**Department of Electrical Engineering and   
Computer Science**

**Faculty Member:** Dr. Hafsa Iqbal **Date:** 26/04/2024

**Semester:** 8th **Group:** GP-1

**Lab Engineer:** Munadi Ahmad Sial

**EE-381 Robotics**

Lab 10: OpenCV Functions and CVBridge

**Group Members**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **PLO5 – CLO4** | | **PLO9 –**  **CLO4** |  |
| **Name** | **Reg. No** | **Analysis of Data in Lab Report** | **Modern Tool Usage** | **Individual and Team Work** | **Total Marks** |
|  |  | **10 Marks** | **5 Marks** | **5 Marks** | **20 Marks** |
| Muhammad Abdullah Sohail | 343642 |  |  |  |  |
| Muhammad Ahmed Mohsin | 333060 |  |  |  |  |
| Muhammad Umer | 345834 |  |  |  |  |
| Hassan Rizwan | 335753 |  |  |  |  |
| Muhammad Saad | 333414 |  |  |  |  |

**Table of Contents**

[2 OpenCV Functions and CVBridge 3](#_Toc165052878)

[2.1 Introduction 3](#_Toc165052879)

[2.2 Objectives 3](#_Toc165052880)

[2.3 Theory 3](#_Toc165052881)

[3 Lab Tasks 5](#_Toc165052882)

[3.1 Lab Task 1 – Gaussian Blur and Canny Edges 5](#_Toc165052883)

[3.2 Lab Task 2 – Bitwise Operations 7](#_Toc165052884)

[3.3 Lab Task 3 – HSV Color Space and InRange Function 8](#_Toc165052885)

[3.4 Lab Task 4 – Centroids 9](#_Toc165052886)

[3.5 Lab Task 5 – Perspective Transformation 11](#_Toc165052887)

[3.6 Lab Task 6 – Pixel Replacement 12](#_Toc165052888)

[3.7 Lab Task 7 – CV Bridge 13](#_Toc165052889)

# OpenCV Functions and CVBridge

## Introduction

This laboratory exercise will focus on additional concepts pertaining to OpenCV which was introduced in the previous lab. OpenCV is a widely used library for image processing and computer vision applications. The field of computer vision is very important for robotics as it can be used to extract important information of a robot’s environment. OpenCV contains a wide selection of functions for vision-based algorithms. These functions range from basic preprocessing to computer vision implementations including blurring, edge-detection, threshold, 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 particular colors in an image
* Use perspective transformation between corresponding points
* Interface ROS with OpenCV using the CVBridge

## 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.

To use OpenCV in ROS nodes, the ROS CVBridge is an important module that is used to transform incoming camera data published to a topic. The result is a conversion to OpenCV compatible objects that can be used with OpenCV functions.

The terminal commands are given as:

**cd <directory>**  change directory

**cd..**  go back to previous directory

**pwd**  print the current directory

**ls** list the contents of the current directory

**python <script.py>** execute python script

A brief summary of the terminal commands needed for working in ROS are provided below:

**colcon build**

build the workspace whenever a node is created or modified

**. install/setup.bash**

makes the terminal “aware” of the workspace (notice the dot and the space)

**ros2 pkg create --build-type ament\_python <package>**

create a new package (must be done in src directory)

**ros2 run <package> <node>**

execute a node

In the lab, you will download a number of image files which you must use in the tasks. Ensure that the terminal shows the names of the group members in the screenshots.

# Lab Tasks

## Lab Task 1 – Gaussian Blur and Canny Edges

Load the images for this task. Use the Gaussian Blur function to blur a certain portion of the images. Specify the size of the kernel for the blur.

Additionally, apply Canny edge detection on the images. You must apply the edge detections one on the original picture and second on the blurred picture. Use same threshold values on all edge detections.

