Lab 6 – Exercise

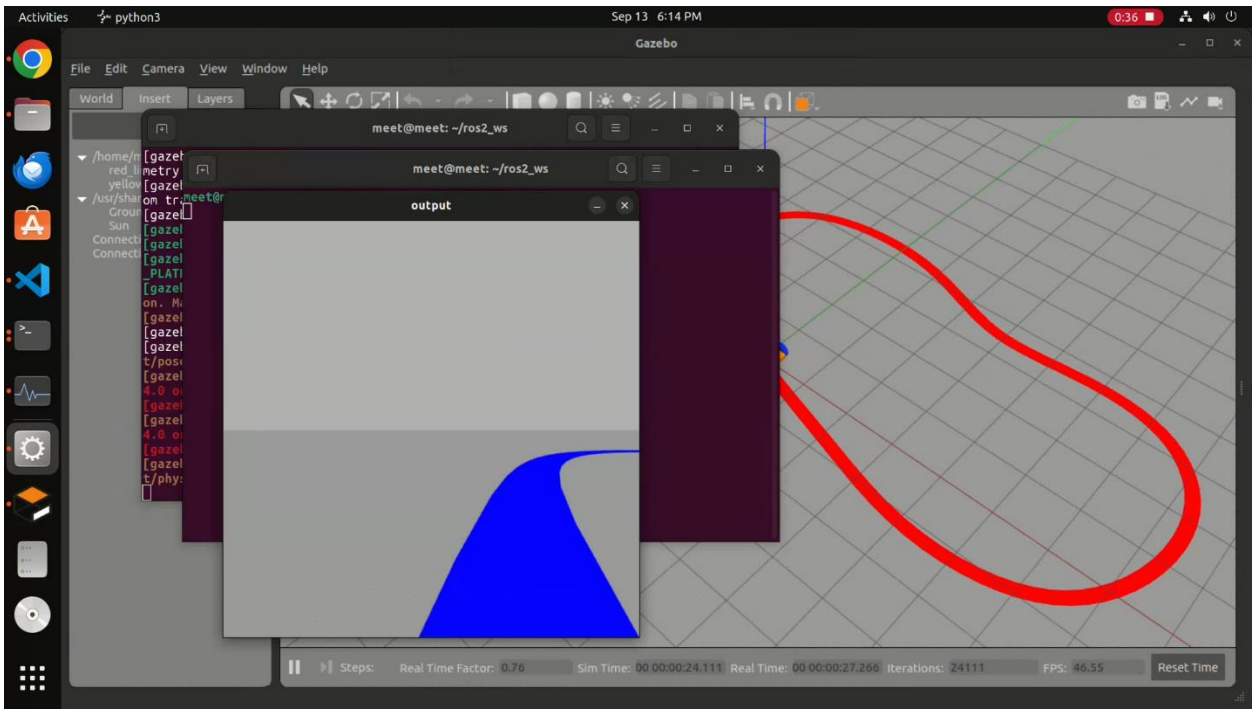
Meet Kansara – 220929270 Roll no. 54

Exercise: To simulate a line-following robot in Gazebo that uses OpenCV to detect and follow a red curved path.

