**Department of Electrical Engineering**

**Faculty Member: Munadi Sial Date:** 15 – 10 - 23

**Semester: 7**  **Group: 3**

# CS477 Computer Vision

**Lab 4: Basic Image Processing Functions in OpenCV**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **PLO4 - CLO4** | **PLO5 -CLO5** | **PLO8 -CLO6** | **PLO9 -CLO7** |
| **Name** | **Reg. No** | **Investigation** | **Modern Tool Usage** | **Ethics** | **Individual and Team Work** |
|  |  | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** |
| **Ali Khalid** | 336676 |  |  |  |  |
| **Ayesha Binte Safi Ullah** | 343856 |  |  |  |  |
| **Maryam Shafiq** | 352203 |  |  |  |  |
| **Sehrish Kazmi** | 334099 |  |  |  |  |

## 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 ranges 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

* 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 particular colors in an image
* Use perspective transformation between corresponding points

## Lab Conduct

* Respect faculty and peers through speech and actions
* The lab faculty will be available to assist the students. In case some aspect of the lab experiment is not understood, the students are advised to seek help from the faculty.
* In the tasks, there are commented lines such as #YOUR CODE STARTS HERE# where you have to provide the code. You must put the code between the #START and #END parts of these commented lines. Do NOT remove the commented lines.
* Use the tab key to provide the indentation in python.

**Theory**

OpenCV is a library that focuses on image processing and computer vision. An image is an array of colored square 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 blurring, edge detection, bitwise operations, centroid determination, color space changing, color range selection and perspective transformation.

A brief summary of the relevant keywords and functions in python is provided below. (For more details, check the slides for this lab)

**print()** output text on console

**input()** get input from user on console

**range()**  create a sequence of numbers

**len()** gives the number of characters in a string

**if** contains code that executes depending on a logical condition

**else** connects with **if** and **elif**, executes when conditions are not met

**elif** equivalent to **else if**

**while** loops code as long as a condition is met

**for** loops code through a sequence of items in an iterable object

**break** exit loop immediately

**continue** jump to the next iteration of the loop

**def** used to define a function

In the lab, you will be provided with a number of image files which you must use in the tasks.

**Lab Task 1 – Gaussian Blur and Canny Edges [1]**

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 second on a slightly blurred picture. Use same threshold values on both edge detections.

***### TASK 1 CODE STARTS HERE ###***

|  |
| --- |
| import cv2  import numpy    # read image  src = cv2.imread('lab4\_house.jpg', cv2.IMREAD\_UNCHANGED)    # apply guassian blur on src image  dst = cv2.GaussianBlur(src,(15,15),cv2.BORDER\_DEFAULT)    # display input and output image  src\_edge = cv2.Canny(src,100,200)  dst\_edge = cv2.Canny(dst,100,200)  non\_edge\_image = numpy.hstack((src,dst))  edge\_image = numpy.hstack((src\_edge,dst\_edge))  cv2.imshow("Gaussian Smoothing",non\_edge\_image)  cv2.imshow("Edges",edge\_image)  cv2.waitKey(0) # waits until a key is pressed  cv2.destroyAllWindows() # destroys the window showing image  #applying edge detection |

*### TASK 1 CODE ENDS HERE ###*

***### BLUR WINDOW SCREENSHOT STARTS HERE ###***

******

*### BLUR WINDOW SCREENSHOT ENDS HERE ###*

***### CANNY 1 SCREENSHOT STARTS HERE ###***

A house with a black background

Description automatically generated

*### CANNY 1 SCREENSHOT ENDS HERE ###*

***### CANNY 2 SCREENSHOT STARTS HERE ###***

A black background with white lines

Description automatically generated

*### CANNY 2 SCREENSHOT ENDS HERE ###*

**Lab Task 2 – Bitwise Operations [1]**

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 ###***

|  |
| --- |
| import cv2  from google.colab.patches import cv2\_imshow  import numpy as np  square1 = cv2.imread("/content/drive/MyDrive/Colab Notebooks/Computer Vision/Labs/CV-Lab 04/lab4\_square1.jpg")  square2 = cv2.imread("/content/drive/MyDrive/Colab Notebooks/Computer Vision/Labs/CV-Lab 04/lab4\_square2.jpg")  # Octagon  oct1 = cv2.bitwise\_or(square1, square2)  oct2 = cv2.bitwise\_xor(square1,oct1)  # oct2 = cv2.bitwise\_and(oc1)  cv2\_imshow(oct2)  # 8 point star  star1 = cv2.bitwise\_xor(square1,square2)  star2 = cv2.bitwise\_and(square1,star1)  cv2\_imshow(star2) |