### TASK 1 CODE STARTS HERE ###

import cv2

import numpy as np

img1 = cv2.imread("robot\_green\_bg.bmp", cv2.COLOR\_BGR2RGB)

img2 = cv2.imread("road.jpg", cv2.COLOR\_BGR2RGB)

*# gaussian blur*

img1\_blur = cv2.GaussianBlur(img1, (7, 7), 0)

img2\_blur = cv2.GaussianBlur(img2, (7, 7), 0)

*# edge detection*

img1\_edges = cv2.Canny(img1, 50, 200)

img1\_blur\_edges = cv2.Canny(img1\_blur, 50, 200)

img2\_edges = cv2.Canny(img2, 50, 200)

img2\_blur\_edges = cv2.Canny(img2\_blur, 50, 200)

*# display*

cv2.imshow("Robot", img1)

cv2.imshow("Robot Blurred", img1\_blur)

cv2.imshow("Robot Edges", img1\_edges)

cv2.imshow("Robot Blurred Edges", img1\_blur\_edges)

cv2.imshow("Road", img2)

cv2.imshow("Road Blurred", img2\_blur)

cv2.imshow("Road Edges", img2\_edges)

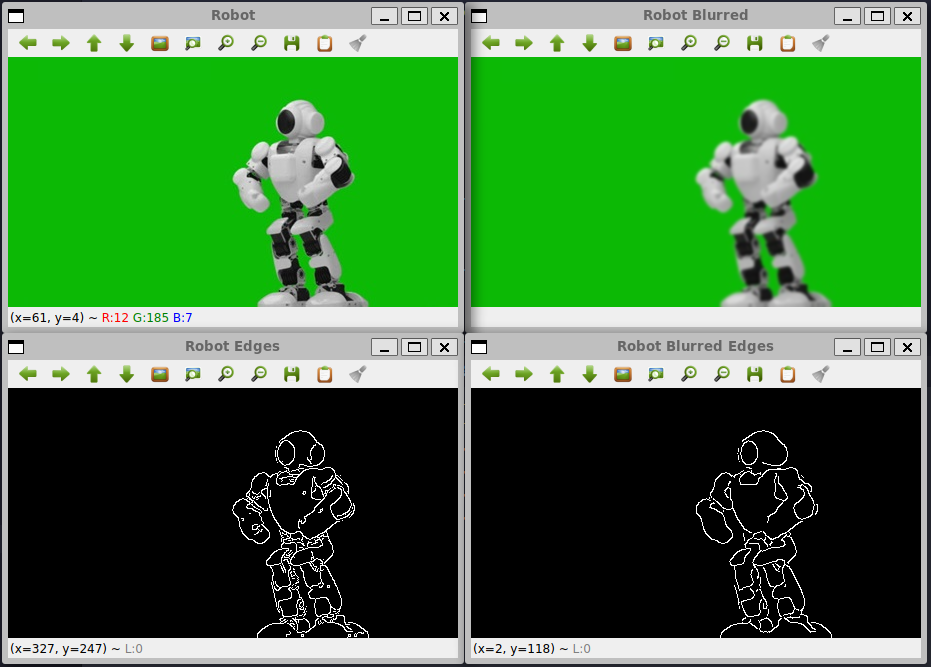
cv2.imshow("Road Blurred Edges", img2\_blur\_edges)