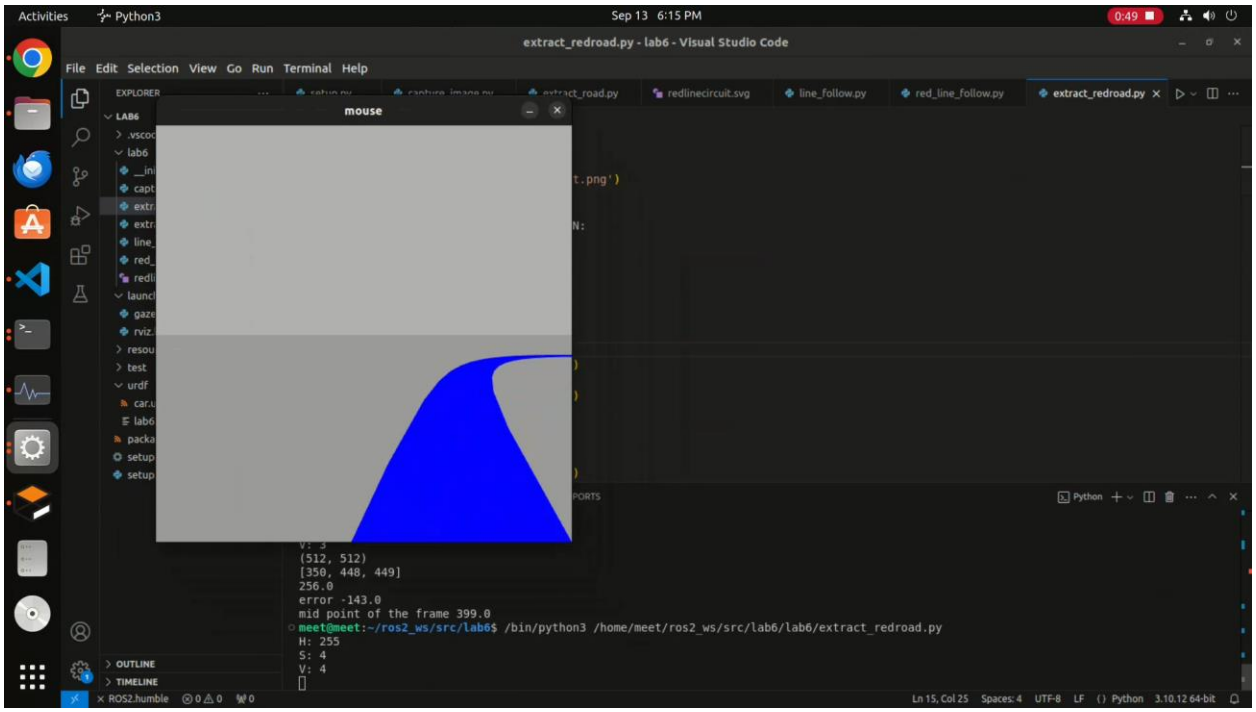
Code Execution and analysis:

1. Capturing image:

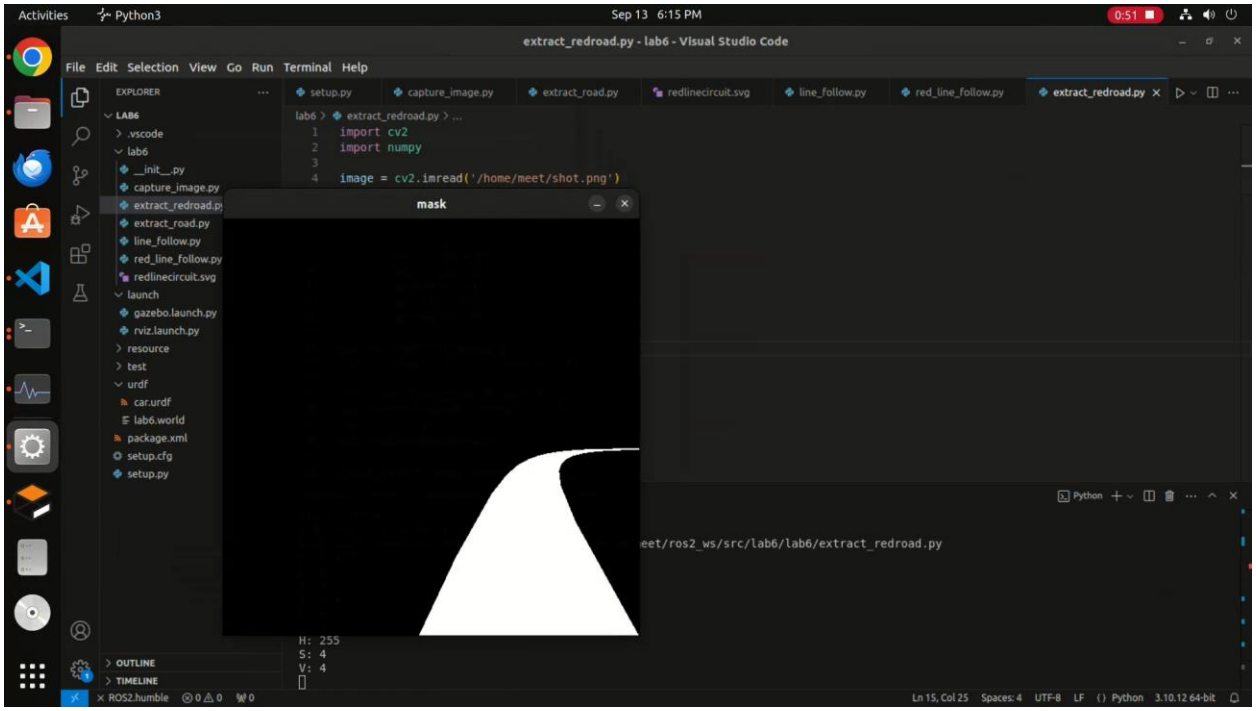
```
import rclpy
import cv2
from rclpy.node import Node
from cv_bridge import CvBridge
from sensor_msgs.msg import Image
class Capture(Node):
  def __init__(self):
    super().__init__('video_subscriber')
    self.subscriber = self.create_subscription(Image,'/camera1/image_raw',self.process_data,10)
    self.out = cv2.VideoWriter('/home/meet/output.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 10,
(512,512))
