**Department of Electrical Engineering and   
Computer Science**

**Faculty Member:** Dr. Hafsa Iqbal **Date:** 30/04/2024

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

**Lab Engineer:** Munadi Ahmad Sial

**EE-381 Robotics**

Lab 11: 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 |  |  |  |  |

## Introduction

This laboratory exercise will focus on using the concepts of OpenCV learnt in the previous labs and applying them for a ROS simulation. The simulation will involve using the mounted camera to acquire image data and then programming the robot to perform line following.

## Objectives

* Use CV Bridge to convert image data to OpenCV object
* Acquire masks for line following
* Determine centroid from the region of interest
* Use the shifting centroid to influence robot’s motion

## 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. In ROS, image data from a camera is sent to a topic. Subscribing to this topic will obtain the image data. In order to use OpenCV, the image data must be converted to an OpenCV object. For this, the CV Bridge is used. After this conversion, we can use OpenCV in the camera images to allow computer vision in a robotics framework.

Common terminal commands for ROS 2 are provided below:

**colcon build --packages-select <package\_name>**

build a specific package whenever a node is created or modified

**. install/setup.bash**

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

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

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

**ros2 pkg create --build-type ament\_python <package\_name> --dependencies rclpy std\_msgs geometry\_msgs sensor\_msgs**

create a new package with dependencies

**ros2 run <package\_name> <node\_name>**

execute a node in the terminal

In the lab, you will be provided with a color tracker program and a “lines” model file. Perform the lab tasks sequentially to complete the line follower simulation. You will work on making a single node. Each successive task will involve adding more to the node.

**Lab Task 1 – CV Bridge \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ [1]**

Place the “lines” model in the Gazebo directory so that it appears in the simulation. Startup Gazebo, then place the NeuronBot (with camera) and “lines” model in the simulation. Use the teleoperation program to move around the robot.

In a package, create a node (that can subscribe and publish). Place the following imports in the node:

**import rclpy**

**from rclpy.node import Node**

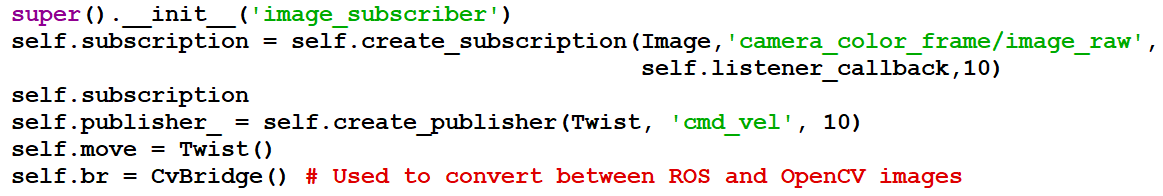
**from geometry\_msgs.msg import Twist**

