GPIO PINS	PWM0/PWM1
GPIO12	PWM0
GPIO18	PWM0
GPIO13	PWM1
GPIO19	PWM1

Table 3.4 Raspberry Pi PWM Pins

Computer Vision OpenCV

OpenCV

OpenCV, or **Open Source Computer Vision Library**, is an open-source computer vision and machine learning software library. It provides a variety of tools and functions that help developers and researchers work with computer vision applications. Here's a brief overview:

1. **Image Processing:** OpenCV includes a wide range of functions for image processing, such as filtering, edge detection, morphological operations, and more.
2. **Computer Vision Algorithms:** It implements various computer vision algorithms, including feature detection, object recognition, and image stitching.
3. **Machine Learning Integration:** OpenCV provides tools to work with machine learning frameworks. It includes machine learning algorithms for classification and clustering.
4. **Camera Calibration:** OpenCV supports camera calibration, essential for computer vision applications like 3D reconstruction.
5. **Video Analysis:** It enables the analysis of video streams, allowing tasks such as motion detection, object tracking, and video stabilization.
6. **Cross-platform:** OpenCV is cross-platform and can be used on Windows, Linux, macOS, Android, and iOS.
7. **Community and Documentation:** OpenCV has a large and active community, and there is extensive documentation and tutorials available, making it easier for developers to get started.
8. **Open Source:** OpenCV is distributed under an open-source license (BSD license), allowing users to modify and distribute the source code freely.

OpenCV with HSV (Hue, Saturation, Value) Color Space

OpenCV is a powerful computer vision library commonly used for image and video processing. When working with color, OpenCV provides support for the HSV (Hue, Saturation, Value) color space, which is closer to how humans perceive color.

Components of HSV:

1. **Hue (H):** Represents the color, expressed as a number from 0 to 360 degrees. Specific color ranges:
   - Red: 0-60 degrees
   - Yellow: 61-120 degrees
   - Green: 121-180 degrees
   - Cyan: 181-240 degrees
   - Blue: 241-300 degrees
   - Magenta: 301-360 degrees

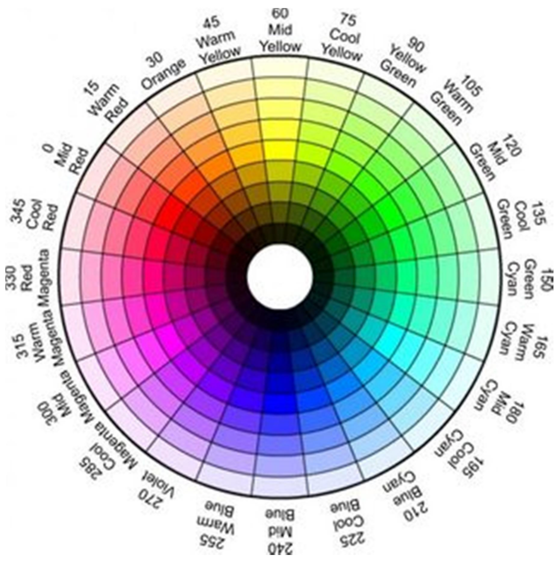

Figure 3.38 HSV Color Wheel

2. **Saturation (S):** Describes the amount of gray in a color, ranging from 0 to 100 percent. Reducing saturation introduces more gray, creating a faded effect. Sometimes expressed as a range from 0 to 1, where 0 is gray, and 1 is a primary color.
3. **Value (V) or Brightness:** Works in conjunction with saturation. Describes the brightness or intensity of the color, ranging from 0 to 100 percent. 0 is completely black, and 100 is the brightest, revealing the most color.

Computer Vision & Motion - Edge/ Lane Detection or Road Following

Lane Detection Steps:

Lane detection involves several key image processing steps to identify and delineate lanes on a road. Here's a summary of the relevant processes:

1. **Grayscale Conversion:** Simplifies image analysis by reducing dimensionality, enhancing computational efficiency, and ensuring compatibility. Grayscale images are simpler to interpret compared to color images.
2. **Edge Detection:** Utilizes operators, such as the Canny edge detector, to identify significant edges in the image.

   Gradient-based and Gaussian-based operators compute first and second-order derivations, respectively.
3. **Image Filtering:** Involves applying filters or kernels to modify the image's appearance or extract specific features. Gaussian Blur is commonly used for smoothing, reducing

noise, and removing high-frequency details.

