Obstacle Management

Contents

[Introduction 1](#_Toc179063471)

[System Architecture 1](#_Toc179063472)

[Flow Diagram for Obstacle Detection and Management 1](#_Toc179063473)

[Pseudo Code for Combined Obstacle Detection 2](#_Toc179063474)

[Visual Obstacle Detection with Raspberry Pi Camera 2](#_Toc179063475)

[Example Code Snippet (Python with OpenCV) 2](#_Toc179063476)

[Obstacle Avoidance and Navigation 3](#_Toc179063477)

[Pseudo Code for Obstacle Avoidance 3](#_Toc179063478)

# Introduction

The prime component used for Obstacle avoidance is the Raspberry Pi Camera V2 alongside an Arduino Mega, GY-271 magnetometer, HC-SR04 ultrasonic sensors, L298N motor driver, and a servo motor to significantly enhance the vehicle's ability to manage and navigate through obstacles. This setup combines ultrasonic distance measurement with advanced visual processing, enabling the vehicle to understand its environment more comprehensively.

# System Architecture

The system leverages the Raspberry Pi for image processing and decision-making, while the Arduino Mega handles real-time sensor data collection and actuator control. The GY-271 provides orientation data, crucial for navigation; the HC-SR04 sensors detect nearby obstacles through ultrasonic waves; the L298N driver controls the motors; and the servo motor adjusts the steering angle.

Obstacle Detection Strategy

1. **Ultrasonic Detection**: The HC-SR04 sensors continuously scan the front and sides of the vehicle for obstacles, providing immediate distance measurements.
2. **Visual Detection**: The Raspberry Pi Camera Module V2 captures real-time video. Image processing algorithms identify obstacles by shape, size, and colour.

# Flow Diagram for Obstacle Detection and Management

1. Collect data from HC-SR04 sensors and Camera Module.
2. If an obstacle is detected by either system:
   * Assess the obstacle's position, size, and colour.
   * Determine the next course of action depending on the colour:
     1. If green, then turn through left
     2. If red, then turn through right
3. Execute one of the above chosen actions based on the colour.
4. Continue navigation.

## Pseudo Code for Combined Obstacle Detection

1. Initialize Ultrasonic Sensors (left\_sensor, right\_sensor)

2. Initialize Gyroscope (gyro\_sensor)

3. Set target\_heading = current reading from gyro\_sensor

4. Initialize Motor Controls (left\_motor, right\_motor)

5. Function: Check\_Obstacles()

a. Read distance from left\_sensor

b. Read distance from right\_sensor

c. If left\_sensor < threshold OR right\_sensor < threshold:

Return True (obstacle detected)

d. Else:

Return False (no obstacle)

6. Function: Adjust\_Course()

a. current\_heading = read gyro\_sensor

b. If current\_heading deviates from target\_heading:

Calculate correction based on difference

Adjust left\_motor and right\_motor to steer towards target\_heading

7. Main Loop:

While True:

a. If Check\_Obstacles() = True:

Stop motors

If left\_sensor < threshold:

Rotate right to avoid obstacle

Else If right\_sensor < threshold:

Rotate left to avoid obstacle

Update target\_heading after avoiding

b. Else:

Move forward

Adjust\_Course()

This pseudo code outlines the logic for integrating ultrasonic and visual obstacle detection. This will prioritize the ultrasonic data but also consider the visual information to understand the surrounding environment and take relevant actions.

# Visual Obstacle Detection with Raspberry Pi Camera

Using vision techniques, the Raspberry Pi analyzes video input to detect obstacles. This can involve color detection, shape recognition, or machine learning models to identify specific objects.

## Example Code Snippet (Python with OpenCV)

1. Import Libraries:

a. Import OpenCV for image processing

b. Import necessary Raspberry Pi GPIO libraries (for motor control, etc.)

c. Import PiCamera module for camera interface

2. Initialize Camera:

a. Start PiCamera

b. Set camera parameters (resolution, frame rate, etc.)

3. Function: Process\_Frame(image)

a. Convert image to grayscale

b. Apply Gaussian Blur to reduce noise

c. Apply Canny Edge Detection to detect edges

d. Find contours in the edge-detected image

e. If significant contours (obstacles) are found:

Return True (obstacle detected)

f. Else:

Return False (no obstacle)

4. Function: Adjust\_Course()

a. Use the center of detected contours (if any) to determine obstacle position

b. If obstacle is towards the left side of the frame:

Adjust motors to steer right

c. If obstacle is towards the right side of the frame:

Adjust motors to steer left

d. If no obstacle is detected:

Move forward

5. Main Loop:

While True:

a. Capture frame from PiCamera

b. If Process\_Frame(frame) = True:

Stop motors

Call Adjust\_Course() to avoid obstacle

c. Else:

Move forward

This snippet demonstrates initializing the camera and processing frames to detect obstacles visually based on pre-trained models to identify patterns, contours, and colour.

# Obstacle Avoidance and Navigation

Upon detecting an obstacle, the system evaluates the best course of action, such as stopping, slowing down, or manoeuvring around the obstacle, based on its size, distance, and position relative to the vehicle as well as executing relevant actions based on the colour.

## Pseudo Code for Obstacle Avoidance

1. Import Libraries:

a. Import OpenCV for image processing

b. Import PiCamera from picamera2 module for camera interface

c. Import necessary GPIO libraries for motor control (e.g., RPi.GPIO)

d. Import NumPy for image manipulation

2. Initialize Camera:

a. Start PiCamera V2

b. Set camera parameters (e.g., resolution to 640x480 for real-time performance)

c. Initialize motor pins (left\_motor, right\_motor)

3. Function: Process\_Frame(image)

a. Convert image to grayscale

b. Apply GaussianBlur to reduce noise

c. Perform Edge Detection using Canny algorithm

d. Identify Contours from edge-detected image

e. If significant contours found (i.e., contours representing objects in front):

Return True (obstacle detected) and contour position (left, center, right)

f. Else:

Return False (no obstacle)

4. Function: Adjust\_Course(contour\_position)

a. If contour\_position is 'left':

Adjust motors to steer right (slow down right motor, speed up left motor)

b. If contour\_position is 'right':

Adjust motors to steer left (slow down left motor, speed up right motor)

c. If contour\_position is 'center':

Stop the robot or rotate to avoid obstacle

d. If no obstacle:

Move forward with balanced motor speeds

5. Function: Motor\_Control(action)

a. If action = 'stop':

Stop both motors

b. If action = 'move forward':

Set both motors to forward motion

c. If action = 'steer left':

Adjust motor speeds to turn left

d. If action = 'steer right':

Adjust motor speeds to turn right

6. Main Loop:

While True:

a. Capture frame from PiCamera V2

b. Resize frame for faster processing (e.g., 320x240 resolution)

c. Process the frame using Process\_Frame()

d. If Process\_Frame() returns True:

Adjust\_Course(contour\_position) based on where the obstacle is detected

e. Else:

Call Motor\_Control('move forward')

This pseudo code outlines a basic strategy for deciding how to avoid detected obstacles, incorporating safety checks before executing manoeuvres.

The combination of ultrasonic sensors and visual processing through the Raspberry Pi Camera Module V2 provides a robust obstacle detection and management system for a self-driving car. By leveraging both distance measurements and visual cues, the vehicle can make informed decisions to safely navigate through its environment. This strategy emphasizes the importance of sensor fusion and adaptive decision-making in the case of the self-driving car.