**Department of Electrical Engineering and   
Computer Science**

**Faculty Member:** Dr. Mohsin Kamal **Dated:** 05/10/2023

**Semester:** 7th **Section:** BEE 12C

**CS-477 Computer Vision**

Lab 4: OpenCV Basic Image Processing Functions

**Group Members**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **PLO4 - CLO4** | | **PLO5 -CLO5** | **PLO8 -CLO6** | **PLO9 -CLO7** |
| **Name** | **Reg. No** | **Viva / Quiz / Lab Performance** | **Analysis of Data in Lab Report** | **Modern Tool Usage** | **Ethics and Safety** | **Individual and Teamwork** |
|  |  | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** |
| Muhammad Ahmed Mohsin | 333060 |  |  |  |  |  |
| Muhammad Umer | 345834 |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Table of Contents**

[2 OpenCV Basic Image Processing Functions 3](#_Toc147490504)

[2.1 Introduction 3](#_Toc147490505)

[2.2 Objectives 3](#_Toc147490506)

[2.3 Software 3](#_Toc147490507)

[3 Lab Tasks 4](#_Toc147490508)

[3.1 Task 1 – Gaussian Blur and Canny Edges 4](#_Toc147490509)

[3.2 Task 2 – Bitwise Operations 5](#_Toc147490510)

[3.3 Task 3 – HSV Color Space and InRange Function 6](#_Toc147490511)

[3.4 Task 4 – Centroids 7](#_Toc147490512)

[3.5 Task 5 – Perspective Transformation 9](#_Toc147490513)

[3.6 Task 6 – Pixel Replacement 10](#_Toc147490514)

[4 Conclusion 11](#_Toc147490515)

# OpenCV Basic Image Processing Functions

## Introduction

This laboratory exercise will focus on additional concepts pertaining to OpenCV which was introduced in the previous lab. OpenCV is a popular and widely used library for image processing and computer vision applications. OpenCV contains a wide selection of functions for vision-based algorithms. These functions range from basic preprocessing such as blurring, edge-detection, thresholding etc. to computer vision implementations such as image stitching, template matching and homogeneous transform etc.

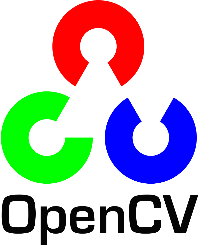
## Objectives

The following are the main objectives of this lab:

* Implement Gaussian blur and Canny edge detection on images
* Use bitwise operations on images
* Determine centroid points in images
* Use HSV color space to isolate colors in an image
* Use perspective transformation between corresponding points

## Software

OpenCV is a library that focuses on image processing and computer vision. An image is an array of colored squares called pixels. Each pixel has a certain location in the array and color values in BGR format. By referring to the array indices, the individual pixels or a range of pixels can be accessed and modified. OpenCV provides many functions for resizing, rotating, and placing objects in images. Rotation involves computing a 2-D rotation matrix which is applied for the transformation of the image.



# Lab Tasks

## Task 1 – Gaussian Blur and Canny Edges

Load the house.jpg file for this task. Use the Gaussian blur function to blur at least one of the windows. Use 13x13 size kernel for the blur.

Additionally, you will need to apply Canny edge detection on the original house image. You must apply 2 of these edge detections; one on the original picture and the second on a slightly blurred picture. Use same threshold values on both edge detections.

### TASK 1 CODE STARTS HERE ###

img = cv2.imread('house.jpg')

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

*# Apply Canny edge detection on a slightly blurred picture*

blurred = cv2.GaussianBlur(img, (13, 13), 0)

*# Apply Canny edge detection on the original house image*

edges = cv2.Canny(img, 100, 200)

edges\_blurred = cv2.Canny(blurred, 10, 80)

cv2.imshow('Original', img)

cv2.imshow('Blurred', blurred)

cv2.imshow('Edges', edges)

cv2.imshow('Edges blurred', edges\_blurred)

cv2.imshow

cv2.waitKey(0)

cv2.destroyAllWindows()

