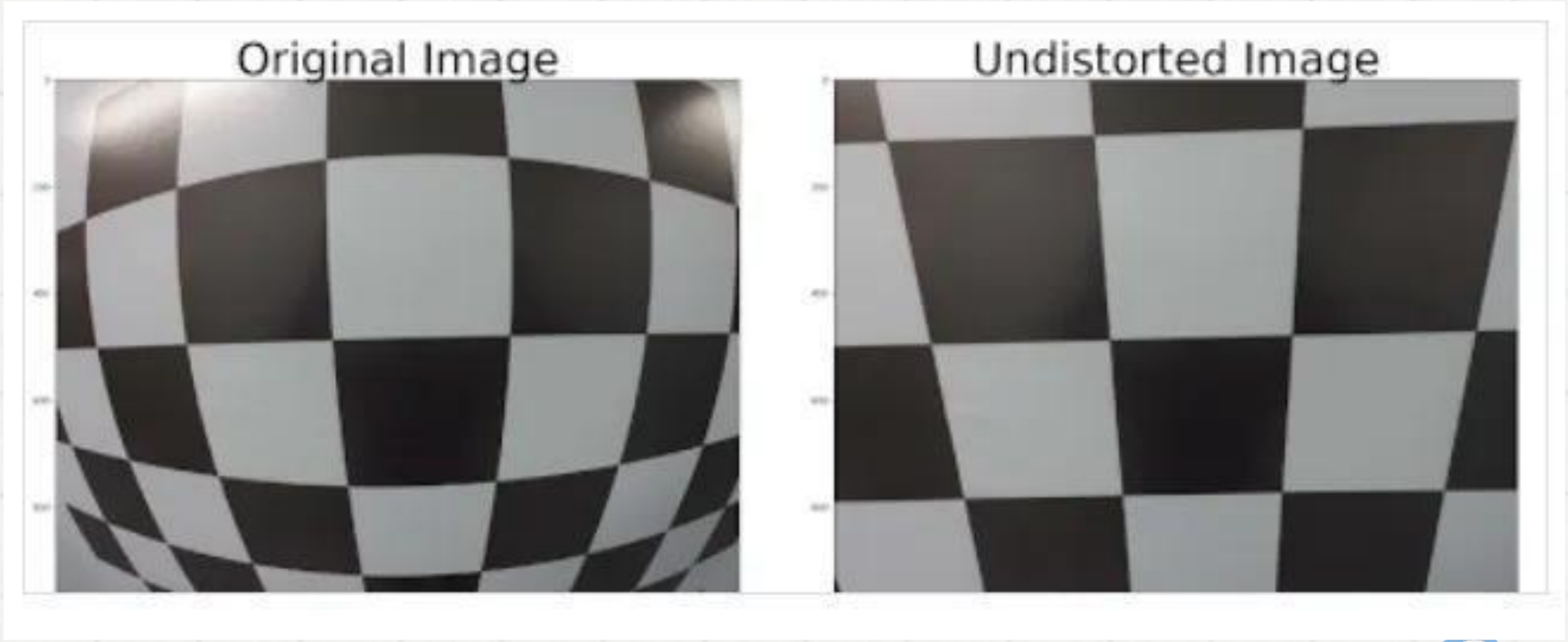
Code



```python
def undistort(s):
    # Load the distorted image
    distorted_img = cv.imread(s)

    if distorted_img is None:
        print(f"Error: Failed to load the image at path: {s}")
        return

    # Undistort the image
    undistorted_img = cv.undistort(distorted_img, cameraMatrix, dist)

    # Display the original and undistorted images
    cv.imshow("Original Image", distorted_img)
    cv.imshow("Undistorted Image", undistorted_img)
    cv.waitKey(0)
    cv.destroyAllWindows()

undistort("/Users/akashpaijwar/Documents/Winter Project/CV101/5th/UD/Cali/1.png")
```

Input

Output



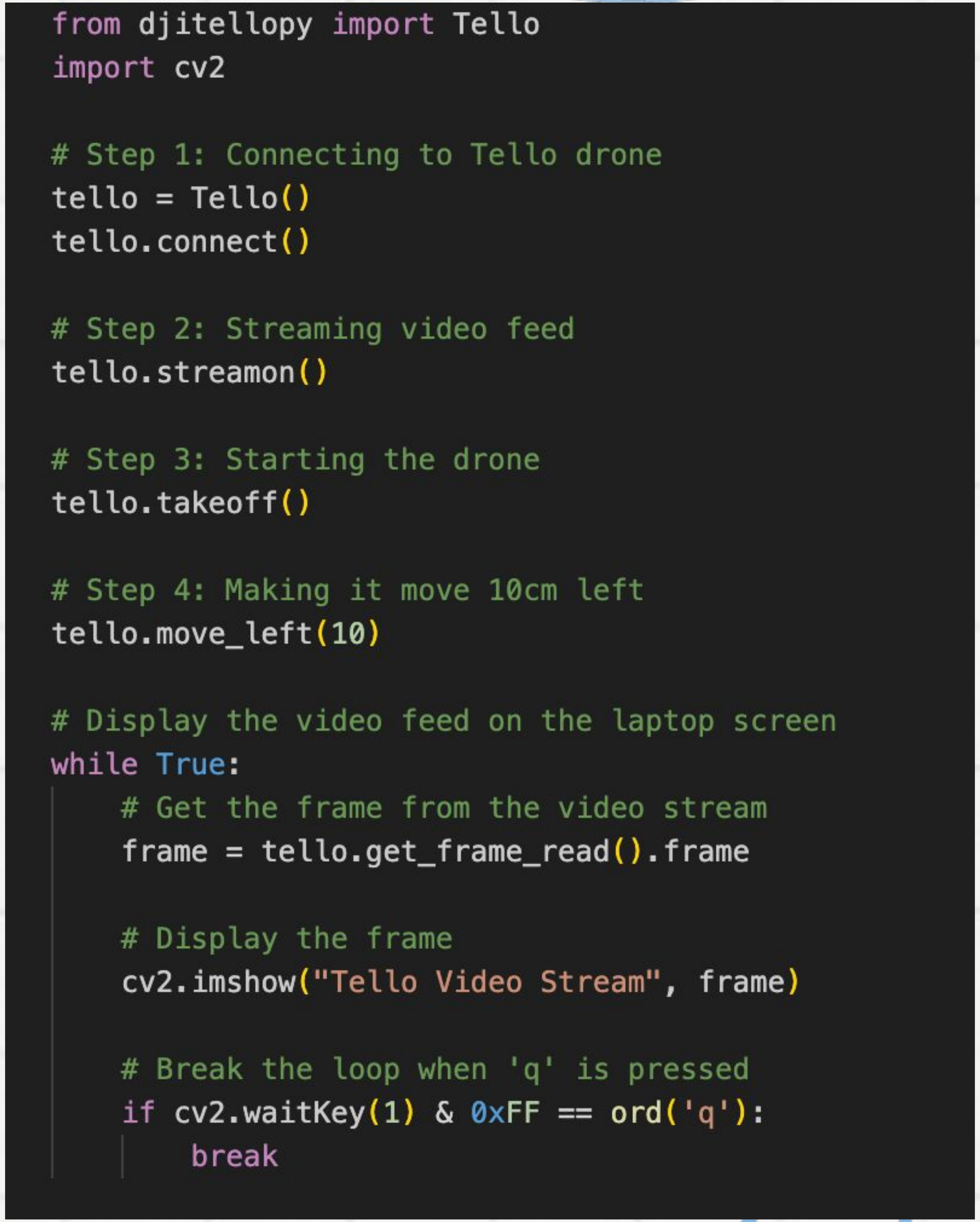
Tello Commands

1. Basic Movement:
    - takeoff: Initiates takeoff.
    - land: Initiates landing.
    - up X: Ascends to an altitude of X centimeters.
    - down X: Descends to an altitude of X centimeters.
    - left X: Moves left by X centimeters.
    - right X: Moves right by X centimeters.
    - forward X: Moves forward by X centimeters.
    - back X: Moves backward by X centimeters.
    - cw X: Rotates clockwise by an angle of X degrees.
    - ccw X: Rotates counterclockwise by an angle of X degrees.
    - flip X: Performs a flip in the specified direction (left, right, forward, backward).