**from sensor\_msgs.msg import Image**

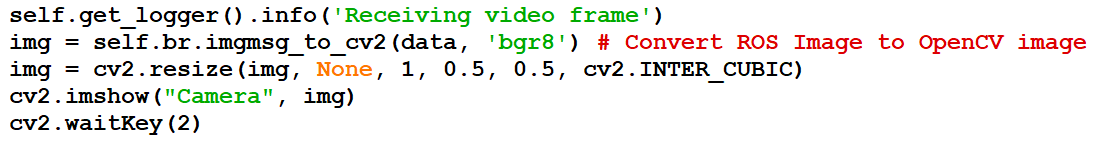
**from cv\_bridge import CvBridge**

**import cv2**

**import numpy**

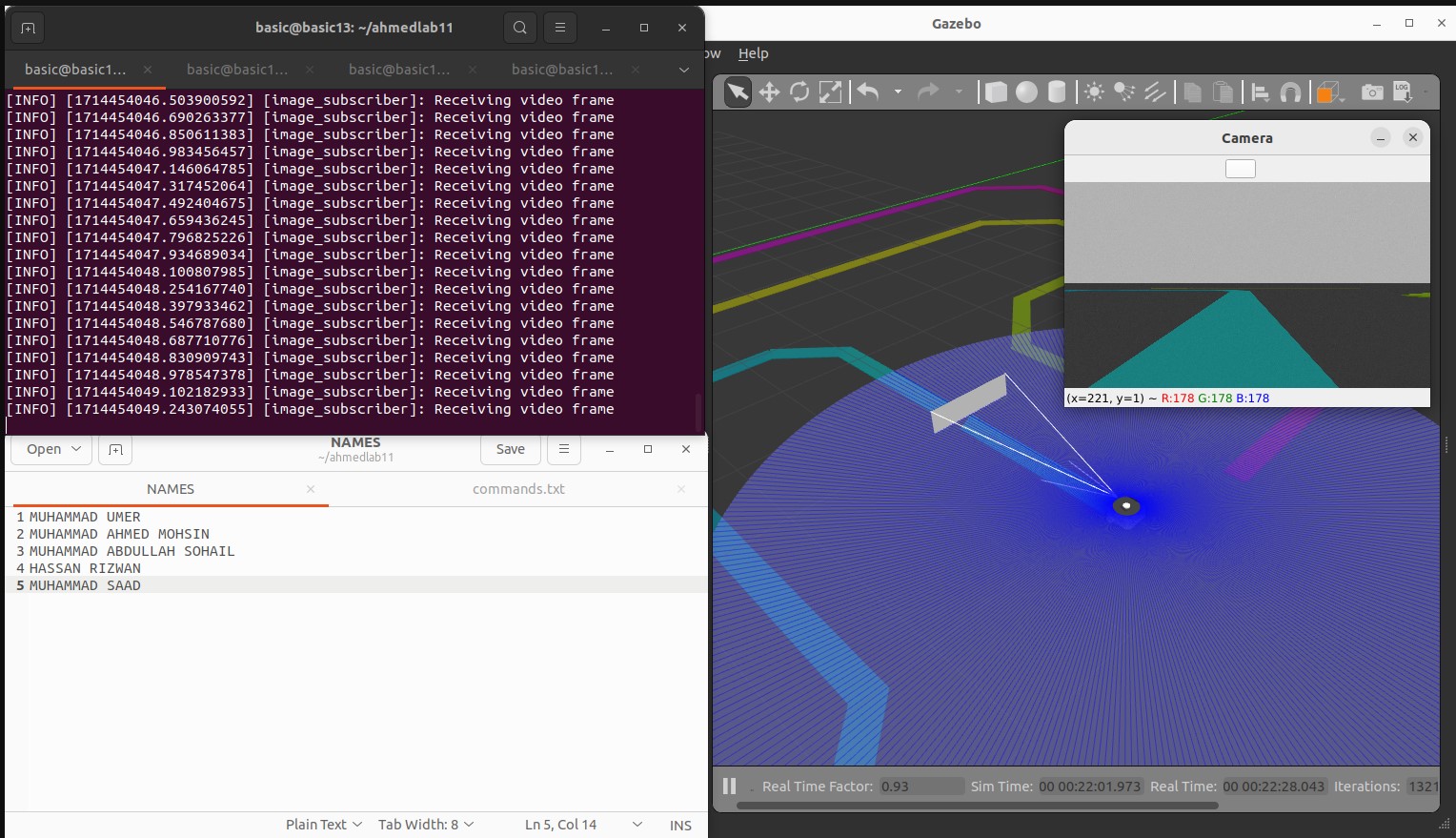
In the node constructor, place the following code:

In the subscriber callback function, place the following code:



Execute the node. A window will appear showing the camera output. Take a screenshot showing the simulation and the camera output. Provide the screenshot.

***### IMAGE DATA SCREENSHOT STARTS HERE ###***



*### IMAGE DATA SCREENSHOT ENDS HERE ###*

**Lab Task 2 – Line Thresholding \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ [2]**

Choose one of the three lines for the line following. You have been provided with the “color\_tracker.py” node. Place the node in the package and execute it. A few windows will appear namely the trackbar window, mask window and result window. Use the sliders to adjust the range of HSV values. Use this to get the required HSV values for the line threshold. Provide a screenshot showing the windows and the simulation.