    self.bridge = CvBridge()
  def process_data(self, data):
    frame = self.bridge.imgmsg_to_cv2(data)
    self.out.write(frame)
    self.img = cv2.imwrite('/home/meet/shot.png', frame)
    cv2.imshow("output", frame)
    cv2.waitKey()
    cv2.destroyAllWindows()
def main(args=None):
  rclpy.init(args=args)
  node = Capture()
  rclpy.spin(node)
  rclpy.shutdown()
if __name__ == '__main__':
  main()
```



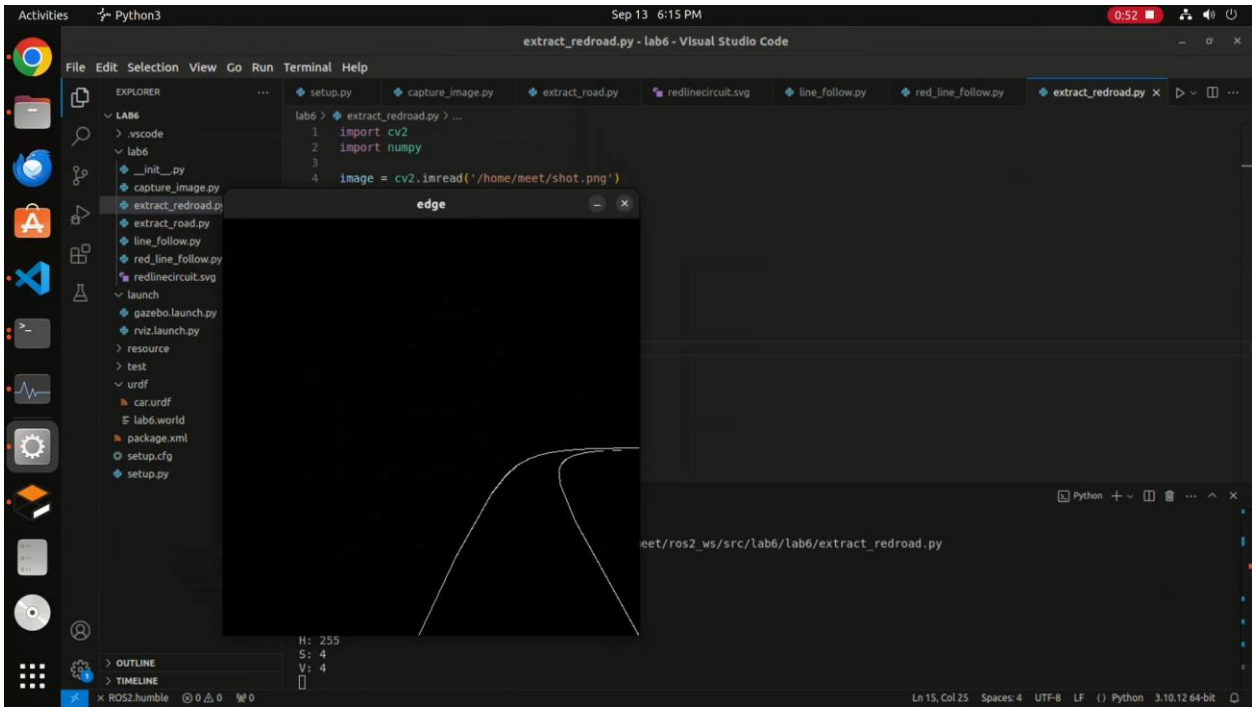
2. Extracting road from image:
    - Image