1. Set Speed:
   - speed X: Sets the drone's speed to X (X can be between 1 and 100).
2. Query Status:
   - battery?: Retrieves the current battery percentage.
   - speed?: Retrieves the current speed setting.
   - time?: Retrieves the total flight time.
3. Emergency Stop:
   - emergency: Performs an emergency stop, causing the drone to land immediately.
4. Stream Control:
   - streamon: Starts video streaming.
   - streamoff: Stops video streaming.
5. Flight Path:
   - go X Y Z speed? mid?: Flies to the coordinates (X, Y, Z) at the specified speed.

Implementaion

Here we are testing fight of drone with a command to move it 10 cm left while streaming the video on our PC screen



```python
from djitellopy import Tello
import cv2

# Step 1: Connecting to Tello drone
tello = Tello()
tello.connect()

# Step 2: Streaming video feed
tello.streamon()

# Step 3: Starting the drone
tello.takeoff()

# Step 4: Making it move 10cm left
tello.move_left(10)

# Display the video feed on the laptop screen
while True:
    # Get the frame from the video stream
    frame = tello.get_frame_read().frame

    # Display the frame
    cv2.imshow("Tello Video Stream", frame)

    # Break the loop when 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

Advanced Air Mobility: The Science Behind Quadcopters

Quadcopters are a type of small aircraft that use four propellers to fly. They are part of NASA's vision for Advanced Air Mobility, a system of safe and efficient air transportation for short range missions in urban and rural locations.

How Quadcopters Fly Upward

Newton's Third Law of Motion

Newton's third law of motion states that for every action, there is an equal and opposite reaction.