***### LINE THRESHOLD SCREENSHOT STARTS HERE ###***

import cv2

import numpy as np

import rclpy

from cv\_bridge import CvBridge

from rclpy.node import Node

from sensor\_msgs.msg import Image

class ImageSubscriber(Node):

    def \_\_init\_\_(self):

        super().\_\_init\_\_("image\_subscriber")

        self.subscription = self.create\_subscription(

            Image, "camera/image\_raw", self.listener\_callback, 10

        )

        self.subscription

        self.br = CvBridge()  # Used to convert between ROS and OpenCV images

        def empty(a):

            pass

        cv2.namedWindow("TrackBars")

        cv2.resizeWindow("TrackBars", (640, 240))

        cv2.createTrackbar("Hue Min", "TrackBars", 0, 179, empty)

        cv2.createTrackbar("Hue Max", "TrackBars", 179, 179, empty)

        cv2.createTrackbar("Sat Min", "TrackBars", 0, 255, empty)

        cv2.createTrackbar("Sat Max", "TrackBars", 255, 255, empty)

        cv2.createTrackbar("Val Min", "TrackBars", 0, 255, empty)

        cv2.createTrackbar("Val Max", "TrackBars", 255, 255, empty)

    def listener\_callback(self, data):

        self.get\_logger().info("Receiving video frame")

        img = self.br.imgmsg\_to\_cv2(

            data, "bgr8"

        )  # Convert ROS Image message to OpenCV image

        img = cv2.resize(img, None, 1, 0.5, 0.5, cv2.INTER\_CUBIC)

        # while True:

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

        hmin = cv2.getTrackbarPos("Hue Min", "TrackBars")

        hmax = cv2.getTrackbarPos("Hue Max", "TrackBars")

        smin = cv2.getTrackbarPos("Sat Min", "TrackBars")

        smax = cv2.getTrackbarPos("Sat Max", "TrackBars")

        vmin = cv2.getTrackbarPos("Val Min", "TrackBars")

        vmax = cv2.getTrackbarPos("Val Max", "TrackBars")

        # making the mask

        low = np.array([hmin, smin, vmin])

        upp = np.array([hmax, smax, vmax])

        mask = cv2.inRange(imgHSV, low, upp)

        result = cv2.bitwise\_and(img, img, mask=mask)

        # cv2.imshow("HSV", imgHSV)

        cv2.imshow("Mask", mask)

        cv2.imshow("Result", result)

        # cv2.waitKey(1)

        # if cv2.waitKey(1) & 0xFF==ord('q'):

        # break

        print("========================================")

        print("hmin, smin, vmin :", [hmin, smin, vmin])

        print("hmax, smax, vmax :", [hmax, smax, vmax])

        print("========================================")

        cv2.waitKey(1)

def main(args=None):

    rclpy.init(args=args)

    image\_subscriber = ImageSubscriber()

    rclpy.spin(image\_subscriber)