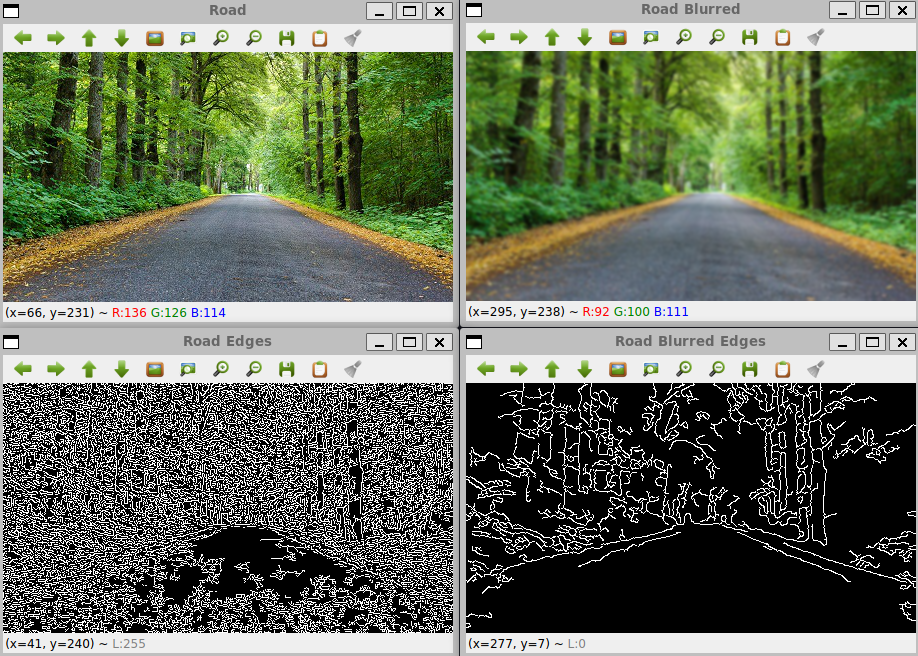
cv2.waitKey(0)

cv2.destroyAllWindows()

### TASK 1 CODE ENDS HERE ###

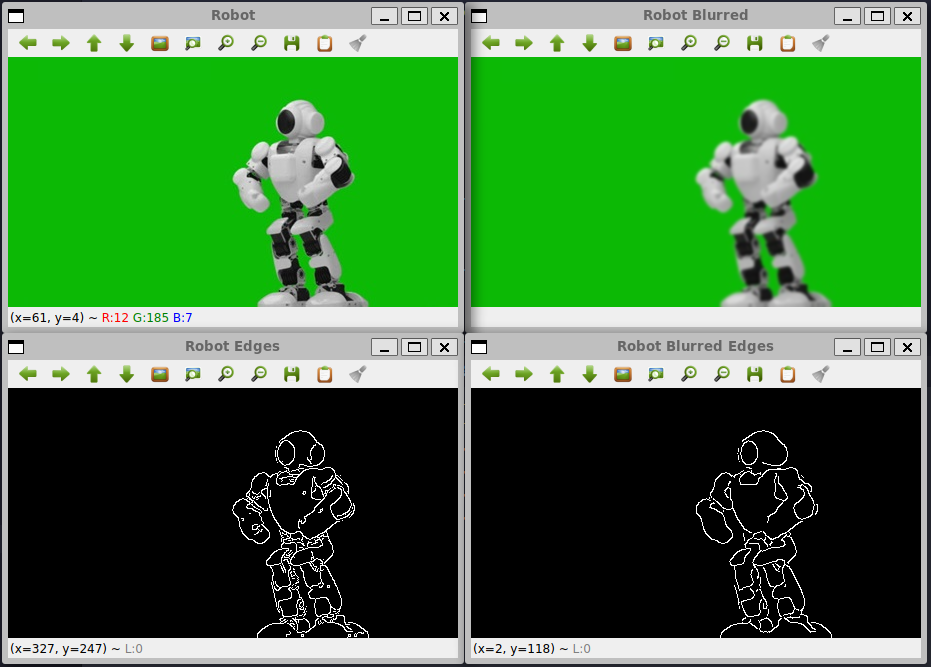
### GAUSSIAN BLUR SCREENSHOTS START HERE ###