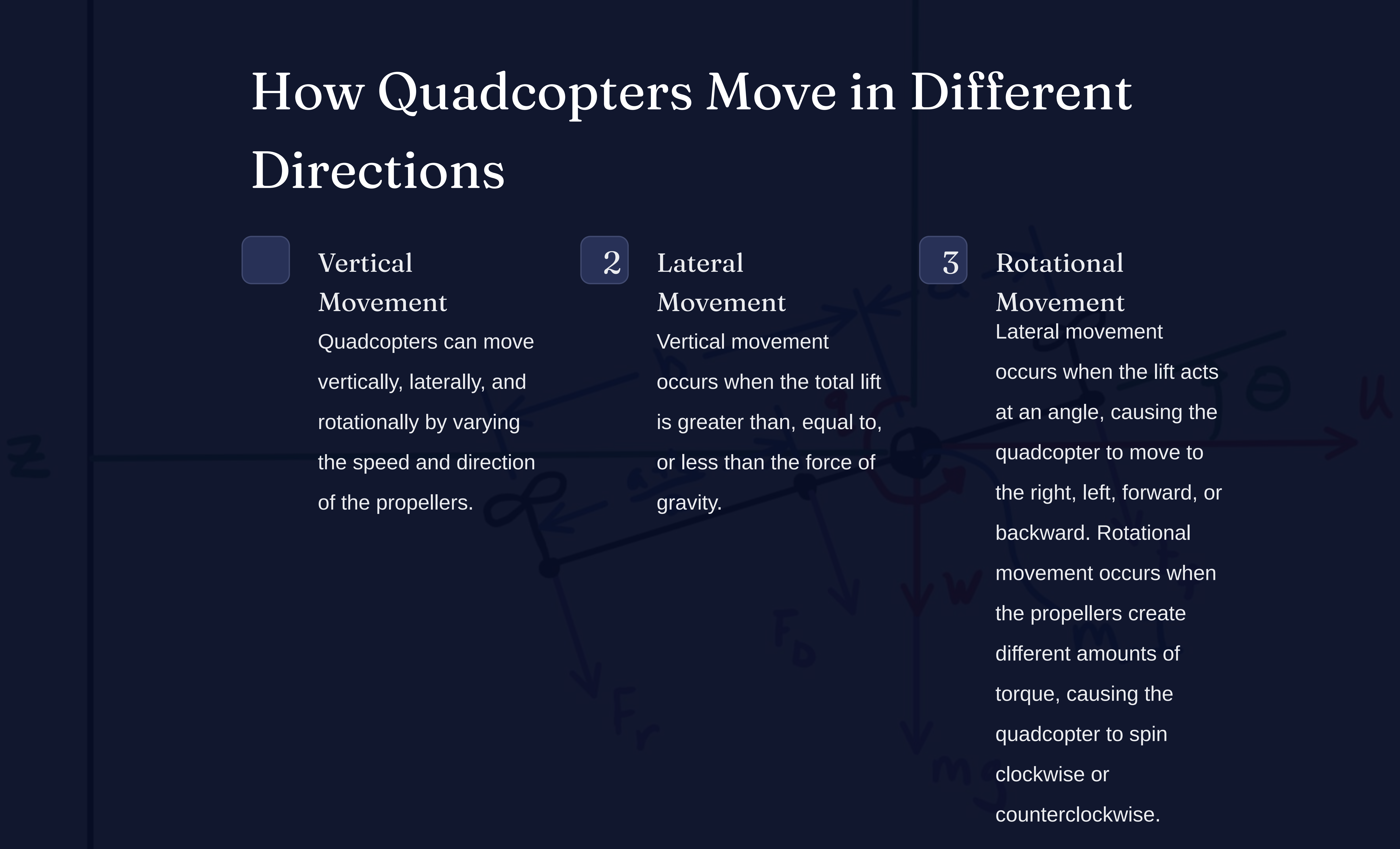
Propellers Pushing Air Downward

When a quadcopter's propellers spin, they push air downward. This is the action.

Force of Lift

The reaction is an upward force pushing on the quadcopter. This force is called lift. When the lift exceeds the force of gravity pulling the quadcopter downward, the quadcopter begins to move up.

How Quadcopters Move in Different Directions



Vertical Movement

Quadcopters can move vertically, laterally, and rotationally by varying the speed and direction of the propellers.

2 Lateral Movement

Vertical movement occurs when the total lift is greater than, equal to, or less than the force of gravity.

3 Rotational Movement

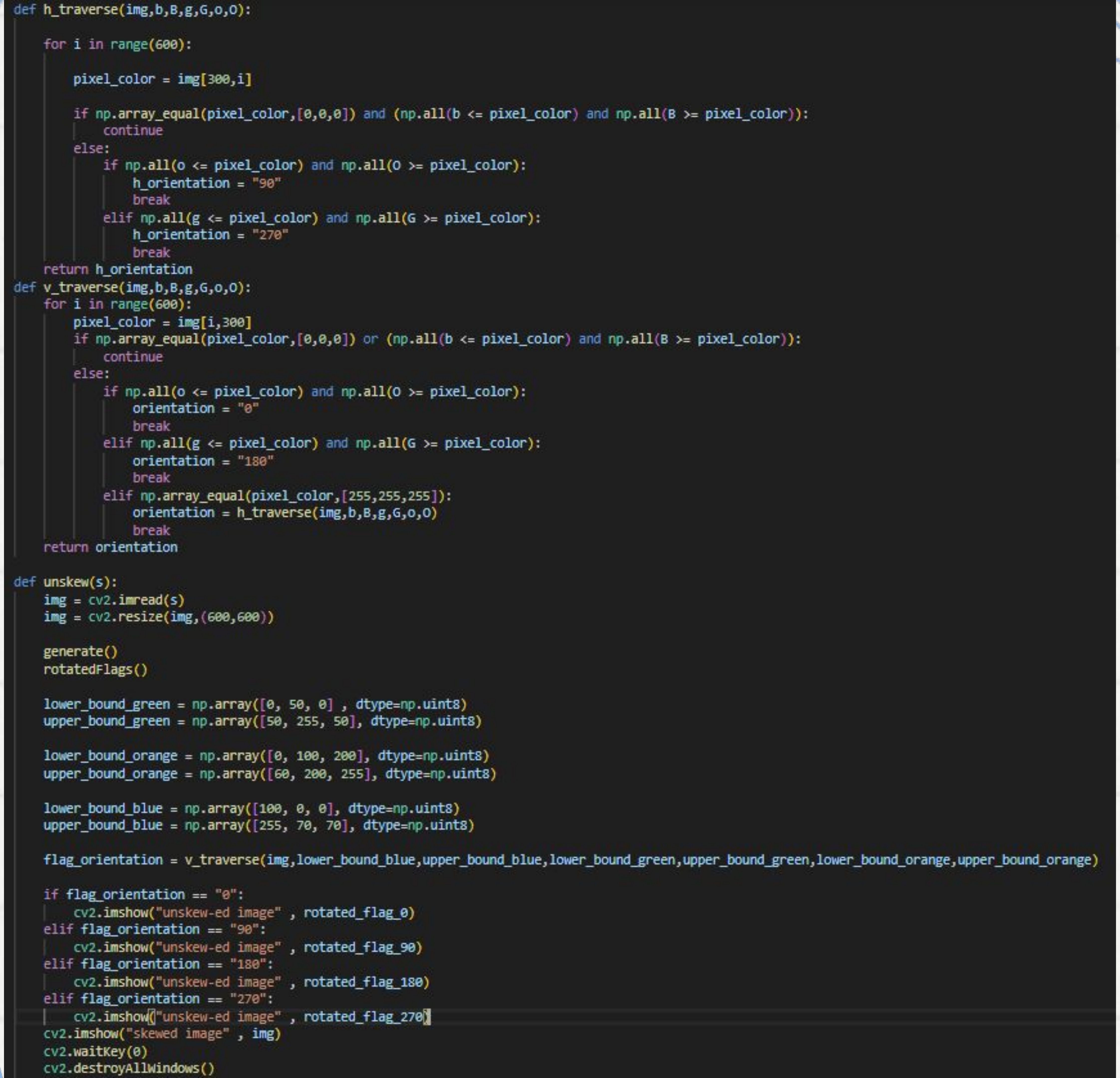
Lateral movement occurs when the lift acts at an angle, causing the quadcopter to move to the right, left, forward, or backward. Rotational movement occurs when the propellers create different amounts of torque, causing the quadcopter to spin clockwise or counterclockwise.

Unscewing Of Image

Skewed images are common in various scenarios and can impact the accuracy of computer vision tasks
We Utilizes the color orientation detection for precise skew angle identification.

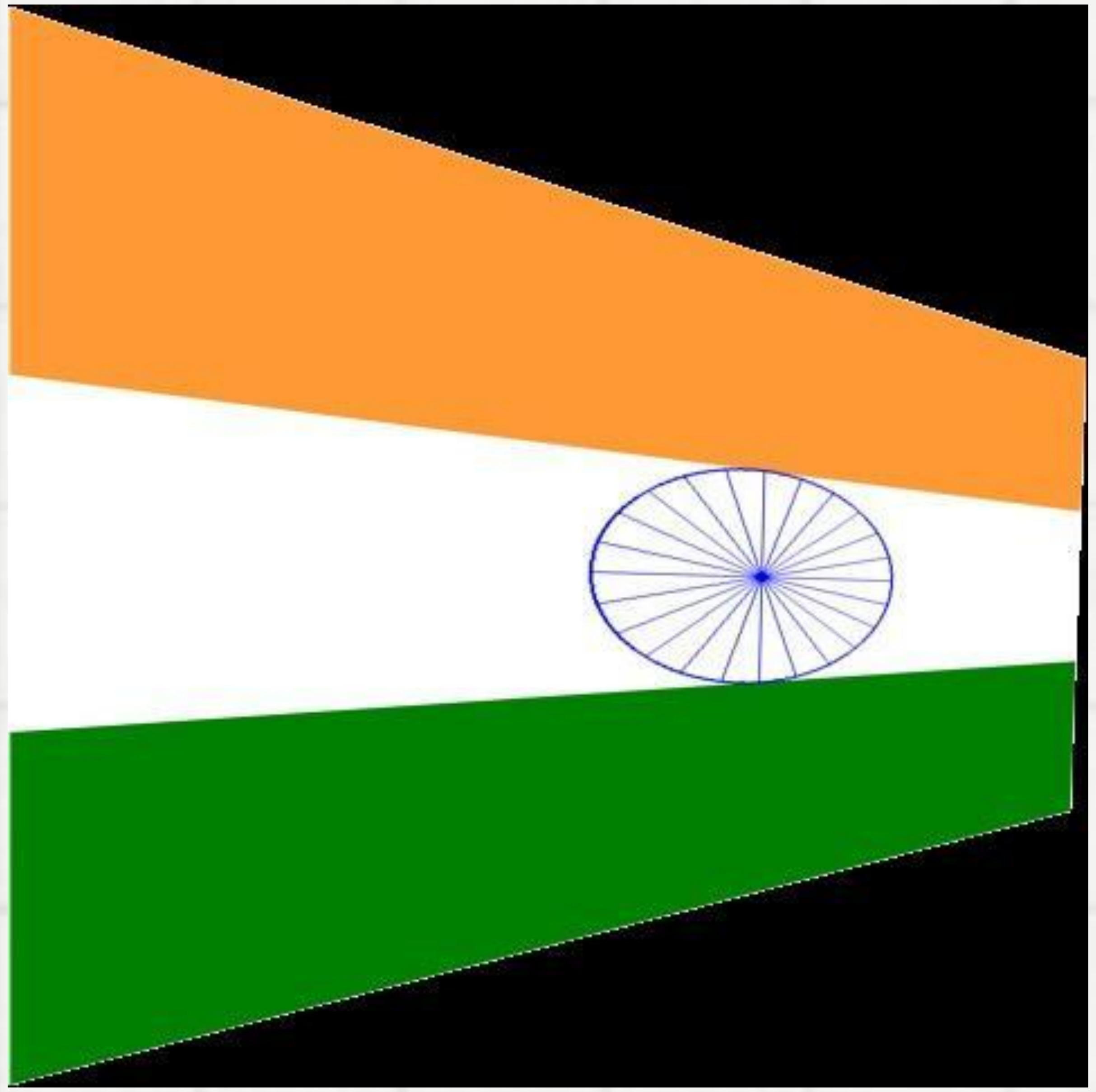
Here Color traversal along horizontal and vertical midline pixel is used to identify the orientation, ensuring accurate unskewing.
Then we employs a rotation function to unskew images, enhancing their visual representation.

Unskewing is done because it improves image alignment for better interpretation and downstream processing

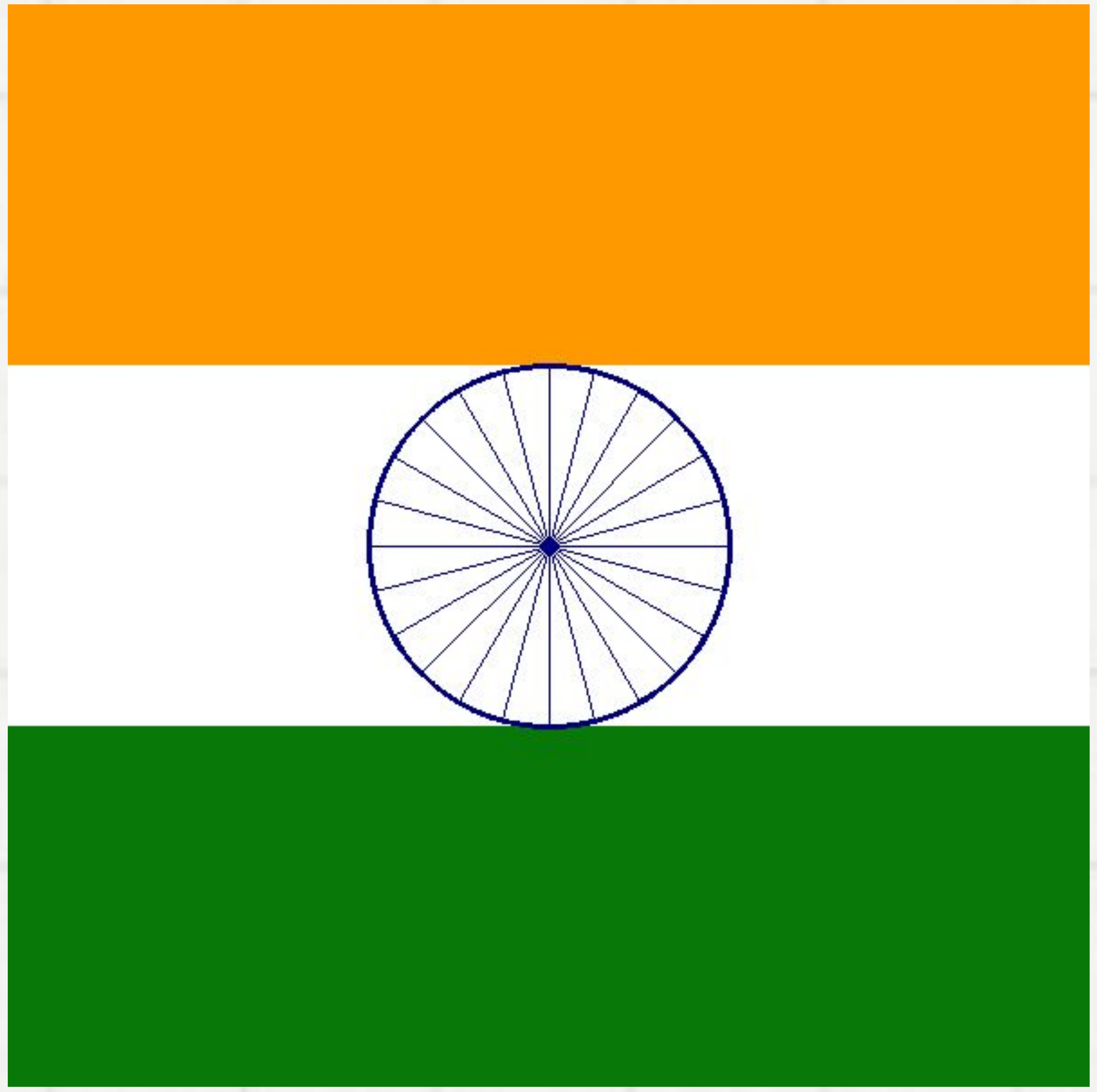