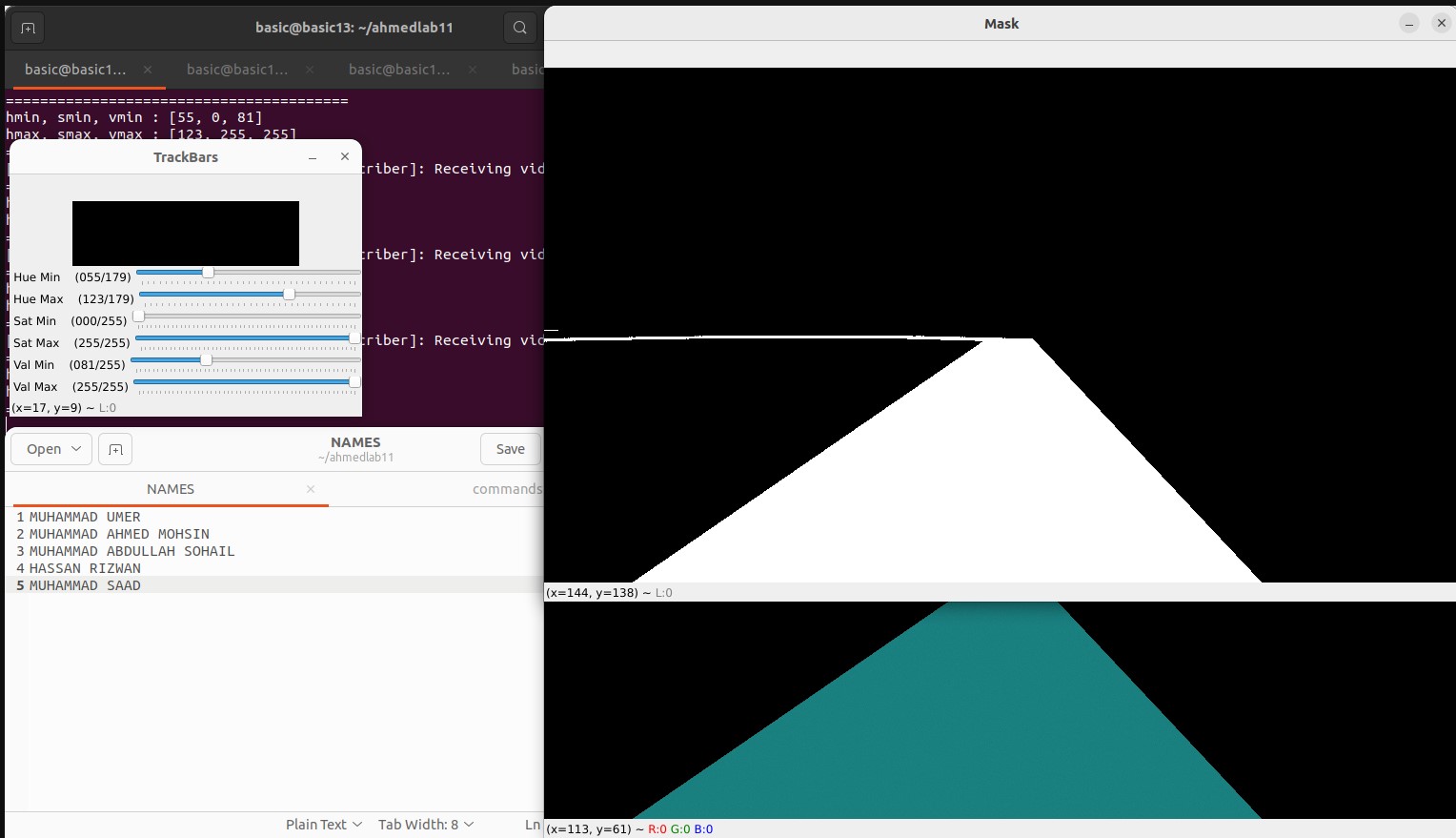
    image\_subscriber.destroy\_node()

    rclpy.shutdown()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Screenshot:**



*### LINE THRESHOLD SCREENSHOT ENDS HERE ###*

**Lab Task 3 – Region of Interest \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_[2]**

Using the HSV values you acquired in the previous task, modify the line following node so that the line threshold appears in a window. Convert the camera image to HSV for this purpose. Also, acquire the region of interest from the masked line (as mentioned in the slides). Show the line threshold and region of interest in separate windows. Execute the node and take a screenshot of the simulation showing the windows.

***### REGION OF INTEREST SCREENSHOT STARTS HERE ###***

from copy import deepcopy

import cv2

import numpy as np

import rclpy

from cv\_bridge import CvBridge

from geometry\_msgs.msg import Twist

from rclpy.node import Node

from sensor\_msgs.msg import Image

class InterestRegion(Node):

    def \_\_init\_\_(self):

        super().\_\_init\_\_("task\_3")

        self.subscription = self.create\_subscription(

            Image, "camera/image\_raw", self.callback, 10

        )

        self.subscription

        self.publisher\_ = self.create\_publisher(Twist, "cmd\_vel", 10)

        self.move = Twist()

        self.br = CvBridge()

    def callback(self, msg):

        self.get\_logger().info("Receiving video frame")

        img = self.br.imgmsg\_to\_cv2(msg, desired\_encoding="bgr8")

        img = cv2.resize(img, None, 1, 0.2, 0.2, cv2.INTER\_CUBIC)

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

        h\_min = 55

        h\_max = 179

        s\_min = 0

        s\_max = 255

        v\_min = 81

        v\_max = 255

        # making the mask

        low = np.array([h\_min, s\_min, v\_min])

        upp = np.array([h\_max, s\_max, v\_max])

        mask = cv2.inRange(img\_hsv, low, upp)

        result = cv2.bitwise\_and(img, img, mask=mask)

        # region of interest (keep the lower 1/5 of the image)

        height, width = mask.shape

        roi = deepcopy(mask)

        roi[: 4 \* height // 5, :] = 0

        # roi = mask[4 \* height // 5 :, :]

        cv2.imshow("ROI", roi)

        cv2.imshow("Mask", mask)

        cv2.imshow("Result", result)

        cv2.waitKey(2)

def main(args=None):

    rclpy.init(args=args)

    node = InterestRegion()

    rclpy.spin(node)