### TASK 1 CODE ENDS HERE ###

### BLUR WINDOW SCREENSHOT STARTS HERE ###



### BLUR WINDOW SCREENSHOT ENDS HERE ###

### CANNY 1 SCREENSHOT STARTS HERE ###



### CANNY 1 SCREENSHOT ENDS HERE ###

### CANNY 2 SCREENSHOT STARTS HERE ###



### CANNY 2 SCREENSHOT ENDS HERE ###

## Task 2 – Bitwise Operations

Load the square1 and square2 files for this task. Use the bitwise operations on these images to make an octagon as well as an 8-pointed star. (Think of the black pixels as 0 and the white pixels as 1 in the binary operations)

### TASK 2 CODE STARTS HERE ###

square1 = cv2.imread("lab4\_square1.jpg")

square1\_not = cv2.bitwise\_not(square1)

square2 = cv2.imread("lab4\_square2.jpg")

*# Make an 8-pointed star*

star = cv2.bitwise\_or(square1\_not, square2)

*# Make an octagon*

octagon = cv2.bitwise\_and(square1\_not, square2)

cv2.imshow('Octagon', octagon)

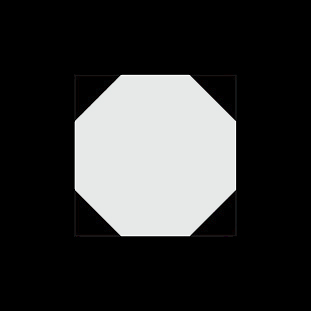
cv2.imshow('Star', star)

cv2.waitKey(0)

cv2.destroyAllWindows()

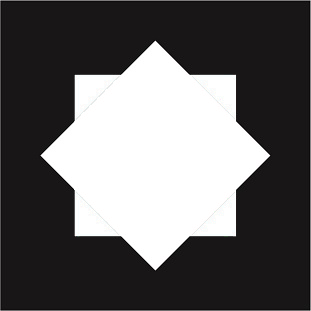
### TASK 2 CODE ENDS HERE ###

### OCTAGON SCREENSHOT STARTS HERE ###



### OCTAGON SCREENSHOT ENDS HERE ###

### STAR SCREENSHOT STARTS HERE ###



### STAR SCREENSHOT ENDS HERE ###

## Task 3 – HSV Color Space and InRange Function

Load the shapes.bmp image and change its color space to HSV (Hue, Saturation, Value). Then, use the inRange function to isolate each shape separately:

cv2.inRange(hsv,np.array([hmin,smin,vmin]),np.array([hmax,smax,vmax]))

You must manually determine the values for the above 6 parameters. H values go from 0 to 179. S and V values go from 0 to 255. Display all four results (black and white images) in four windows. Then, take a screenshot

### TASK 3 CODE STARTS HERE ###

img = cv2.imread("lab4\_shapes.bmp")

img\_hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

*# Isolate each shape separately*

blue = cv2.inRange(img\_hsv, np.array([100, 100, 100]), np.array([120, 255, 255]))

green = cv2.inRange(img\_hsv, np.array([40, 100, 100]), np.array([80, 255, 255]))

red = cv2.inRange(img\_hsv, np.array([130, 100, 100]), np.array([179, 255, 255]))

cyan = cv2.inRange(img\_hsv, np.array([80, 100, 100]), np.array([100, 255, 255]))

cv2.imshow("Blue", blue)

cv2.imshow("Green", green)

cv2.imshow("Red", red)

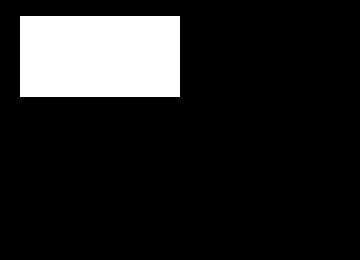
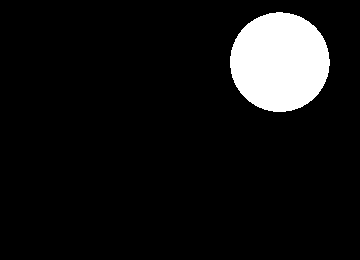
cv2.imshow("Cyan", cyan)