```python
def h_traverse(img,b,B,g,G,o,O):

    for i in range(600):

        pixel_color = img[300,i]

        if np.array_equal(pixel_color,[0,0,0]) and (np.all(b <= pixel_color) and np.all(B >= pixel_color)):
            continue
        else:
            if np.all(o <= pixel_color) and np.all(O >= pixel_color):
                h_orientation = "90"
                break
            elif np.all(g <= pixel_color) and np.all(G >= pixel_color):
                h_orientation = "270"
                break
    return h_orientation
def v_traverse(img,b,B,g,G,o,O):
    for i in range(600):
        pixel_color = img[i,300]
        if np.array_equal(pixel_color,[0,0,0]) or (np.all(b <= pixel_color) and np.all(B >= pixel_color)):
            continue
        else:
            if np.all(o <= pixel_color) and np.all(O >= pixel_color):
                orientation = "0"
                break
            elif np.all(g <= pixel_color) and np.all(G >= pixel_color):
                orientation = "180"
                break
            elif np.array_equal(pixel_color,[255,255,255]):
                orientation = h_traverse(img,b,B,g,G,o,O)
                break
    return orientation

def unskew(s):
    img = cv2.imread(s)
    img = cv2.resize(img,(600,600))

    generate()
    rotatedFlags()

    lower_bound_green = np.array([0, 50, 0] , dtype=np.uint8)
    upper_bound_green = np.array([50, 255, 50], dtype=np.uint8)

    lower_bound_orange = np.array([0, 100, 200], dtype=np.uint8)
    upper_bound_orange = np.array([60, 200, 255], dtype=np.uint8)

    lower_bound_blue = np.array([100, 0, 0], dtype=np.uint8)
    upper_bound_blue = np.array([255, 70, 70], dtype=np.uint8)

    flag_orientation = v_traverse(img,lower_bound_blue,upper_bound_blue,lower_bound_green,upper_bound_green,lower_bound_orange,upper_bound_orange)

    if flag_orientation == "0":
        cv2.imshow("unskew-ed image" , rotated_flag_0)
    elif flag_orientation == "90":
        cv2.imshow("unskew-ed image" , rotated_flag_90)
    elif flag_orientation == "180":
        cv2.imshow("unskew-ed image" , rotated_flag_180)
    elif flag_orientation == "270":
        cv2.imshow("unskew-ed image" , rotated_flag_270)
    cv2.imshow("skewed image" , img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