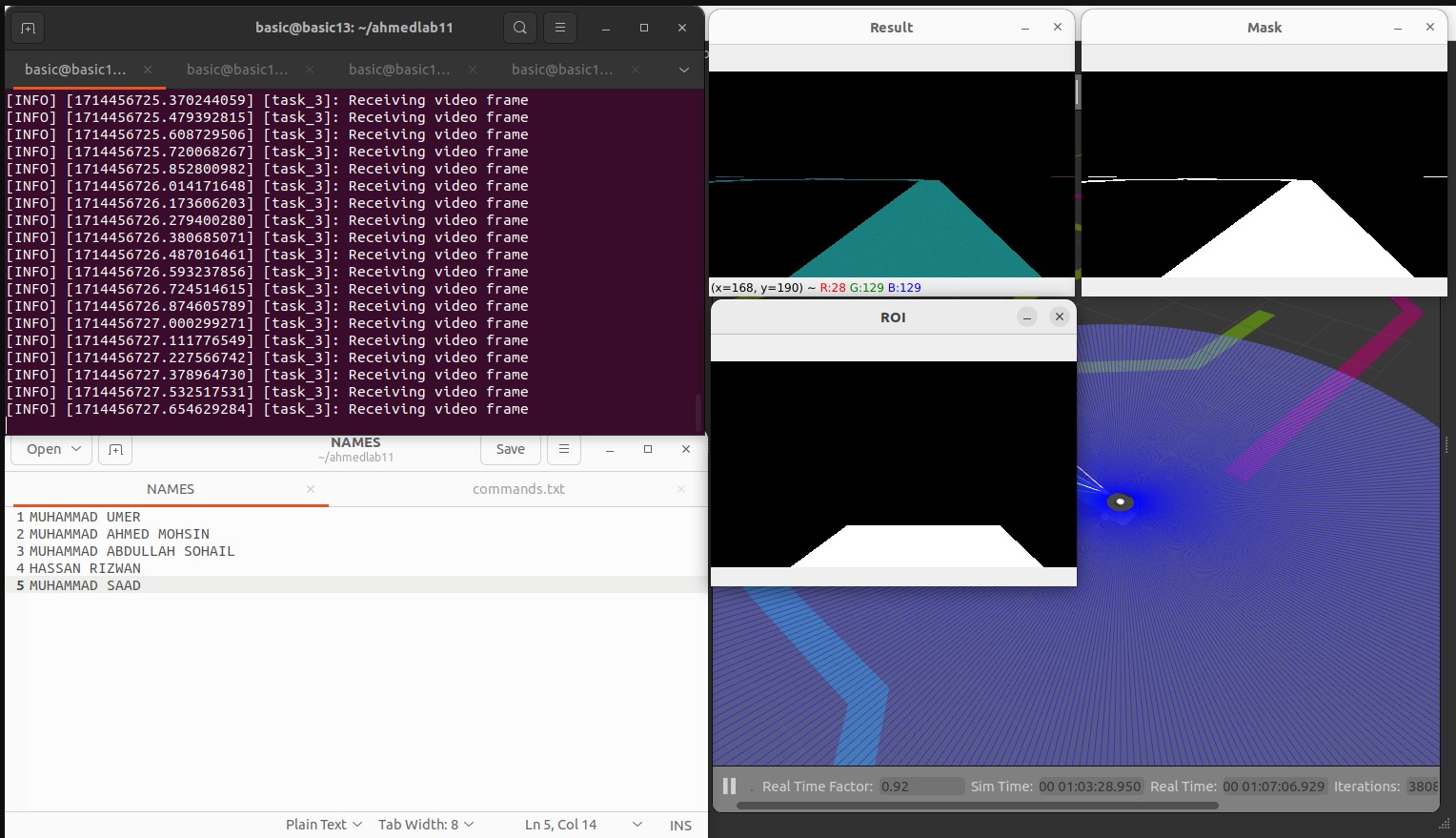
    node.destroy\_node()

    rclpy.shutdown()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Screenshot:**



*### REGION OF INTEREST SCREENSHOT ENDS HERE ###*

**Lab Task 4 – Centroid Indicator \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_[2]**

Using the region of interest from the previous task, get the centroid of the region and place a small circle in the centroid point. Test the simulation to see that the centroid tries to stay in the line center as the robot moves.