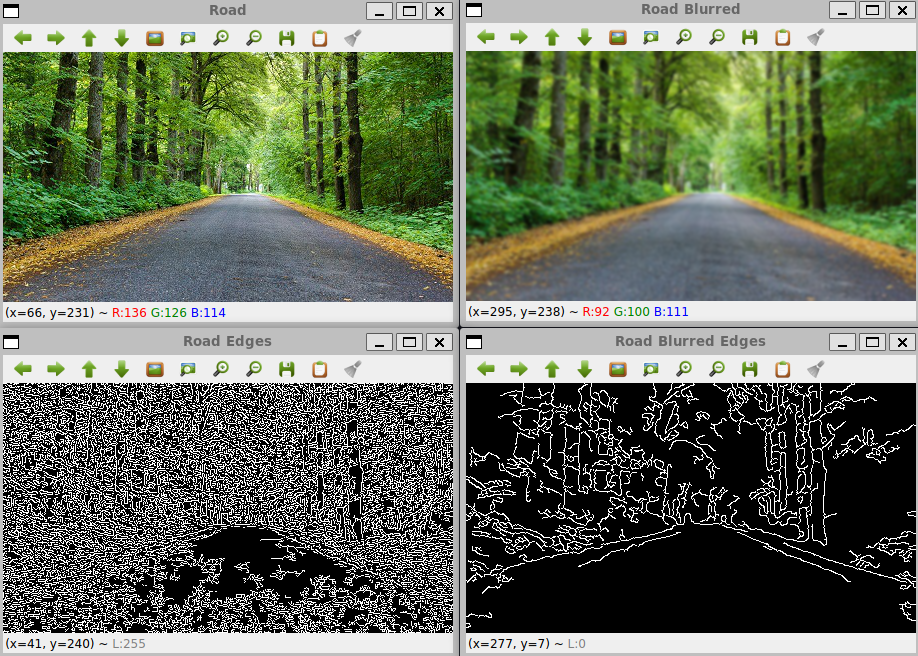




### GAUSSIAN BLUR SCREENSHOTS END HERE ###

### CANNY SCREENSHOTS START HERE ###





### CANNY SCREENSHOTS END HERE ###

## Lab 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 ###

import cv2

import numpy as np

img1 = cv2.imread("square1.jpg", 0)

img1\_not = cv2.bitwise\_not(img1)

img2 = cv2.imread("square2.jpg", 0)

*# 8-pointed star*

star = cv2.bitwise\_or(img1\_not, img2)

*# octagon*

octagon = cv2.bitwise\_and(img1\_not, img2)

*# display*

cv2.imshow("Star", star)

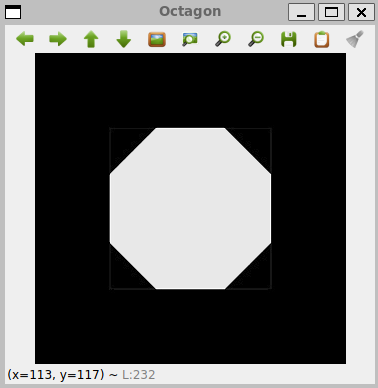
cv2.imshow("Octagon", octagon)

cv2.waitKey(0)

cv2.destroyAllWindows()

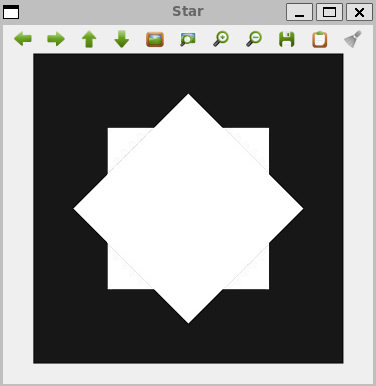
### TASK 2 CODE ENDS HERE ###

### OCTAGON SCREENSHOT STARTS HERE ###



### OCTAGON SCREENSHOT ENDS HERE ###

### STAR SCREENSHOT STARTS HERE ###



### STAR SCREENSHOT ENDS HERE ###

## Lab 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 while S and V values go from 0 to 255. Display all four results (black and white images) in four windows. Then, take a screenshot. Experiment with V values first to remove the background white color.