INPUT



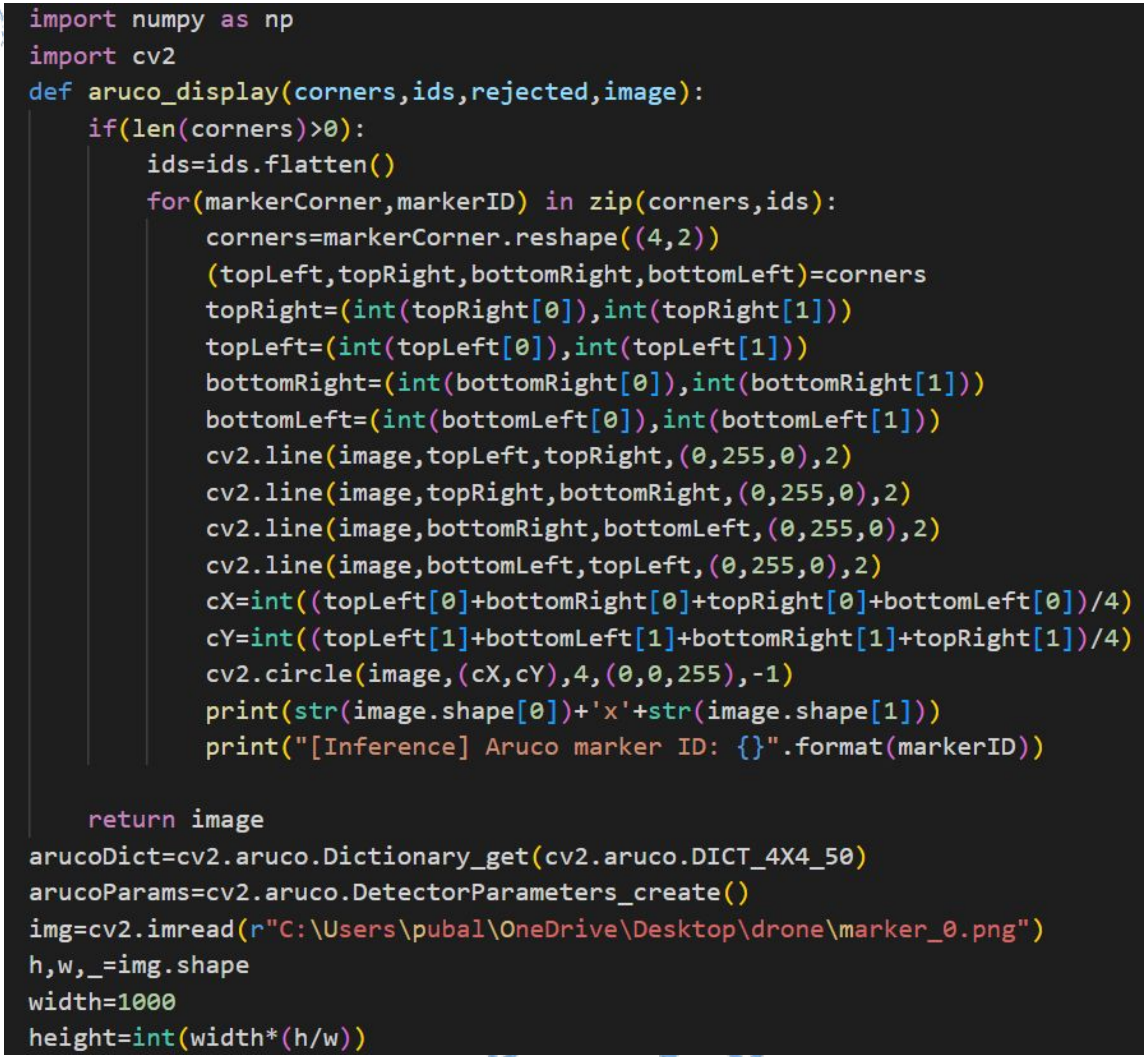
OUTPUT

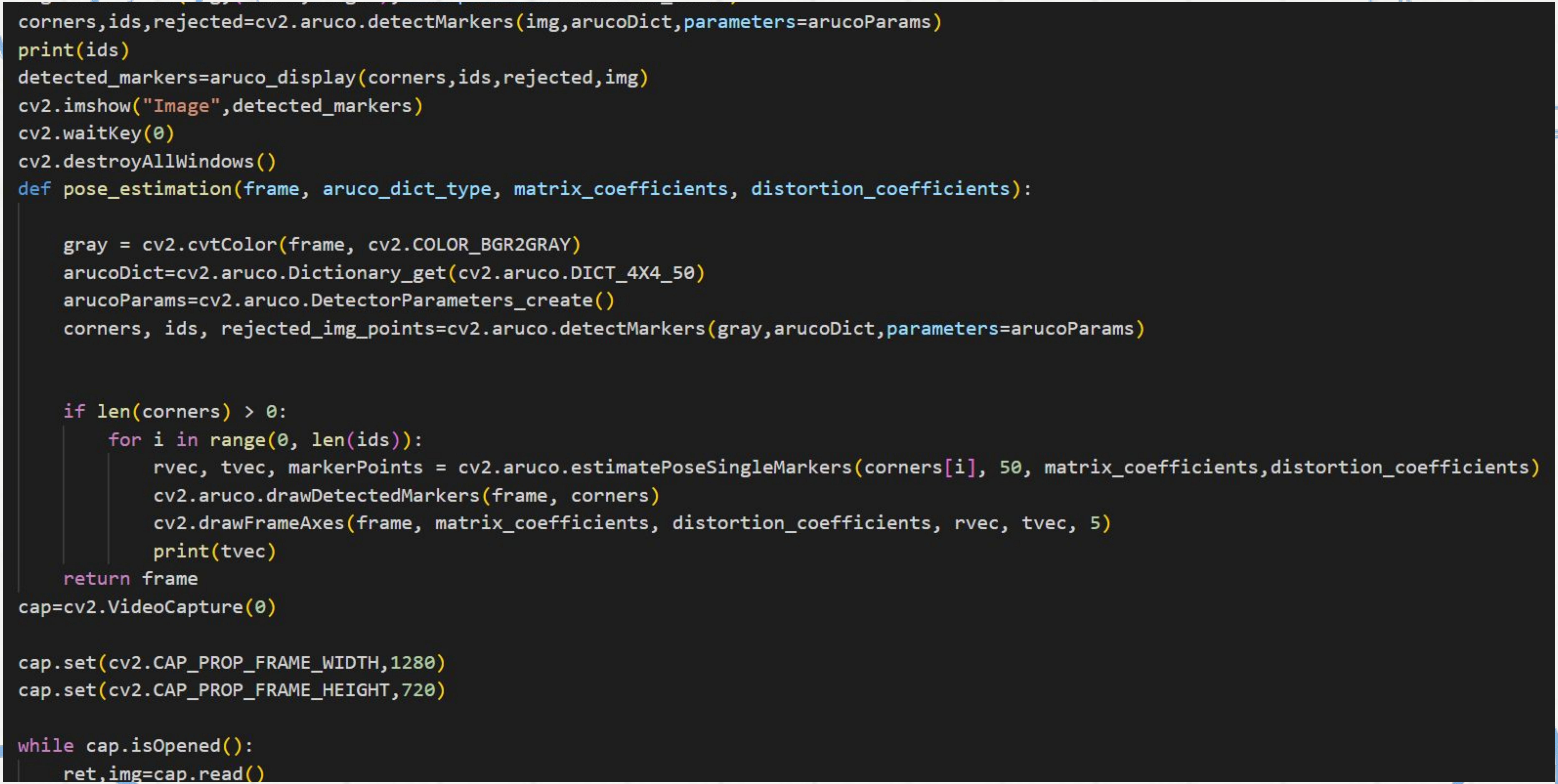


Pose Estimation

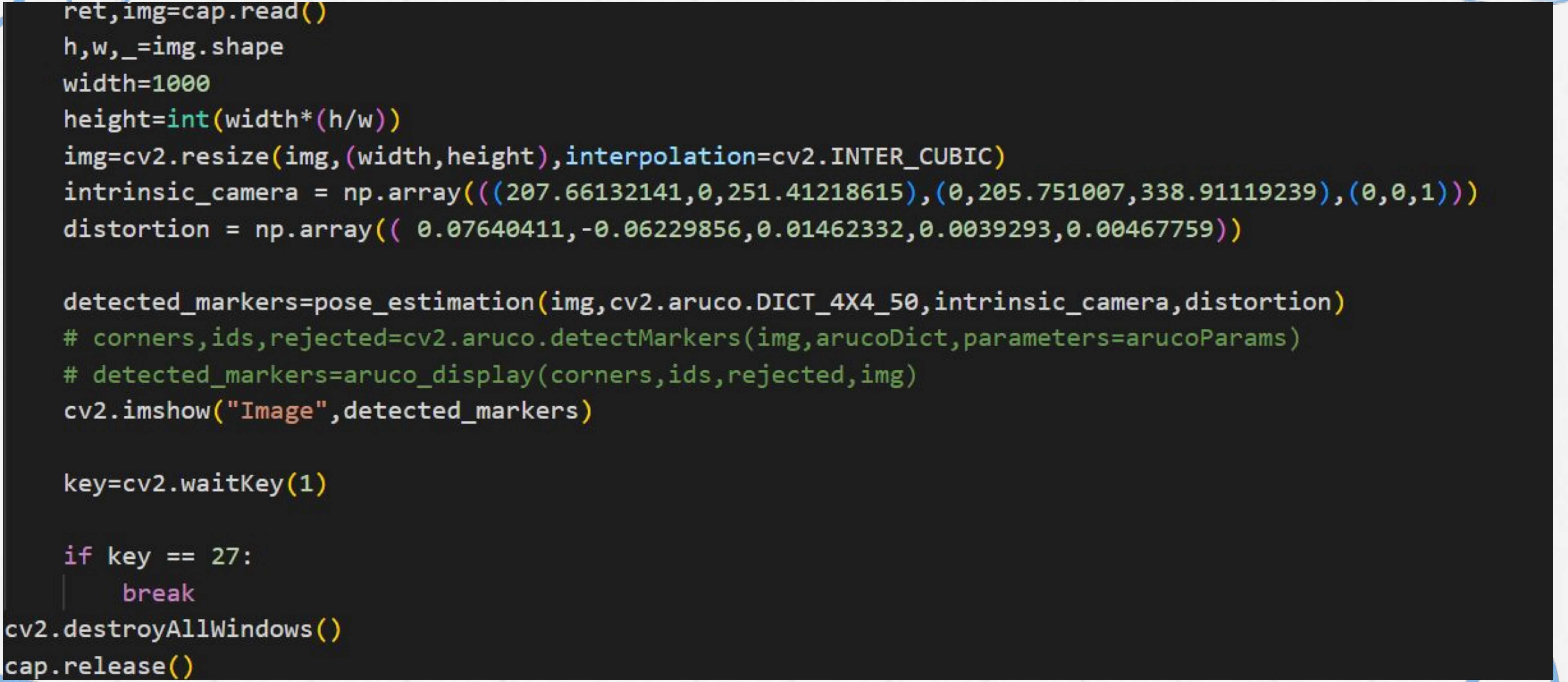
Pose estimation is a computer vision task that involves determining the spatial positions of key points or joints on a person or object in images or videos. These key points typically correspond to specific anatomical landmarks, such as joints or other distinctive features. The primary objective of pose estimation is to understand and represent the orientation, position, and configuration of the detected object or person.