4. **Convolution:** Applies a mathematical operator to each pixel using a kernel matrix, modifying pixel values.
5. **Segmentation:** Involves partitioning the image into multiple regions where pixels share common characteristics. Thresholding, a widely used segmentation technique, is employed.
6. **Hough Transform:** A feature extraction method for detecting simple shapes, such as lines, circles, etc., in an image. Useful for identifying and representing lanes in lane detection algorithms.

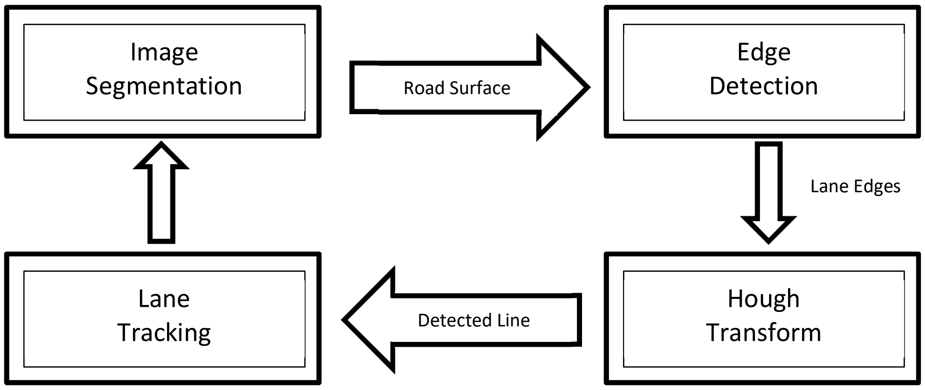

Figure 3.39 Lane Detection Steps

4.3 Lane Following Robot

Objective

To be able to integrate USB cameras to detect lines or take turns based on colors.

Challenge Brief

The challenge is to build a lane-following robot using a USB camera which is integrated with Raspberry Pi that can successfully navigate a lane with twists and turns. The lane is a black line on a white surface, and the bot should be able to follow the lane without deviating from it. The bot should be able to move forward, turn left or right based on colors seen on the arena and stop when it reaches the end of the blue box. It should also be able to adjust its speed based on the complexity of the lane.

Solution

Components Required

1. Raspberry Pi
2. L298N Motor Driver Module
3. DC Motors (2)
4. USB camera
5. Jumper Wires
6. Chassis and Wheels
7. Power Supply (9V Battery for Motor Driver and Power Bank for Raspberry Pi)

Connection

Motor Driver Connections:

1. Connect the input pins (in1, in2, in3, in4) to the corresponding GPIO pins.
2. Connect the enable pins (en1, en2) to the PWM-capable GPIO pins.
3. in1 = 3, in2 = 5, in3 = 29, in4 = 31, enA = 32 and enB = 33 (GPIO.BOARD mode refers to the physical pin numbering on the Raspberry Pi's header)

DC Motor Connections:

1. Connect the DC motors to the output terminals of the motor driver.

Power Supply:

1. Connect the +12V input of the L298N to the external power source's positive terminal.
2. Connect the GND input of the L298N to the external power source's negative terminal.
3. Raspberry Pi to power bank

Camera Module:

1. Connect the USB camera to the dedicated camera port on the Raspberry Pi.

Circuit Diagram

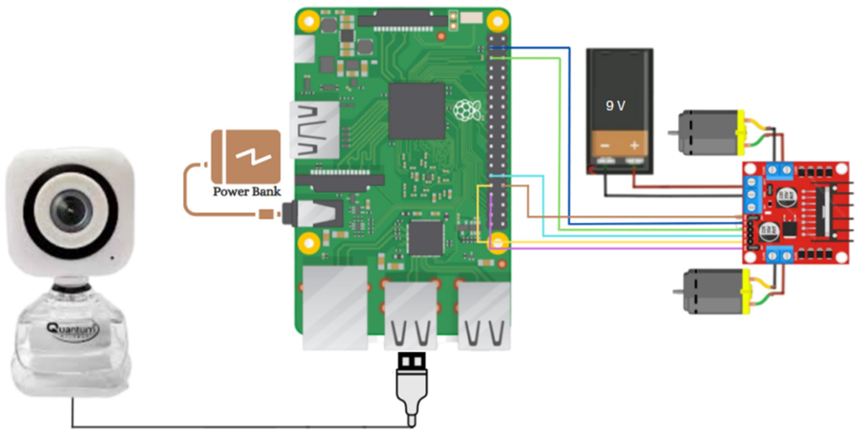

Figure 3.44 Circuit Diagram of Lane Following Robot

Code

```
import numpy as np
import RPi.GPIO as GPIO
import cv2
import time

cap = cv2.VideoCapture(0)
cap.set(3, 160)
cap.set(4, 120)
cap.set(cv2.CAP_PROP_FPS, 20)
fps = int(cap.get(5))
print("fps:",fps)

in1 = 3
in2 = 5
in3 = 29
in4 = 31
enA = 32
enB = 33
c=""

GPIO.setmode(GPIO.BOARD)
GPIO.setup(enA, GPIO.OUT)
GPIO.setup(enB, GPIO.OUT)
GPIO.setup(in1, GPIO.OUT)
GPIO.setup(in2, GPIO.OUT)
GPIO.setup(in3, GPIO.OUT)
GPIO.setup(in4, GPIO.OUT)

p1 = GPIO.PWM(enA, 30)
p2 = GPIO.PWM(enB, 30)
p1.start(15)
p2.start(15)

def forward():
    GPIO.output(in1, GPIO.HIGH)
    GPIO.output(in2, GPIO.LOW)
    GPIO.output(in3, GPIO.HIGH)
    GPIO.output(in4, GPIO.LOW)
    p1.start(13)
    p2.start(13)

def backward():
    GPIO.output(in1, GPIO.LOW)
    GPIO.output(in2, GPIO.HIGH)
    GPIO.output(in3, GPIO.LOW)
    GPIO.output(in4, GPIO.HIGH)

def left():
    GPIO.output(in1, GPIO.HIGH)
    GPIO.output(in2, GPIO.LOW)
    GPIO.output(in3, GPIO.LOW)
    GPIO.output(in4, GPIO.HIGH)
    p1.start(12)
    p2.start(15)

def right():
    GPIO.output(in1, GPIO.LOW)
    GPIO.output(in2, GPIO.HIGH)
    GPIO.output(in3, GPIO.HIGH)
    GPIO.output(in4, GPIO.LOW)
    p1.start(15)
    p2.start(12)

def stop():
    GPIO.output(in1, GPIO.LOW)
    GPIO.output(in2, GPIO.LOW)
    GPIO.output(in3, GPIO.LOW)
    GPIO.output(in4, GPIO.LOW)

stop()

try:
    while True:
        ret, frame = cap.read()
        hsvFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

# Set range for red color and define mask
```

```
        red_lower = np.array([136, 87, 111], np.uint8)
        red_upper = np.array([180, 255, 255], np.uint8)
        red_mask = cv2.inRange(hsvFrame, red_lower, red_upper)

    # Set range for green color and define mask
        green_lower = np.array([25, 52, 72], np.uint8)
        green_upper = np.array([102, 255, 255], np.uint8)
        green_mask = cv2.inRange(hsvFrame, green_lower, green_upper)

    # Set range for blue color and define mask
        blue_lower = np.array([94, 80, 2], np.uint8)
        blue_upper = np.array([120, 255, 255], np.uint8)
        blue_mask = cv2.inRange(hsvFrame, blue_lower, blue_upper)

        kernel = np.ones((5, 5), "uint8")

            # For red color
        red_mask = cv2.dilate(red_mask, kernel)
        res_red = cv2.bitwise_and(frame, frame,mask = red_mask)

    # For green color
        green_mask = cv2.dilate(green_mask, kernel)
        res_green = cv2.bitwise_and(frame, frame,mask = green_mask)

    # For blue color
        blue_mask = cv2.dilate(blue_mask, kernel)
        res_blue = cv2.bitwise_and(frame, frame,mask = blue_mask)

        low_b = np.uint8([80,80,80])
        high_b = np.uint8([0,0,0])
        mask = cv2.inRange(frame, high_b, low_b)
        contours, hierarchy = cv2.findContours(mask, 1,
cv2.CHAIN_APPROX_NONE)
        if len(contours) > 0 :
            c = max(contours, key=cv2.contourArea)
            M = cv2.moments(c)
            if M["m00"] !=0 :
                cx = int(M['m10']/M['m00'])
                cy = int(M['m01']/M['m00'])
                print("CX : "+str(cx)+"  CY : "+str(cy))
                if cx >= 120 :
                    print("Turn Right")
                    right()

                if cx < 120 and cx > 40 :
                    print("On Track!")
                    forward()
                if cx <=40 :
                    print("Turn Left")
                    left()
                cv2.circle(frame, (cx,cy), 5, (255,255,255), -1)
        else :
            print("I don't see the line")
            #time.sleep(1)
            stop()
            time.sleep(1)

            # Creating contour to track green color
            contours, hierarchy =
cv2.findContours(green_mask,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

            for pic, contour in enumerate(contours):
                area = cv2.contourArea(contour)
                if(area > 300):
                    print("Green Detected")
                    p1.ChangeDutyCycle(10)
                    p2.ChangeDutyCycle(10)
                    left()
                    time.sleep(0.3)

            # Creating contour to track blue color
            contours, hierarchy =
cv2.findContours(blue_mask,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

            for pic, contour in enumerate(contours):
                area = cv2.contourArea(contour)
                if(area > 300):
                    print("Blue Detected")
                    #forward()
                    time.sleep(14)
                    stop()
                    #time.sleep(5)
                    GPIO.cleanup()
                    cap.release()
                    cv2.destroyAllWindows()

             # Creating contour to track red color
            contours, hierarchy =
cv2.findContours(red_mask,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

            for pic, contour in enumerate(contours):
                area = cv2.contourArea(contour)
                if(area > 300):
                    print("Red Detected")
                    p1.ChangeDutyCycle(10)
                    p2.ChangeDutyCycle(10)
                    right()
                    time.sleep(0.6)
        cv2.drawContours(frame, c, -1, (0,255,0), 1)
        cv2.imshow("Mask",mask)
        cv2.imshow("Frame",frame)
        if cv2.waitKey(1) & 0xff == ord('q'):  # 1 is the time in ms
            stop()
            break
finally:
    GPIO.cleanup()
    cap.release()
    cv2.destroyAllWindows()
```