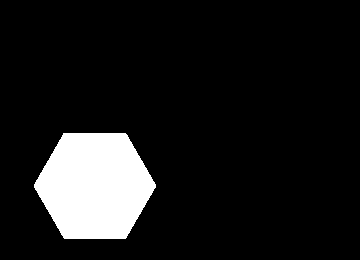
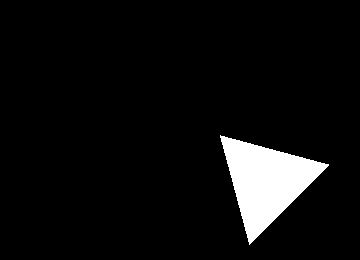
cv2.waitKey(0)

cv2.destroyAllWindows()

### TASK 3 CODE ENDS HERE ###

### TASK 3 SCREENSHOT STARTS HERE ###

### TASK 3 SCREENSHOT ENDS HERE ###

## Task 4 – Centroids

In the previous task, you acquired binary "masks" for each shape. In this task, you will determine the center of each shape. Using the code from the previous task, place a yellow circle at the center point of all shapes in the shapes.jpg image. Your screenshot must show a single image (shapes.jpg) with the center points marked.

To find the center point (cx, cy) in an image (img), use the moments function as shown in the code example:

M = cv2.moments (img)

if M['m00'] > 0:

cx = int(M['m10']/M['m00'])

cy = int(M['m01']/M['m00'])

### TASK 4 CODE STARTS HERE ###

M\_blue = cv2.moments(blue)

M\_green = cv2.moments(green)

M\_red = cv2.moments(red)

M\_cyan = cv2.moments(cyan)

if M\_blue['m00'] > 0:

    cx\_blue = *int*(M\_blue['m10']/M\_blue['m00'])

    cy\_blue = *int*(M\_blue['m01']/M\_blue['m00'])

if M\_green['m00'] > 0:

    cx\_green = *int*(M\_green['m10']/M\_green['m00'])

    cy\_green = *int*(M\_green['m01']/M\_green['m00'])

if M\_red['m00'] > 0:

    cx\_red = *int*(M\_red['m10']/M\_red['m00'])

    cy\_red = *int*(M\_red['m01']/M\_red['m00'])

if M\_cyan['m00'] > 0:

    cx\_cyan = *int*(M\_cyan['m10']/M\_cyan['m00'])

    cy\_cyan = *int*(M\_cyan['m01']/M\_cyan['m00'])

cv2.circle(img, (cx\_blue, cy\_blue), 5, (0, 255, 255), -1)

cv2.circle(img, (cx\_green, cy\_green), 5, (0, 255, 255), -1)

cv2.circle(img, (cx\_red, cy\_red), 5, (0, 255, 255), -1)

cv2.circle(img, (cx\_cyan, cy\_cyan), 5, (0, 255, 255), -1)

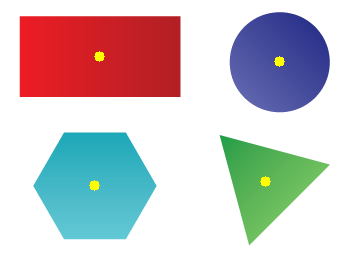
cv2.imshow("Shapes", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

### TASK 4 CODE ENDS HERE ###

### TASK 4 SCREENSHOT STARTS HERE ###



### TASK 4 SCREENSHOT ENDS HERE ###

## Task 5 – Perspective Transformation

Load the persp.jpg file for this task. Apply perspective transformation by using the four corners in the quadrilateral and map them to the four outer corners of the image file. The final result should be a flat, rectangular image (the width/height ratio in the final image is 3/4)