It also has diverse applications across various domains, including human-computer interaction, augmented reality, healthcare, sports analytics, and security. It plays a crucial role in understanding and interpreting the body language, movements, and gestures of humans or objects in visual data.


```python
import numpy as np
import cv2
def aruco_display(corners,ids,rejected,image):
    if(len(corners)>0):
        ids=ids.flatten()
        for(markerCorner,markerID) in zip(corners,ids):
            corners=markerCorner.reshape((4,2))
            (topLeft,topRight,bottomRight,bottomLeft)=corners
            topRight=(int(topRight[0]),int(topRight[1]))
            topLeft=(int(topLeft[0]),int(topLeft[1]))
            bottomRight=(int(bottomRight[0]),int(bottomRight[1]))
            bottomLeft=(int(bottomLeft[0]),int(bottomLeft[1]))
            cv2.line(image,topLeft,topRight,(0,255,0),2)
            cv2.line(image,topRight,bottomRight,(0,255,0),2)
            cv2.line(image,bottomRight,bottomLeft,(0,255,0),2)
            cv2.line(image,bottomLeft,topLeft,(0,255,0),2)
            cX=int((topLeft[0]+bottomRight[0]+topRight[0]+bottomLeft[0])/4)
            cY=int((topLeft[1]+bottomLeft[1]+bottomRight[1]+topRight[1])/4)
            cv2.circle(image,(cX,cY),4,(0,0,255),-1)
            print(str(image.shape[0])+'x'+str(image.shape[1]))
            print("[Inference] Aruco marker ID: {}".format(markerID))

    return image
arucoDict=cv2.aruco.Dictionary_get(cv2.aruco.DICT_4X4_50)
arucoParams=cv2.aruco.DetectorParameters_create()
img=cv2.imread(r"C:\Users\pubal\OneDrive\Desktop\drone\marker_0.png")
h,w,_=img.shape
width=1000
height=int(width*(h/w))
```



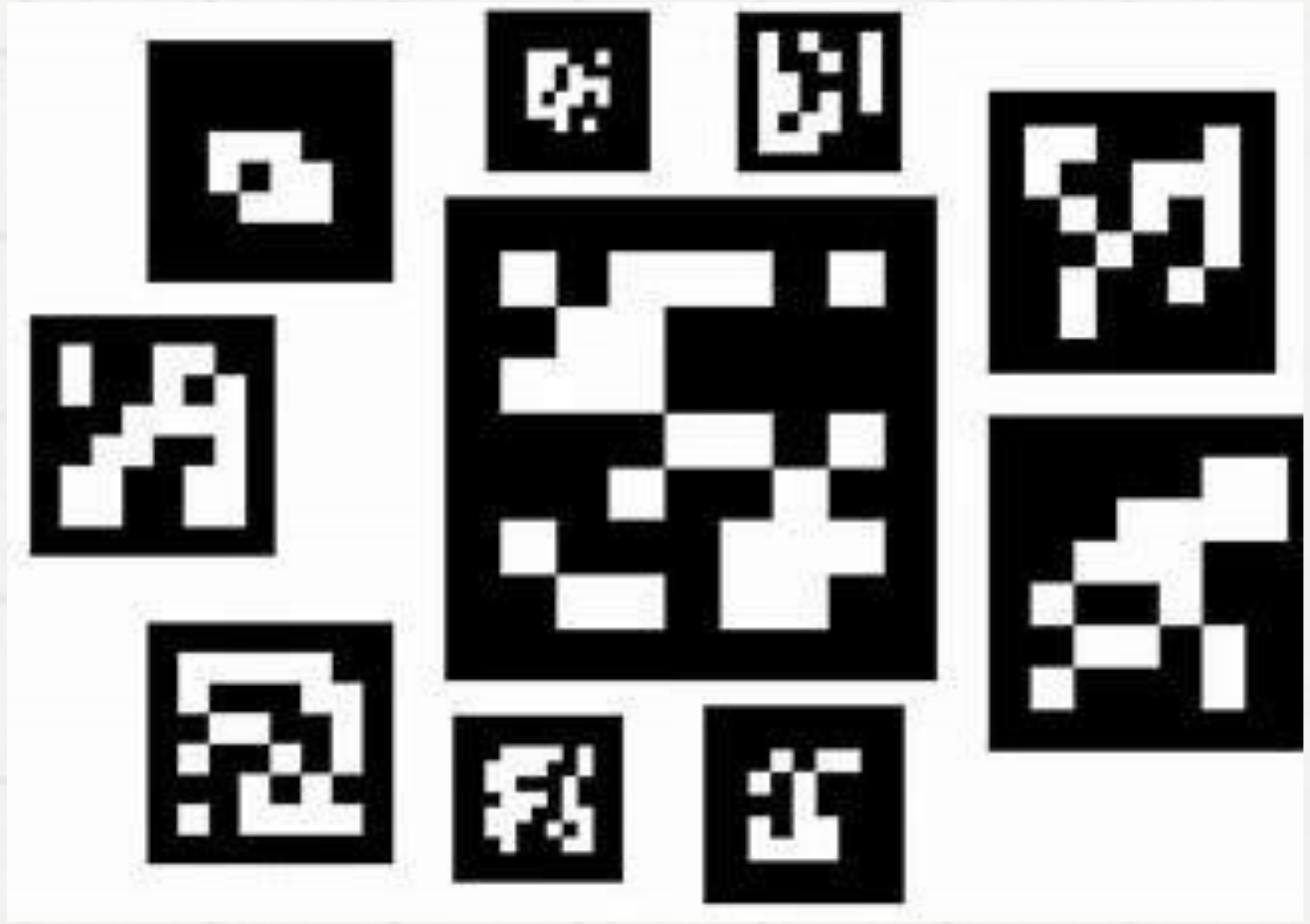
```python
corners,ids,rejected=cv2.aruco.detectMarkers(img,arucoDict,parameters=arucoParams)
print(ids)
detected_markers=aruco_display(corners,ids,rejected,img)
cv2.imshow("Image",detected_markers)
cv2.waitKey(0)
cv2.destroyAllWindows()
def pose_estimation(frame, aruco_dict_type, matrix_coefficients, distortion_coefficients):

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    arucoDict=cv2.aruco.Dictionary_get(cv2.aruco.DICT_4X4_50)
    arucoParams=cv2.aruco.DetectorParameters_create()
    corners, ids, rejected_img_points=cv2.aruco.detectMarkers(gray,arucoDict,parameters=arucoParams)


    if len(corners) > 0:
        for i in range(0, len(ids)):
            rvec, tvec, markerPoints = cv2.aruco.estimatePoseSingleMarkers(corners[i], 50, matrix_coefficients,distortion_coefficients)
            cv2.aruco.drawDetectedMarkers(frame, corners)
            cv2.drawFrameAxes(frame, matrix_coefficients, distortion_coefficients, rvec, tvec, 5)
            print(tvec)
    return frame
cap=cv2.VideoCapture(0)

cap.set(cv2.CAP_PROP_FRAME_WIDTH,1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT,720)

while cap.isOpened():
    ret,img=cap.read()
```



```
    ret,img=cap.read()
    h,w,_=img.shape
    width=1000
    height=int(width*(h/w))
    img=cv2.resize(img,(width,height),interpolation=cv2.INTER_CUBIC)
    intrinsic_camera = np.array(((207.66132141,0,251.41218615),(0,205.751007,338.91119239),(0,0,1)))
    distortion = np.array(( 0.07640411,-0.06229856,0.01462332,0.0039293,0.00467759))

    detected_markers=pose_estimation(img,cv2.aruco.DICT_4X4_50,intrinsic_camera,distortion)
    # corners,ids,rejected=cv2.aruco.detectMarkers(img,arucoDict,parameters=arucoParams)
    # detected_markers=aruco_display(corners,ids,rejected,img)
    cv2.imshow("Image",detected_markers)

    key=cv2.waitKey(1)

    if key == 27:
        break
cv2.destroyAllWindows()
cap.release()
```

ArUco Markers

ArUco markers are a type of augmented reality (AR) markers that are widely used in computer vision applications. These markers are implanted with a unique identifying code and feature square black-and-white patterns. The ArUco library, an open-source library for detecting these markers, is frequently linked to the ArUco marker system.