*### TASK 2 CODE ENDS HERE ###*

***### OCTAGON SCREENSHOT STARTS HERE ###***

A white octagon on a black background

Description automatically generated

*### OCTAGON SCREENSHOT ENDS HERE ###*

***### STAR SCREENSHOT STARTS HERE ###***

A black and white logo

Description automatically generated

*### STAR SCREENSHOT ENDS HERE ###*

**Lab Task 3 – HSV Color Space and InRange Function [2]**

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 v alues 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

Hint: Experiment with V values first to remove the background white color.

***### TASK 3 CODE STARTS HERE ###***

|  |
| --- |
| import cv2  from google.colab.patches import cv2\_imshow  import numpy as np  shapes = cv2.imread("/content/drive/MyDrive/Colab Notebooks/Computer Vision/Labs/CV-Lab 04/lab4\_shapes.bmp")  # cv2\_imshow(shapes)  shapes\_hsv = cv2.cvtColor(shapes,cv2.COLOR\_BGR2HSV)  # cv2\_imshow(shapes\_hsv)  rectangle = cv2.inRange(shapes\_hsv, np.array([150,50,50]), np.array([179,255,255]))  circle = cv2.inRange(shapes\_hsv, np.array([100,50,50]),np.array([120,255,255]))  octagon = cv2.inRange(shapes\_hsv, np.array([90,50,50]), np.array([100,255,255]))  triangle = cv2.inRange(shapes\_hsv, np.array([30,50,50]), np.array([70,255,255]))  cv2\_imshow(rectangle)  cv2\_imshow(circle)  cv2\_imshow(octagon)  cv2\_imshow(triangle)  all\_shapes = rectangle + circle + octagon + triangle  cv2\_imshow(all\_shapes) |

*### TASK 3 CODE ENDS HERE ###*

***### TASK 3 SCREENSHOT STARTS HERE ###***

A white circle in black background

Description automatically generated

A black and white flag

Description automatically generated

A white hexagon on a black background

Description automatically generated

A white triangle on a black background

Description automatically generated

*### TASK 3 SCREENSHOT ENDS HERE ###*

**Lab Task 4 – Centroids [1]**

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'])

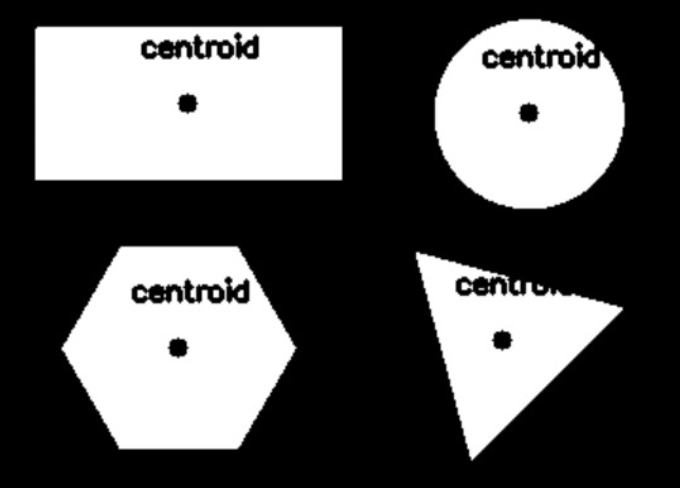
cy = int(M['m01']/M['m00'])