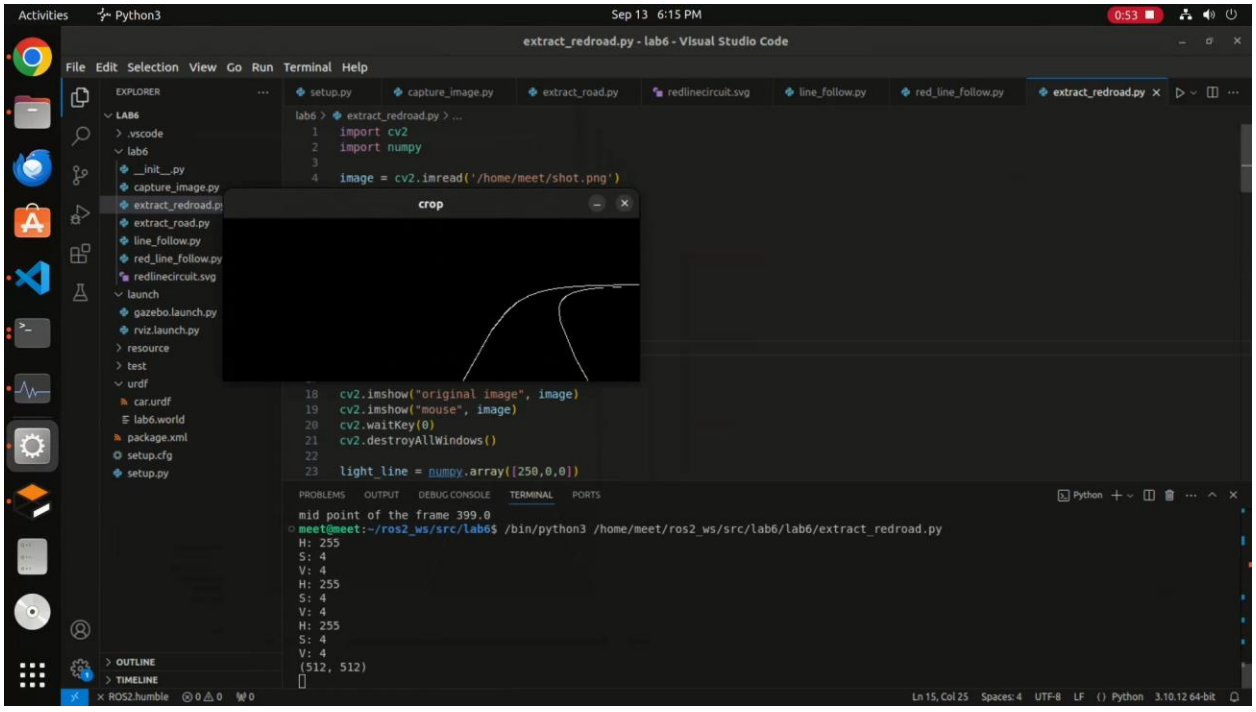
- Mask



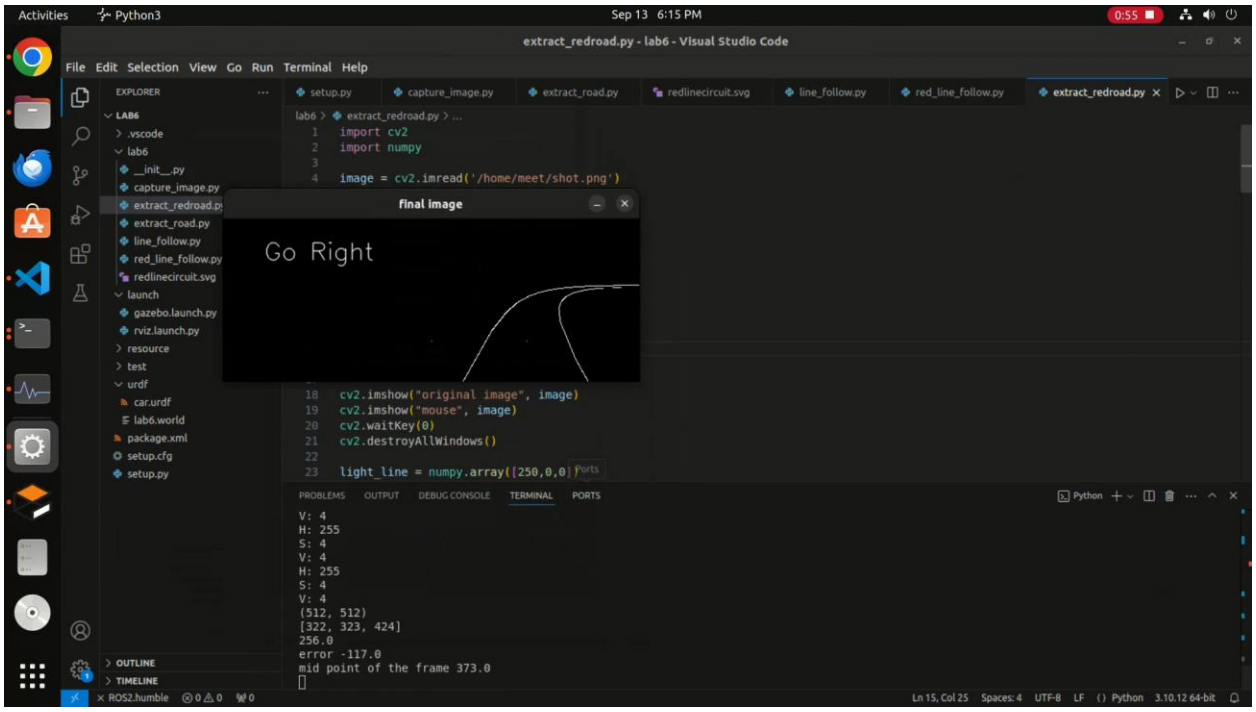
- Edge


Activities
Python3
Sep 13 6:15 PM
extract_redroad.py - lab6 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
edge
H: 255
S: 4
V: 4

- Crop


Activities
Python3
Sep 13 6:15 PM
extract_redroad.py - lab6 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
crop
mid point of the frame 399.0
meet@meet:~/ros2_ws/src/lab6$ /bin/python3 /home/meet/ros2_ws/src/lab6/lab6/extract_redroad.py
(512, 512)