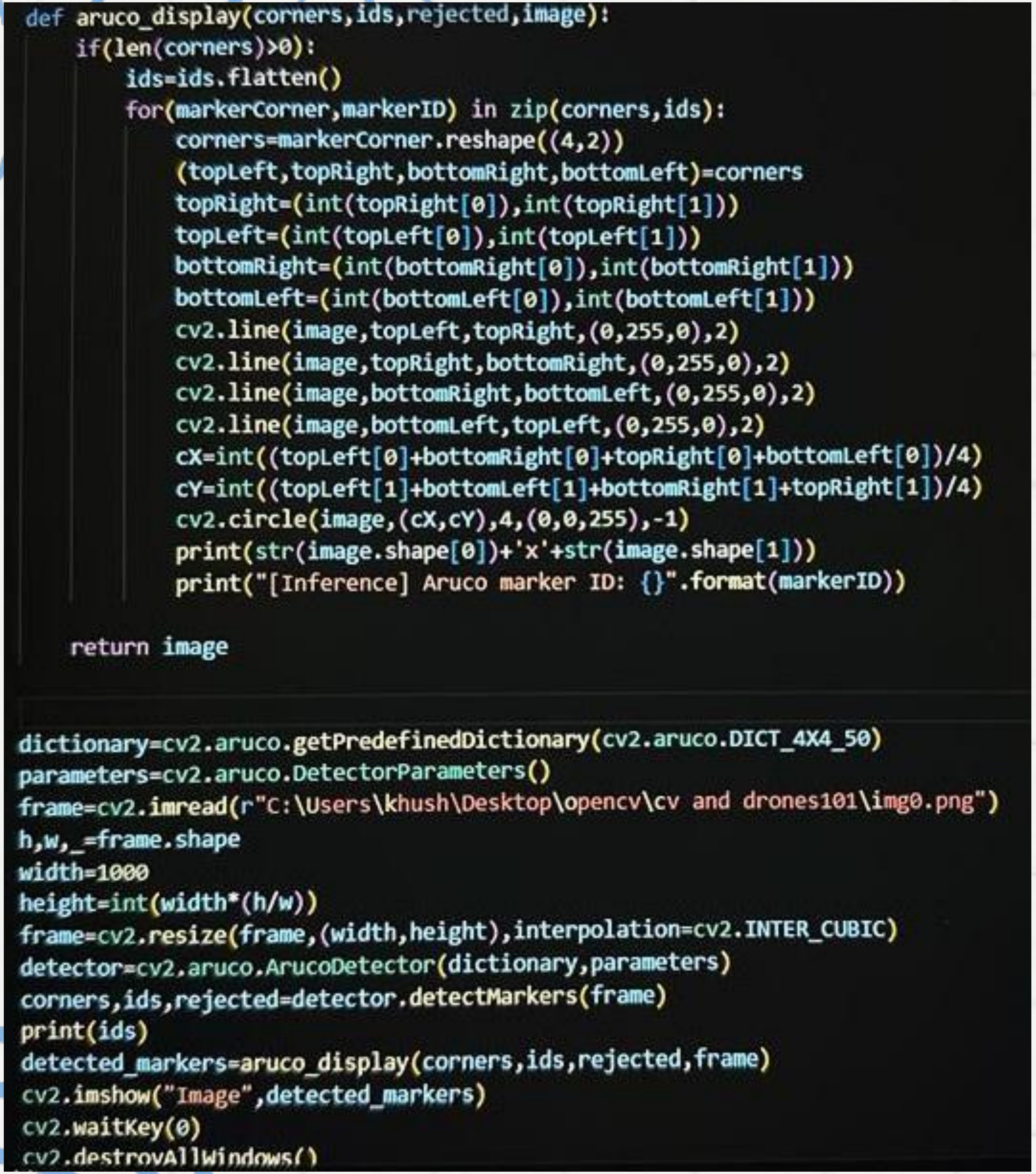


Detection Of Markers

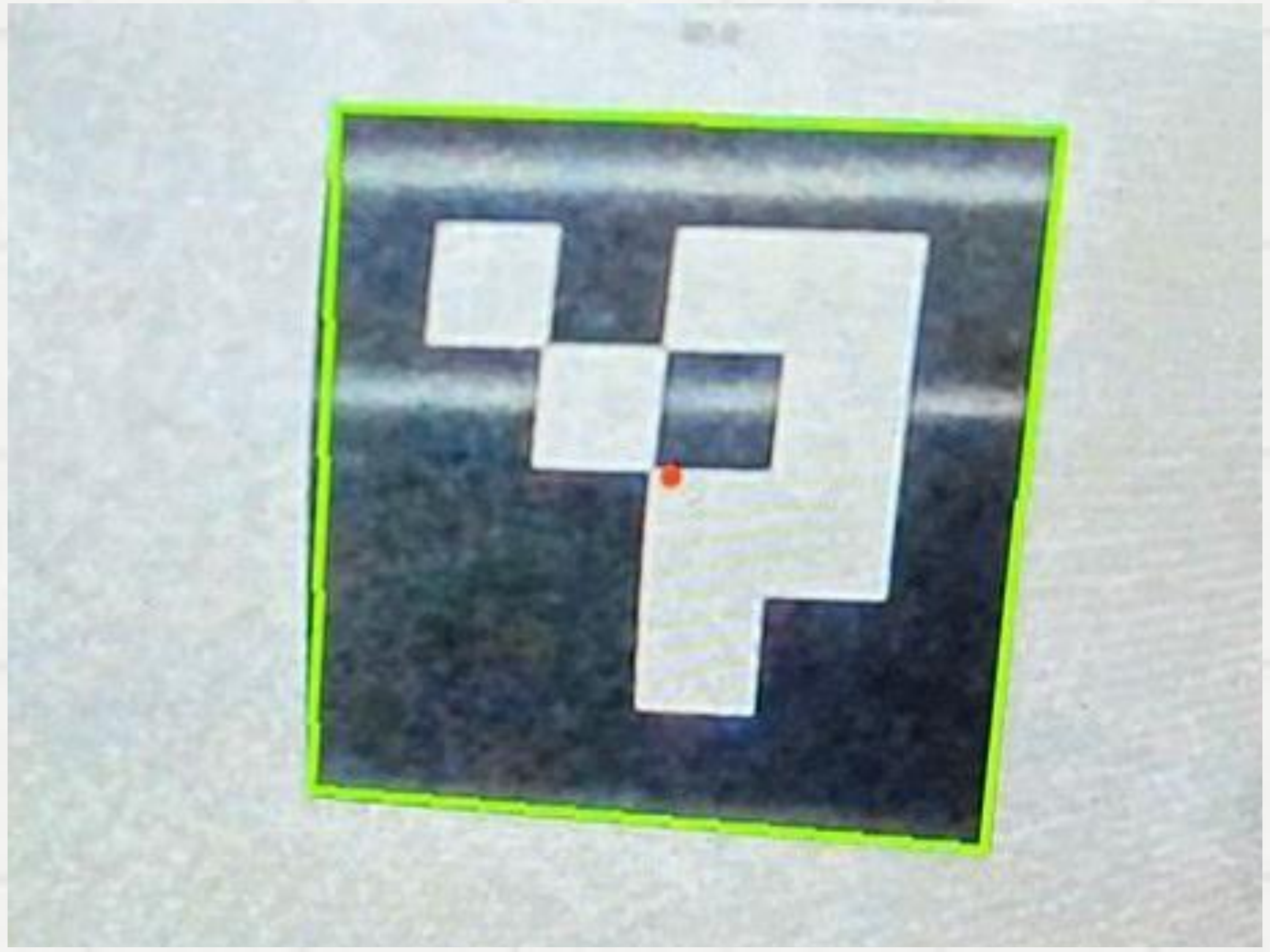
It involves using a computer vision library or algorithm to detect ArUco markers in the images. The ArUco library is a popular open-source library for this purpose. The main steps involved are:

- Thresholding: Conversion of the image to a binary image
- Contour Detection: Identify contours in the binary image.
- Corner Detection: Extract corners from the contours.
- Marker Identification: Check to see if the corner pattern corresponds with a recognized ArUco marking.