Constructed Model

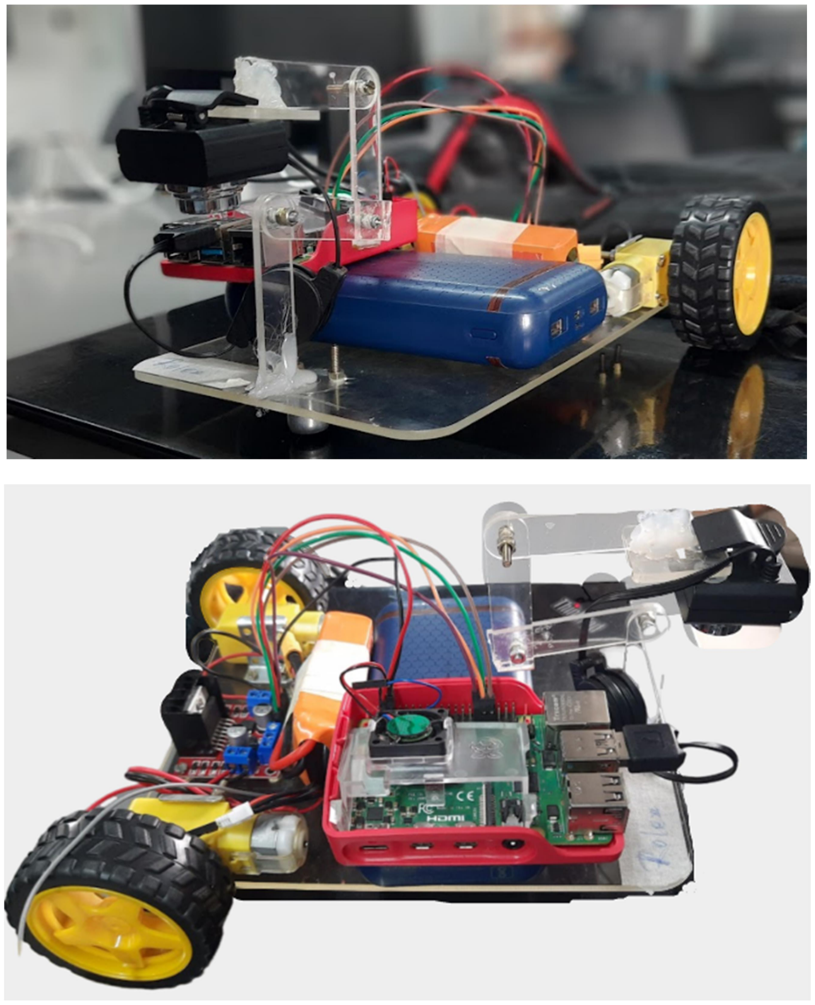

Figure 3.45 Constructed Model of the Lane Following Robot

Execution Video

https://drive.google.com/file/d/1nKAb7WtzY59dmGCOwx5N86YkGQy6cIBK/view?usp=sharing