### TASK 3 CODE STARTS HERE ###

import cv2

import numpy as np

img = cv2.imread("shapes.bmp")

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

*# isolates colors*

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)

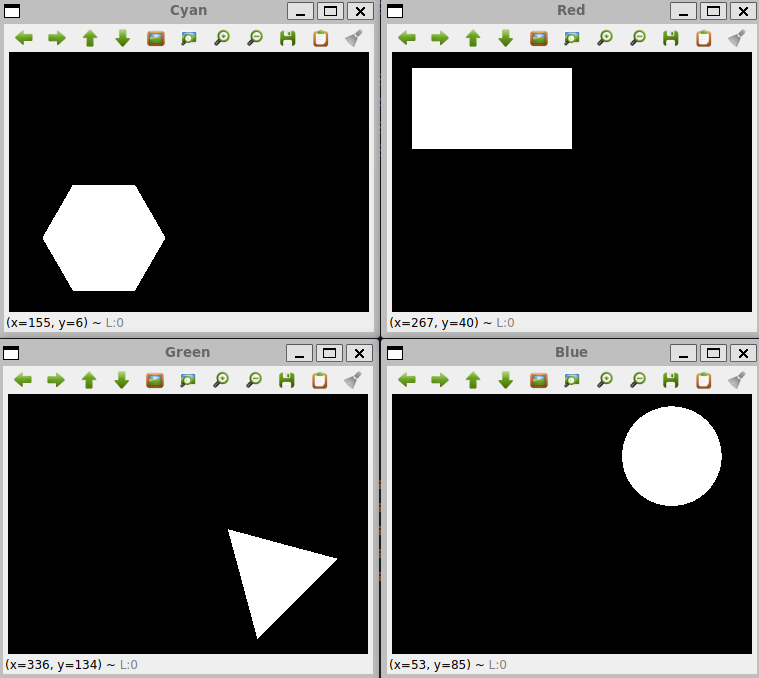
cv2.imshow("Cyan", cyan)

cv2.waitKey(0)

cv2.destroyAllWindows()

### TASK 3 CODE ENDS HERE ###

### TASK 3 SCREENSHOT STARTS HERE ###



### TASK 3 SCREENSHOT ENDS HERE ###

## Lab 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 colored 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 ###

import cv2

import numpy as np

img = cv2.imread("shapes.bmp")

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

*# isolates colors*

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]))

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)

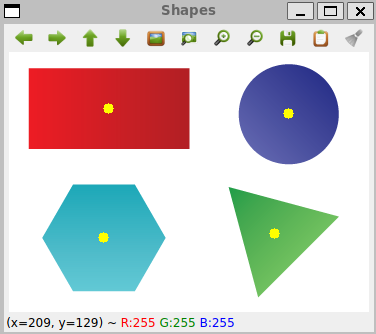
cv2.imshow("Shapes", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

### TASK 4 CODE ENDS HERE ###

### TASK 4 SCREENSHOT STARTS HERE ###



### TASK 4 SCREENSHOT ENDS HERE ###

## Lab 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). Also, apply perspective transformation to warp your downloaded images. Include all resulting transforms in the screenshot.

### TASK 5 CODE STARTS HERE ###

import cv2

import numpy as np

img = cv2.imread("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))

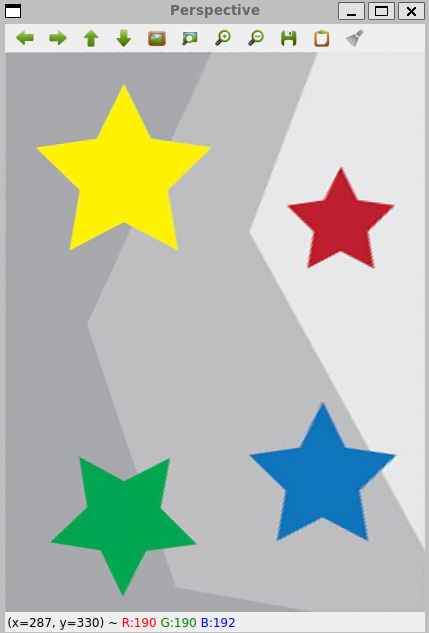
cv2.imshow("Perspective", dst)

cv2.waitKey(0)

cv2.destroyAllWindows()