Next, make the circle change color as it moves farther from the image center. Ensure that the centroid is shown on the line in the original camera window. Provide a screenshot showing the centroid indicator.

***### CENTROID SCREENSHOT STARTS HERE ###***

import cv2

import numpy as np

import rclpy

from cv\_bridge import CvBridge

from geometry\_msgs.msg import Twist

from rclpy.node import Node

from sensor\_msgs.msg import Image

class InterestRegion(Node):

    def \_\_init\_\_(self):

        super().\_\_init\_\_("task\_3")

        self.subscription = self.create\_subscription(

            Image, "camera/image\_raw", self.callback, 10

        )

        self.subscription

        self.publisher\_ = self.create\_publisher(Twist, "cmd\_vel", 10)

        self.move = Twist()

        self.br = CvBridge()

    def callback(self, msg):

        self.get\_logger().info("Receiving video frame")

        img = self.br.imgmsg\_to\_cv2(msg, desired\_encoding="bgr8")

        img = cv2.resize(img, None, 1, 0.2, 0.2, cv2.INTER\_CUBIC)

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

        h\_min = 55

        h\_max = 179

        s\_min = 0

        s\_max = 255

        v\_min = 81

        v\_max = 255

        # making the mask

        low = np.array([h\_min, s\_min, v\_min])

        upp = np.array([h\_max, s\_max, v\_max])

        mask = cv2.inRange(img\_hsv, low, upp)

        result = cv2.bitwise\_and(img, img, mask=mask)

        # region of interest (keep the lower 1/5 of the image)

        height, width = mask.shape

        roi = deepcopy(mask)

        roi[: 4 \* height // 5, :] = 0

        # center of the region of interest

        M = cv2.moments(roi)

        if M["m00"] != 0:

            cX = int(M["m10"] / M["m00"])

            cY = int(M["m01"] / M["m00"])

        else:

            cX, cY = 0, 0

        # draw the center of the region of interest

        # change center color based on distance from the center

        center\_roi = (width // 2, height // 2)

        distance = np.sqrt((cX - center\_roi[0]) \*\* 2 + (cY - center\_roi[1]) \*\* 2)

        if distance < 120:

            color = (255, 0, 0)

        else:

            color = (0, 0, 255)

        centroid = np.zeros((height, width, 3), np.uint8)

        centroid[:, :] = cv2.cvtColor(roi, cv2.COLOR\_GRAY2BGR)

        cv2.circle(centroid, (cX, cY), 5, color, -1)

        cv2.imshow("Centroid", centroid)

        cv2.waitKey(2)

def main(args=None):

    rclpy.init(args=args)

    node = InterestRegion()

    rclpy.spin(node)