- Directions

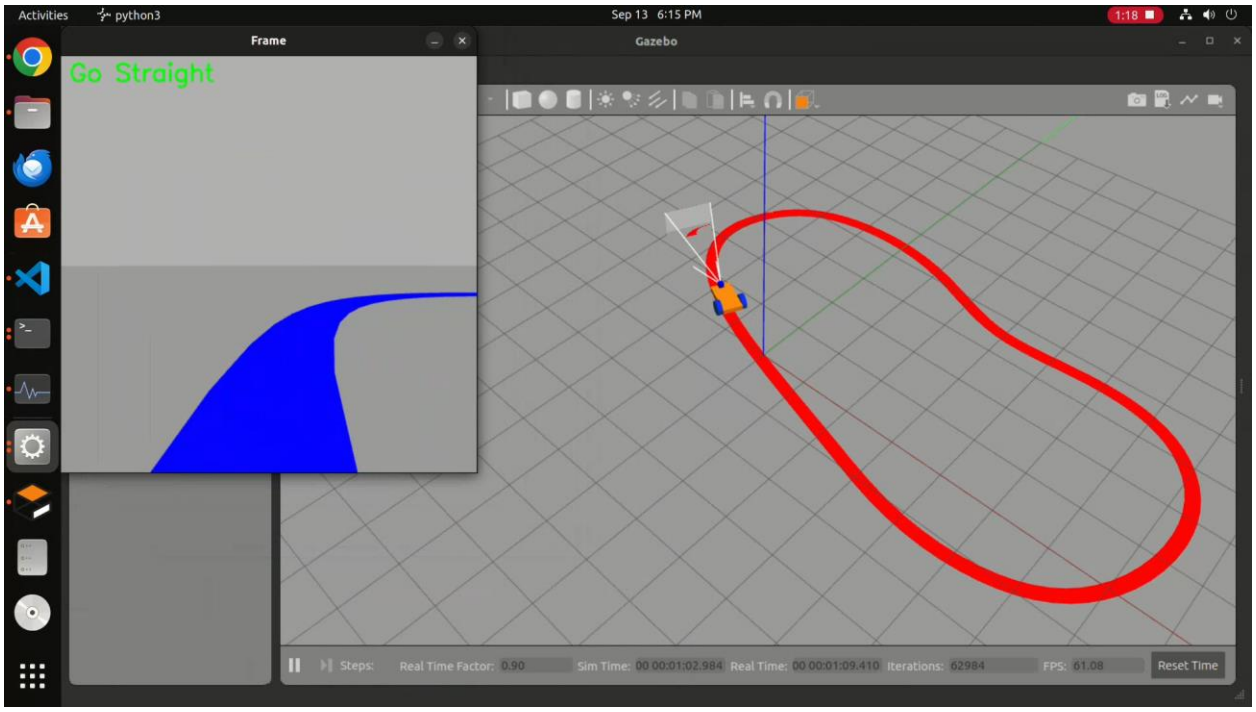


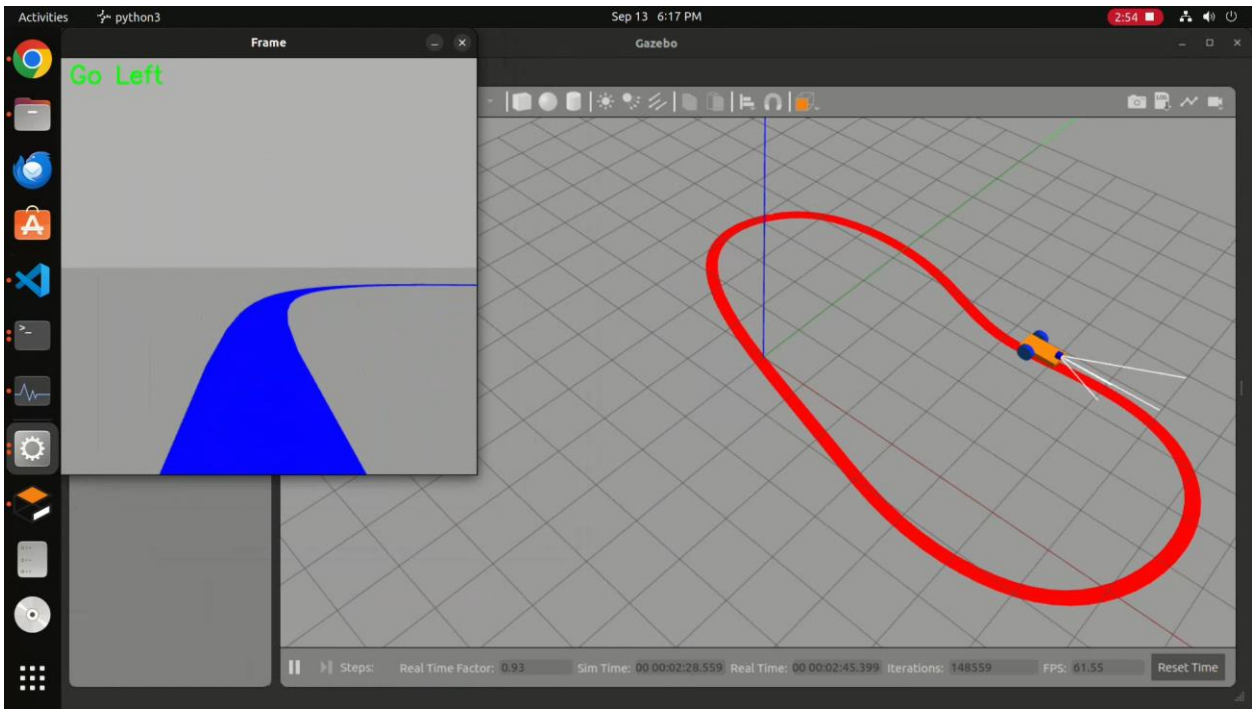
3. Line following:

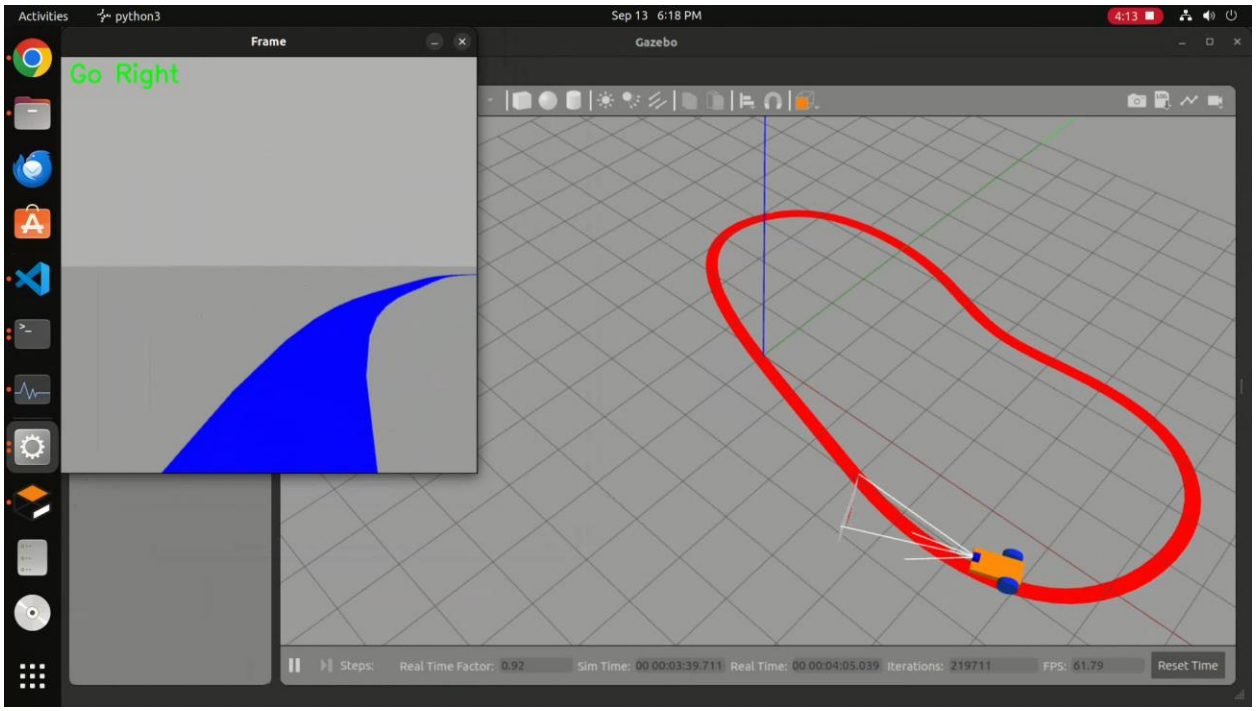
```
#!/usr/bin/env python3

import cv2
import numpy as np
import rclpy
from rclpy.node import Node
from cv_bridge import CvBridge
from sensor_msgs.msg import Image
from geometry_msgs.msg import Twist

class LineFollower(Node):
    def __init__(self):
        super().__init__('line_follower')
        self.bridge = CvBridge()
        self.subscriber = self.create_subscription(Image, '/camera1/image_raw', self.process_data,
10)
        self.publisher = self.create_publisher(Twist, '/cmd_vel', 40)
        self.timer = self.create_timer(0.1, self.send_cmd_vel)
        self.velocity = Twist()
        self.error = 0
        self.last_error = 0
        self.integral = 0
        self.action = ""
        self.get_logger().info("Node Started!")

        # PID constants
        self.Kp = 0.003
        self.Ki = 0.0001
        self.Kd = 0.005
        # Speed constants
        self.MAX_SPEED = 0.2
        self.MIN_SPEED = 0.05
        self.MAX_ANGULAR_SPEED = 0.5
```

```
    def send_cmd_vel(self):
        if abs(self.error) < 10:
            self.velocity.linear.x = self.MAX_SPEED
            self.velocity.angular.z = 0.0
            self.action = "Go Straight"
        else:
            # PID control
            P = self.Kp * self.error
            self.integral += self.error
            I = self.Ki * self.integral
            D = self.Kd * (self.error - self.last_error)

            angular_z = P + I + D
            angular_z = max(min(angular_z, self.MAX_ANGULAR_SPEED), -self.MAX_ANGULAR_SPEED)

            self.velocity.linear.x = max(self.MAX_SPEED - abs(angular_z), self.MIN_SPEED)
            self.velocity.angular.z = angular_z

            self.action = "Adjusting" if abs(angular_z) < 0.1 else ("Go Left" if angular_z > 0 else
"Go Right")

        self.publisher.publish(self.velocity)
        self.last_error = self.error

    def process_data(self, data):
        self.get_logger().info("Image Received!")
        frame = self.bridge.imgmsg_to_cv2(data)

        # Use the original HSV values
        light_line = np.array([250,0,0])
        dark_line = np.array([255,10,10])
        mask = cv2.inRange(frame, light_line, dark_line)

        # Crop the bottom part of the image
        h, w = mask.shape
        crop_h = int(h * 0.8)
        cropped = mask[crop_h:h, :]

        # Find contours
        contours, _ = cv2.findContours(cropped, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        if contours:
            # Find the largest contour
            largest_contour = max(contours, key=cv2.contourArea)
            M = cv2.moments(largest_contour)

            if M["m00"] != 0:
                cx = int(M["m10"] / M["m00"])
                self.error = w // 2 - cx
            else:
                self.error = 0
        else:
            self.error = 0

        cv2.putText(frame, self.action, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        cv2.imshow('Frame', frame)
        cv2.imshow('Mask', mask)
        cv2.waitKey(1)

def main(args=None):
    rclpy.init(args=args)
    node = LineFollower()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

- Robot following red curve







Conclusion The robot successfully followed the red curve using OpenCV for real-time image processing in Gazebo.