### TASK 5 CODE STARTS HERE ###

img = cv2.imread("lab4\_persp.jpg")

rows, cols, ch = img.shape

pts1 = np.float32([[34, 18], [282, 61], [217, 548], [397, 393]])

pts2 = np.float32([[0, 0], [420, 0], [0, 560], [420, 560]])

M = cv2.getPerspectiveTransform(pts1, pts2)

dst = cv2.warpPerspective(img, M, (cols, rows))

cv2.imshow("Perspective", dst)

cv2.waitKey(0)

cv2.destroyAllWindows()

### TASK 5 CODE ENDS HERE ###

### TASK 5 SCREENSHOT STARTS HERE ###



### TASK 5 SCREENSHOT ENDS HERE ###

## Task 6 – Pixel Replacement

Load the robot\_green\_bg file for this task. Using your knowledge from task 3, replace the green color of the background with red by scanning through each pixel. You can use RGB or HSV color space for this task. The following syntax examples will help:

* To check a pixel color in blue channel:

if img[i,j,0] < 100

* To set a pixel color as black:

img[i,j,:] = (0,0,0)

Take a screenshot of the picture with the modified background. Next, load the road picture. Modify the code so that the robot is placed on the road picture. You must go through each pixel in the image. You can place the robot anywhere, but the entire robot must be visible. Take the screenshot.

### TASK 6 CODE STARTS HERE ###

*# Replace the green color of the background with red*

img = cv2.imread("lab4\_robot\_green\_bg.bmp")

img\_hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

rows, cols, ch = img.shape

*# Through HSV color space*

green = cv2.inRange(img\_hsv, np.array([40, 100, 100]), np.array([80, 255, 255]))

green[green > 0] = 255

img[green == 255] = (0, 0, 255)

cv2.imshow("Robot\_HSV", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

*# Place the robot on the road picture*

img = cv2.imread("lab4\_robot\_green\_bg.bmp")

img\_hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

img\_road = cv2.imread("lab4\_road.jpg")

rows, cols, ch = img.shape

for i in range(rows):

    for j in range(cols):

        if img[i, j, 0] < 100 and img[i, j, 1] > 100 and img[i, j, 2] < 100:

            pass

        else:

            img\_road[i, j, :] = img[i, j, :]

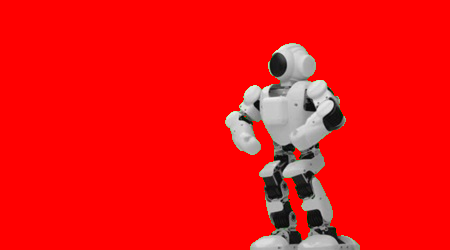
cv2.imshow("Robot on the road", img\_road)

cv2.waitKey(0)

cv2.destroyAllWindows()

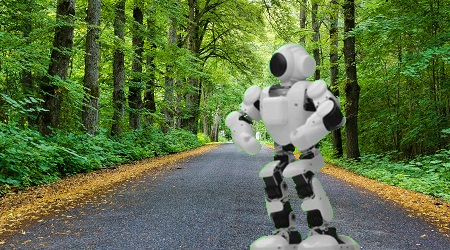
### TASK 6 CODE ENDS HERE ###

### RED BACKGROUND SCREENSHOT STARTS HERE ###



### RED BACKGROUND SCREENSHOT ENDS HERE ###

### ROAD ROBOT SCREENSHOT STARTS HERE ###



### ROAD ROBOT SCREENSHOT ENDS HERE ###

# Conclusion

In this laboratory exercise, we explored additional concepts pertaining to OpenCV, a popular and widely used library for image processing and computer vision applications. We implemented various OpenCV functions to perform basic preprocessing tasks such as blurring, edge-detection, and thresholding, as well as computer vision tasks such as image stitching and template matching.

Through this exercise, we gained a deeper understanding of the capabilities of OpenCV and how it can be used to develop a variety of image processing and computer vision applications. We also learned how to use OpenCV functions to perform common tasks such as image filtering, feature detection, and image matching.