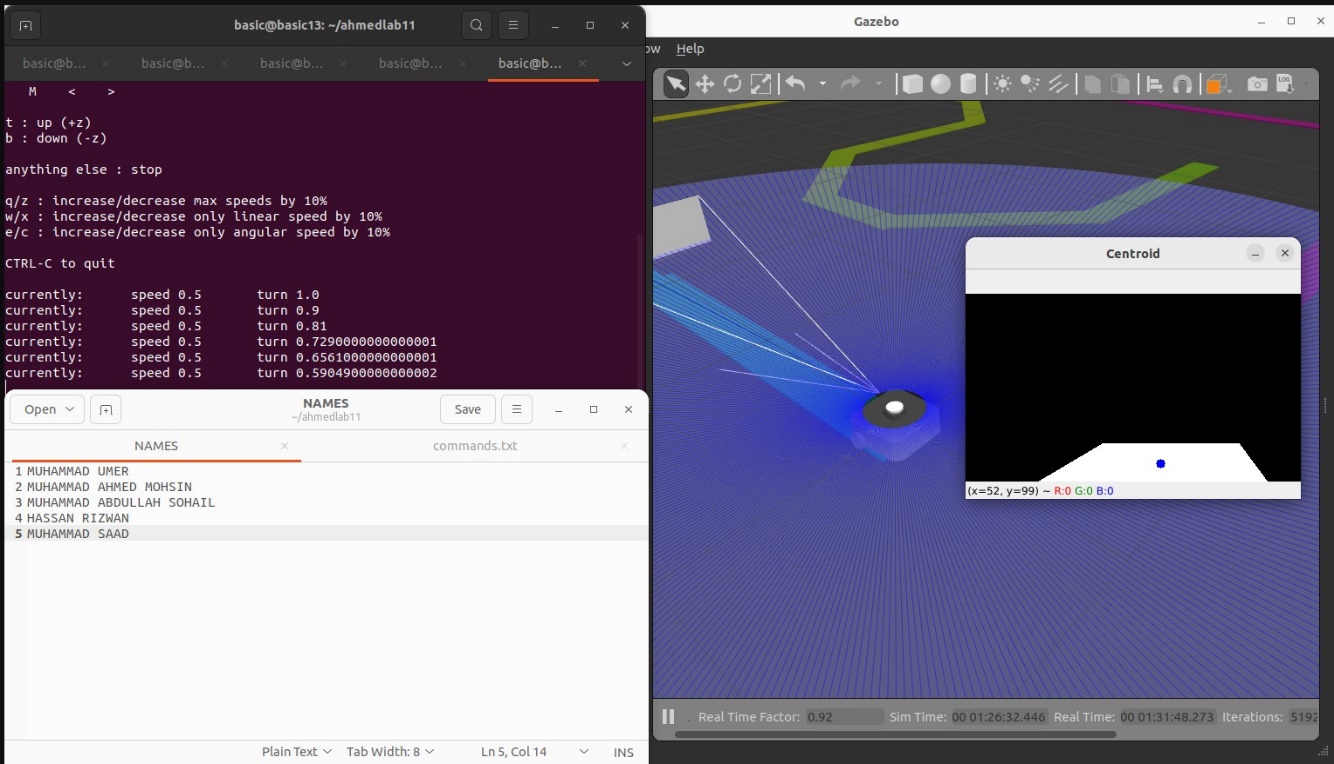
    node.destroy\_node()

    rclpy.shutdown()

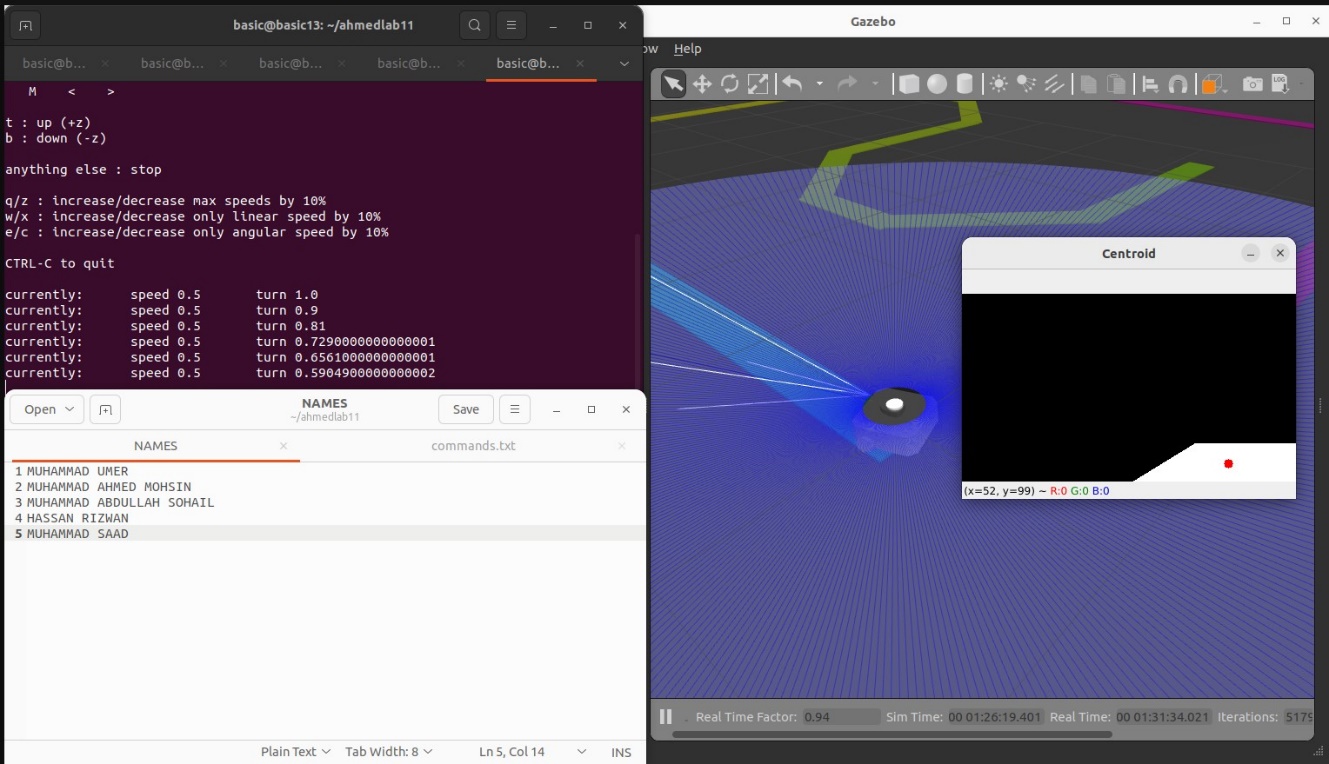
if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Screenshot A**