***### TASK 4 CODE STARTS HERE ###***

|  |
| --- |
| # lab4 task4  # rectangle  M = cv2.moments (rectangle)  if M['m00'] > 0:  cx = int(M['m10']/M['m00'])  cy = int(M['m01']/M['m00'])  cv2.circle(rectangle, (cx, cy), 5, (0, 255, 255), -1)  cv2.putText(rectangle, "centroid", (cx - 25, cy - 25),cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 255), 2)  cv2\_imshow(rectangle)  # circle  M = cv2.moments (circle)  if M['m00'] > 0:  cx = int(M['m10']/M['m00'])  cy = int(M['m01']/M['m00'])  cv2.circle(circle, (cx, cy), 5, (0, 255, 255), -1)  cv2.putText(circle, "centroid", (cx - 25, cy - 25),cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 255), 2)  cv2\_imshow(circle)  # octagon  M = cv2.moments (octagon)  if M['m00'] > 0:  cx = int(M['m10']/M['m00'])  cy = int(M['m01']/M['m00'])  cv2.circle(octagon, (cx, cy), 5, (0, 255, 255), -1)  cv2.putText(octagon, "centroid", (cx - 25, cy - 25),cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 255), 2)  cv2\_imshow(octagon)  # triangle  M = cv2.moments (triangle)  if M['m00'] > 0:  cx = int(M['m10']/M['m00'])  cy = int(M['m01']/M['m00'])  cv2.circle(triangle, (cx, cy), 5, (0, 255, 255), -1)  cv2.putText(triangle, "centroid", (cx - 25, cy - 25),cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 255), 2)  cv2\_imshow(triangle) |

*### TASK 4 CODE ENDS HERE ###*

***### TASK 4 SCREENSHOT STARTS HERE ###***



*### TASK 4 SCREENSHOT ENDS HERE ###*

**Lab Task 5 – Perspective Transformation [2]**

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 ###***

|  |
| --- |
| import cv2  import numpy as np  img = cv2.imread("lab4\_persp.jpg")  height, width = img.shape[:2]  original\_image\_coords = np.float32([[35,19],[217,547],[281,61],[400,280]])  transform\_image\_coords = np.float32([[0,0],[0,560],[420,0],[560,420]])  # Calculate the perspective transformation matrix  M = cv2.getPerspectiveTransform(original\_image\_coords, transform\_image\_coords)  # Apply the perspective transformation to the image  warped\_image = cv2.warpPerspective(img, M, (width, height))  print(height,width)  cv2.imshow("image",warped\_image)  cv2.imwrite("saved.jpg",warped\_image)  cv2.waitKey(0) # waits until a key is pressed  cv2.destroyAllWindows() # destroys the window showing image |

*### TASK 5 CODE ENDS HERE ###*

***### TASK 5 SCREENSHOT STARTS HERE ###***



*### TASK 5 SCREENSHOT ENDS HERE ###*

**Lab Task 6 – Pixel Replacement [1]**

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)

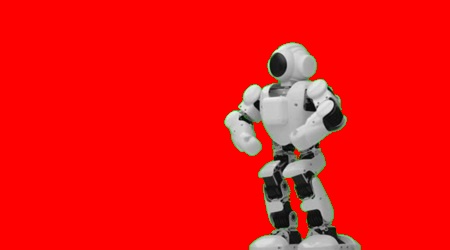
Take a screenshot of the picture with the modified background. Next, load the road picture. Modify the code so that the robot is placed on the road picture. You must go through each pixel in the image. You can place the robot anywhere but the entire robot must be visible. Take the screenshot.

***### TASK 6 CODE STARTS HERE ###***

|  |
| --- |
| import cv2  import numpy as np  robot\_image = cv2.imread("lab3\_robot\_green\_bg.bmp")  height, width = robot\_image.shape[:2]  *# Define the lower and upper green thresholds*  lower\_green = np.array([0, 100, 0], dtype=np.uint8)  *# Lower limit for green*  upper\_green = np.array([50, 255, 50], dtype=np.uint8)  *# Upper limit for green*  green\_mask = cv2.inRange(robot\_image, lower\_green, upper\_green)    list1 = list()  for y in range(height):      for x in range(width):  *# Get the pixel value at (x, y)*    *#check the range of green pixels*          if tuple(robot\_image[y, x, :]) not in list1:              list1.append(tuple(robot\_image[y,x,:]))            if green\_mask[y, x] == 255:              robot\_image[y, x, :] = (0,0,255)    cv2.imshow("image", robot\_image)  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 ###***

A white robot walking on a road with trees

Description automatically generated

*### ROAD ROBOT SCREENSHOT ENDS HERE ###*