**COMPUTER VISION**

Obstacle Avoidance Robot

An obstacle avoidance robot using **Canny edge detection**:

1. **Capturing real-time video** from a camera (e.g., Raspberry Pi Camera or USB camera).
2. **Applying Canny edge detection** to detect edges of obstacles.
3. **Processing the detected edges** to identify free paths.
4. **Controlling the robot's movement** based on detected obstacles (using motors and a microcontroller like Raspberry Pi).

**How it Works**

1. **Canny Edge Detection** detects obstacle edges.
2. **Splits image into left & right regions**:
   * **More edges on the left?** → Turn right.
   * **More edges on the right?** → Turn left.
   * **Edges on both sides?** → Stop.
   * **No edges detected?** → Move forward.
3. **Motor control via Raspberry Pi GPIO.**

**Steps to Achieve 85% Accuracy:**

* **Preprocessing**: Gaussian filtering to remove noise before Canny edge detection.
* **Adaptive Edge Detection**: Tuning high/low threshold values dynamically based on lighting conditions.
* **Region of Interest (ROI)**: Focusing on specific areas in the image to identify obstacles efficiently.
* **Machine Learning Integration**: Training a lightweight ML model (e.g., SVM, CNN) to refine edge-based detection.
* **Sensor Fusion**: Combining **ultrasonic sensors** with Canny edges to improve accuracy.
* **Accuracy Improvements:**
* **Fine-Tune Canny Thresholds**: Adjust dynamically using adaptive thresholds.  
  **Use Morphological Operations**: Improve edge detection accuracy with cv2.dilate() and cv2.erode().  
  **Combine with Ultrasonic Sensor**: Ensure no missed obstacles using sensor fusion.  
  **CNN:** Use a small neural network for refined path prediction.

Basic Canny edge detection:

import cv2

import numpy as np

# Load the image

image = cv2.imread('image.jpg', cv2.IMREAD\_GRAYSCALE)

# Apply Gaussian Blur to reduce noise

blurred = cv2.GaussianBlur(image, (5, 5), 0)

# Apply Canny Edge Detection

edges = cv2.Canny(blurred, 50, 150) # (low\_threshold, high\_threshold)

# Display results

cv2.imshow('Original Image', image)

cv2.imshow('Canny Edges', edges)

cv2.waitKey(0)

cv2.destroyAllWindows()

Sample code:

import cv2

import numpy as np

import RPi.GPIO as GPIO

import time

# Motor GPIO Pins

LEFT\_MOTOR\_FORWARD = 17

LEFT\_MOTOR\_BACKWARD = 27

RIGHT\_MOTOR\_FORWARD = 22

RIGHT\_MOTOR\_BACKWARD = 23

# GPIO Setup

GPIO.setmode(GPIO.BCM)

GPIO.setup(LEFT\_MOTOR\_FORWARD, GPIO.OUT)

GPIO.setup(LEFT\_MOTOR\_BACKWARD, GPIO.OUT)

GPIO.setup(RIGHT\_MOTOR\_FORWARD, GPIO.OUT)

GPIO.setup(RIGHT\_MOTOR\_BACKWARD, GPIO.OUT)

def move\_forward():

GPIO.output(LEFT\_MOTOR\_FORWARD, GPIO.HIGH)

GPIO.output(RIGHT\_MOTOR\_FORWARD, GPIO.HIGH)

GPIO.output(LEFT\_MOTOR\_BACKWARD, GPIO.LOW)

GPIO.output(RIGHT\_MOTOR\_BACKWARD, GPIO.LOW)

def move\_backward():

GPIO.output(LEFT\_MOTOR\_FORWARD, GPIO.LOW)

GPIO.output(RIGHT\_MOTOR\_FORWARD, GPIO.LOW)

GPIO.output(LEFT\_MOTOR\_BACKWARD, GPIO.HIGH)

GPIO.output(RIGHT\_MOTOR\_BACKWARD, GPIO.HIGH)

def turn\_left():

GPIO.output(LEFT\_MOTOR\_FORWARD, GPIO.LOW)

GPIO.output(RIGHT\_MOTOR\_FORWARD, GPIO.HIGH)

GPIO.output(LEFT\_MOTOR\_BACKWARD, GPIO.HIGH)

GPIO.output(RIGHT\_MOTOR\_BACKWARD, GPIO.LOW)

def turn\_right():

GPIO.output(LEFT\_MOTOR\_FORWARD, GPIO.HIGH)

GPIO.output(RIGHT\_MOTOR\_FORWARD, GPIO.LOW)

GPIO.output(LEFT\_MOTOR\_BACKWARD, GPIO.LOW)

GPIO.output(RIGHT\_MOTOR\_BACKWARD, GPIO.HIGH)

def stop():

GPIO.output(LEFT\_MOTOR\_FORWARD, GPIO.LOW)

GPIO.output(RIGHT\_MOTOR\_FORWARD, GPIO.LOW)

GPIO.output(LEFT\_MOTOR\_BACKWARD, GPIO.LOW)

GPIO.output(RIGHT\_MOTOR\_BACKWARD, GPIO.LOW)

# Capture Video

cap = cv2.VideoCapture(0)

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

# Convert to grayscale and apply Gaussian Blur

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# Apply Canny Edge Detection

edges = cv2.Canny(blurred, 50, 150)

# Split into left and right regions for decision making

height, width = edges.shape

left\_region = edges[:, :width//2]

right\_region = edges[:, width//2:]

# Count white pixels (edges)

left\_edges = np.sum(left\_region == 255)

right\_edges = np.sum(right\_region == 255)

# Decision Making

threshold = 5000 # Tune based on testing

if left\_edges > threshold and right\_edges > threshold:

stop()

elif left\_edges > threshold:

turn\_right()

elif right\_edges > threshold:

turn\_left()

else:

move\_forward()

# Display processed frame

cv2.imshow('Edges', edges)

# Exit with 'q' key

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

break

# Cleanup

cap.release()

cv2.destroyAllWindows()

GPIO.cleanup()