**Screenshot B**



*### CENTROID SCREENSHOT ENDS HERE ###*

**Lab Task 5 – Motion Control \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_[3]**

Using the centroid values from the previous task, make the robot move on the line by publishing appropriate twist messages. Linear velocity must be provided to keep the robot moving forward on the line. The shifting centroid will influence the angular velocity of the robot.

Make a video of the line following robot for the submission. During the video, you must type your names into the terminal before executing the program. Also, provide the final code of the line following node.

***### LINE FOLLOWING CODE STARTS HERE ###***

**Code:**

from copy import deepcopy

import cv2

import numpy as np

import rclpy

from cv\_bridge import CvBridge

from geometry\_msgs.msg import Twist

from rclpy.node import Node

from sensor\_msgs.msg import Image

class LineFollower(Node):

    def \_\_init\_\_(self):

        super().\_\_init\_\_("task\_3")

        self.subscription = self.create\_subscription(

            Image, "camera/image\_raw", self.callback, 10

        )

        self.subscription

        self.publisher\_ = self.create\_publisher(Twist, "cmd\_vel", 10)

        self.move = Twist()

        self.br = CvBridge()

    def callback(self, msg):

        self.get\_logger().info("Receiving video frame")

        img = self.br.imgmsg\_to\_cv2(msg, desired\_encoding="bgr8")

        img = cv2.resize(img, None, 1, 0.2, 0.2, cv2.INTER\_CUBIC)

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

        twist = Twist()

        linear\_x = 0.3

        h\_min = 55

        h\_max = 179

        s\_min = 0

        s\_max = 255

        v\_min = 81

        v\_max = 255

        # making the mask

        low = np.array([h\_min, s\_min, v\_min])

        upp = np.array([h\_max, s\_max, v\_max])

        mask = cv2.inRange(img\_hsv, low, upp)

        result = cv2.bitwise\_and(img, img, mask=mask)

        # region of interest (keep the lower 1/5 of the image)

        height, width = mask.shape

        roi = deepcopy(mask)

        roi[: 4 \* height // 5, :] = 0

        # roi = mask[4 \* height // 5 :, :]

        # center of the region of interest

        M = cv2.moments(roi)

        if M["m00"] != 0:

            cX = int(M["m10"] / M["m00"])

            cY = int(M["m01"] / M["m00"])

        else:

            cX, cY = 0, 0

        center\_roi = (width // 2, (4 \* height // 5) // 2)

        distance\_center = np.sqrt((cX - center\_roi[0]) \*\* 2 + (cY - center\_roi[1]) \*\* 2)

        # move the robot based on the distance from the center of the region of interest

        # follow the line by turning left or right

        if distance\_center < 120:

            twist.linear.x = linear\_x

            twist.angular.z = 0

        else:

            twist.linear.x = 0

            twist.angular.z = 0.3 if cX < center\_roi[0] else -0.3

        self.publisher\_.publish(twist)

def main(args=None):

    rclpy.init(args=args)

    node = LineFollower()

    rclpy.spin(node)

    node.destroy\_node()

    rclpy.shutdown()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

*### LINE FOLLOWING CODE ENDS HERE ###*

## Conclusion

In this lab, we successfully implemented a line-following algorithm using ROS and Gazebo simulation. By fine-tuning HSV values, extracting regions of interest, and implementing centroid control, we achieved accurate line detection and robot guidance. This hands-on experience provided valuable insights into computer vision-based robotics and laid a strong foundation for future developments in autonomous navigation systems.