Example



```
def aruco_display(corners,ids,rejected,image):
    if(len(corners)>0):
        ids=ids.flatten()
        for(markerCorner,markerID) in zip(corners,ids):
            corners=markerCorner.reshape((4,2))
            (topLeft,topRight,bottomRight,bottomLeft)=corners
            topRight=(int(topRight[0]),int(topRight[1]))
            topLeft=(int(topLeft[0]),int(topLeft[1]))
            bottomRight=(int(bottomRight[0]),int(bottomRight[1]))
            bottomLeft=(int(bottomLeft[0]),int(bottomLeft[1]))
            cv2.line(image,topLeft,topRight,(0,255,0),2)
            cv2.line(image,topRight,bottomRight,(0,255,0),2)
            cv2.line(image,bottomRight,bottomLeft,(0,255,0),2)
            cv2.line(image,bottomLeft,topLeft,(0,255,0),2)
            cX=int((topLeft[0]+bottomRight[0]+topRight[0]+bottomLeft[0])/4)
            cY=int((topLeft[1]+bottomLeft[1]+bottomRight[1]+topRight[1])/4)
            cv2.circle(image,(cX,cY),4,(0,0,255),-1)
            print(str(image.shape[0])+'x'+str(image.shape[1]))
            print("[Inference] Aruco marker ID: {}".format(markerID))

    return image

dictionary=cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_4X4_50)
parameters=cv2.aruco.DetectorParameters()
frame=cv2.imread(r"C:\Users\khush\Desktop\opencv\cv and drones101\img0.png")
h,w,_=frame.shape
width=1000
height=int(width*(h/w))
frame=cv2.resize(frame,(width,height),interpolation=cv2.INTER_CUBIC)
detector=cv2.aruco.ArucoDetector(dictionary,parameters)
corners,ids,rejected=detector.detectMarkers(frame)
print(ids)
detected_markers=aruco_display(corners,ids,rejected,frame)
cv2.imshow("Image",detected_markers)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



PID CONTROL

A PID Controller or a Proportional-Integral-Derivative controller is a control loop feedback mechanism (controller) widely used in industrial control systems.
A PID controller calculates an error value as the difference between a measured process variable and a desired setpoint (desired outcome). The controller attempts to minimize the error by adjusting the process through the use of a manipulated variable.
The PID controller algorithm involves three separate constant parameters, and is accordingly sometimes called three-term control:
Proportional (P)
Integral (I)
Derivative (D)
Simply put, these values can be interpreted in terms of time: ‘P’ depends on the present error, ‘I’ on the accumulation of past errors, and ‘D’ is a prediction of future errors, based on the current rate of change. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve, a damper, or the power supplied to a heating element.

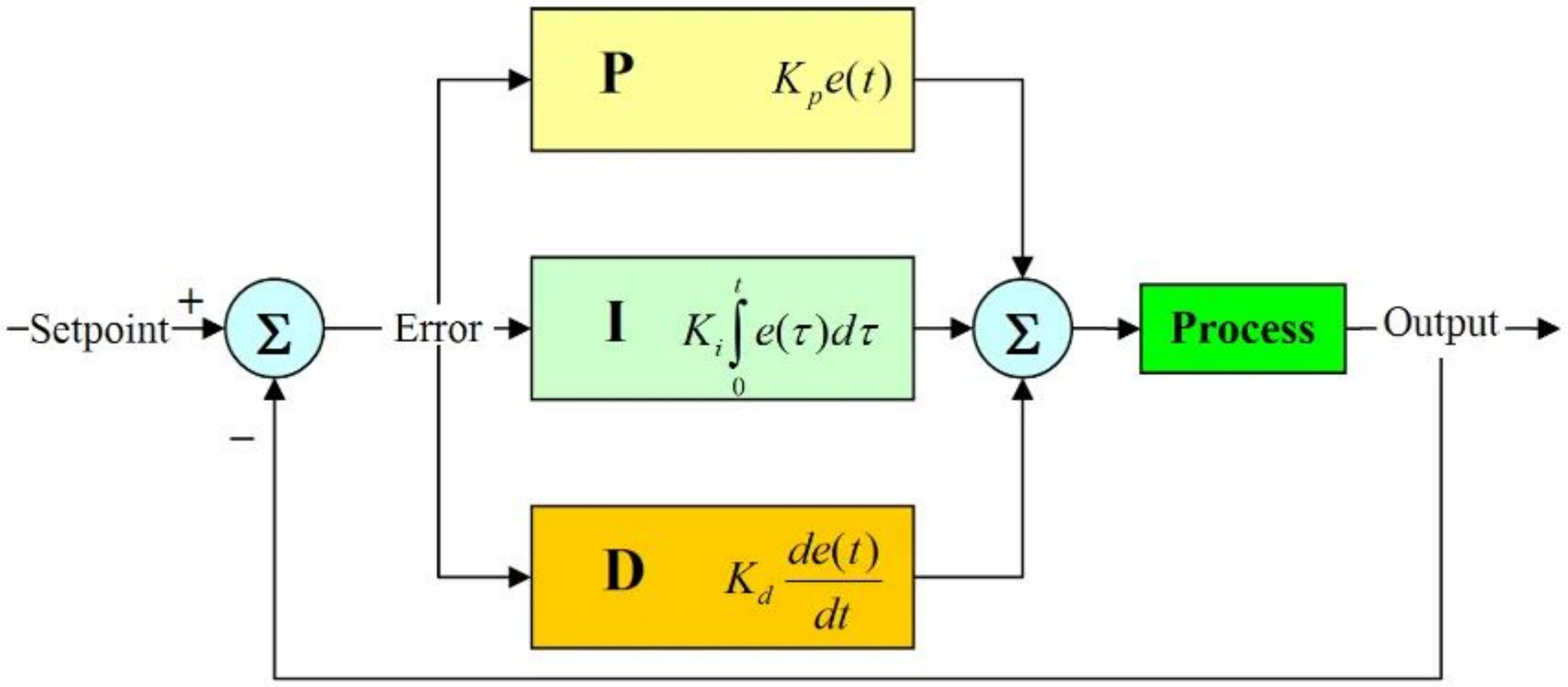
The basic terminology that one would require to understand PID are:

- Error: The error is the amount at which a device isn't doing something right. For example, suppose the robot is located at x=5 but it should be at x=7, then the error is 2.
- Proportional (P): The proportional term is directly proportional to the error at present.
- Integral (I): The integral term depends on the cumulative error made over a period of time (t).
- Derivative (D): The derivative term depends on rate of change of error.
- P-Factor (Kp): A constant value used to increase or decrease the impact of Proportional
- I-Factor (Ki): A constant value used to increase or decrease the impact of Integral
- D-Factor (Kd): A constant value used to increase or decrease the impact of Derivative

Note- Each term (P, I, D) will need to be tweaked in the code. Hence, they are included in the code by multiplying with respective constant factors.

The PID control scheme is named after its three correcting terms, whose sum constitutes the manipulated variable (MV). The proportional, integral, and derivative terms are summed to calculate the output of the PID controller. Defining as the controller output, the final form of the PID algorithm is:

$$\mathrm{u}(t) = \mathrm{MV}(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{d}{dt} e(t)$$


P
$K_p e(t)$
I
$K_i \int_0^t e(\tau) d\tau$
D
$K_d \frac{de(t)}{dt}$
Setpoint
+
−
Σ
Error
Σ
Process
Output