### TASK 5 CODE ENDS HERE ###

### TASK 5 SCREENSHOT STARTS HERE ###



### TASK 5 SCREENSHOT ENDS HERE ###

## Lab 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 ###

import cv2

import numpy as np

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

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

rows, cols, ch = img.shape

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()

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

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

img\_road = cv2.imread("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, :]

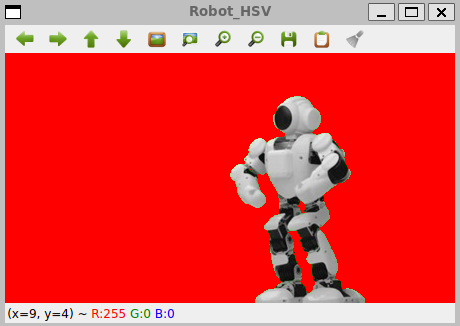
cv2.imshow("Robot on the road", img\_road)

cv2.waitKey(0)

cv2.destroyAllWindows()

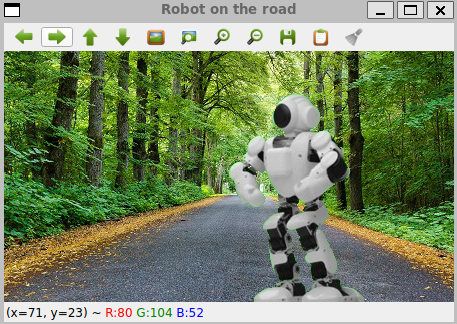
### TASK 6 CODE ENDS HERE ###

### RED BACKGROUND SCREENSHOT STARTS HERE ###



### RED BACKGROUND SCREENSHOT ENDS HERE ###

### ROAD ROBOT SCREENSHOT STARTS HERE ###



### ROAD ROBOT SCREENSHOT ENDS HERE ###

## Lab Task 7 – CV Bridge

In this task, you will use OpenCV for a robot simulation. Load the simulation of the robot containing a camera. In a newly created package, create a node (that can subscribe). Place the following imports in the node:

import rclpy

from rclpy.node import Node

from geometry\_msgs.msg import Twist

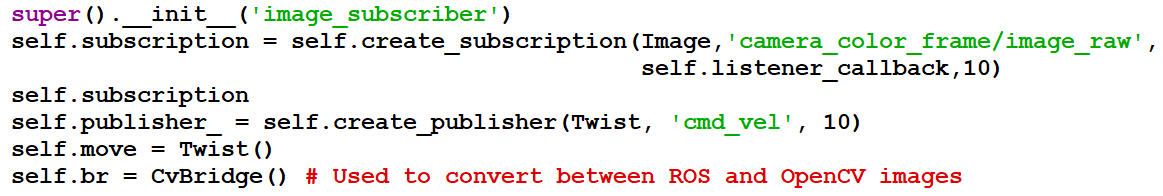
from sensor\_msgs.msg import Image

from cv\_bridge import CvBridge

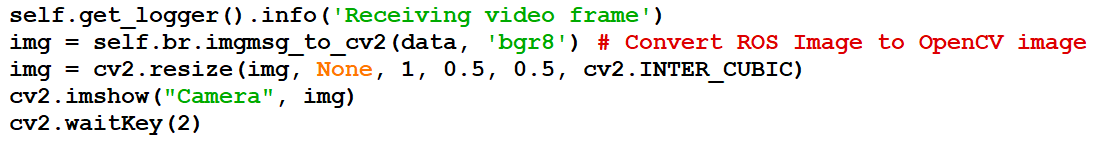
import cv2

import numpy as np

In the node constructor, place the following code:



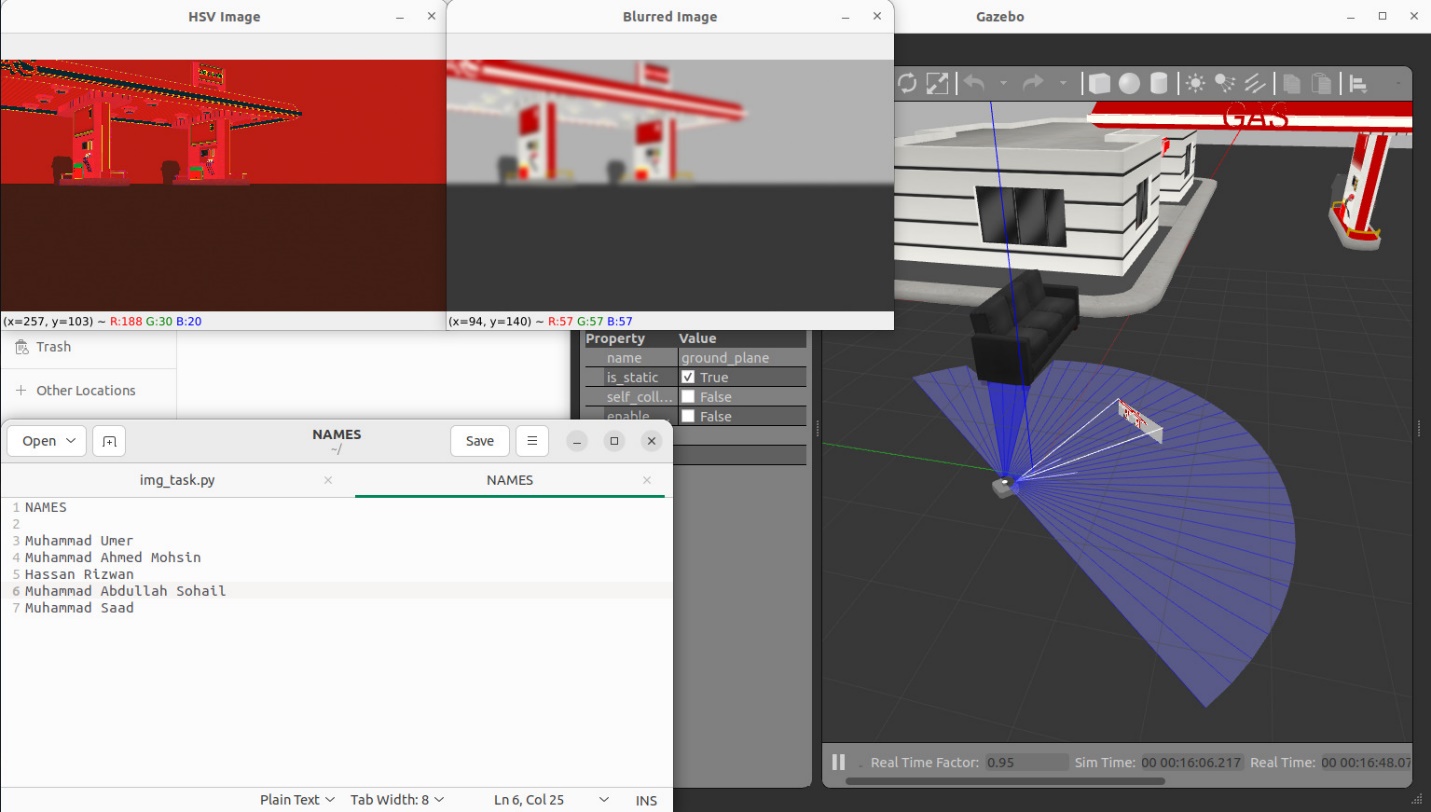
In the subscriber callback function, place the following code:



Build the package and execute the node. A window will appear showing the camera output.

For the task submission, edit the subscriber code so that the image frames are converted into HSV space (refer to task 3). Modify the HSV color values so that the camera output shows very different colors and display the output window. Additionally, display a second window showing a blurred view of the camera output (refer to task 1). Take a SINGLE screenshot showing the robot simulation and the resulting windows. Provide the screenshot for the task submission.

### TASK 7 HSV+BLUR SCREENSHOT STARTS HERE ###



### TASK 7 HSV+BLUR SCREENSHOT ENDS HERE ###