Enhanced Robot Navigation System

Introduction

This document details an advanced autonomous navigation system for robots, integrating ultrasound and camera sensors to facilitate obstacle detection and avoidance. The system has been enhanced with several key features to improve its functionality and reliability:

- **Data Logging**: Timestamped data collection for both ultrasound and camera sensors, enabling precise tracking and analysis of sensor readings.
- **Structured Data Storage**: Utilization of the Pandas library for organized and efficient data management, facilitating easy access and analysis.
- **Enhanced Navigation Logic**: Improved algorithms for region detection, allowing the robot to make informed navigation decisions.

System Components

Movement Control

The robot is equipped with functions to control its movement, enabling navigation through various environments. These functions include:

- **Forward Movement**: Propels the robot forward at a specified speed for a set duration.
- **Backward Movement**: Moves the robot backward at a specified speed for a set duration.
- **Left Turn**: Executes a left turn by adjusting motor speeds.
- **Right Turn**: Executes a right turn by adjusting motor speeds.
- **Stop**: Halts all motor activity for a specified period.

The implementation of these functions is as follows:

```
def forward():
    motor(15, 15, 0.3)  # Move forward at speed 15 for 0.3 seconds

def backward():
    motor(-15, -15, 0.3)  # Move backward at speed 15 for 0.3 seconds

def left():
    motor(-20, 20, 0.1)  # Turn left: right motor forward, left motor backward

def right():
    motor(20, -20, 0.1)  # Turn right: left motor forward, right motor backward
```

```
def stop():
    motor(0, 0, 1)  # Stop all motors for 1 second
```

Note: The `motor` function parameters represent the speed of the left motor, the speed of the right motor, and the duration of the action, respectively.

Sensor Functions

The system relies on specific functions to interact with its sensors:

- **Ultrasound Sensor**:
  - `set_distance(True)` : Activates the ultrasound sensor.
  - `set_distance(False)` : Deactivates the ultrasound sensor.
  - `get_distance()` : Retrieves the current distance measurement in centimeters.
- **Camera Sensor**:
  - `set_camera(resolution)` : Activates the camera and sets its resolution (e.g., "8x6").
  - `set_camera()` : Deactivates the camera.
  - `get_camera()` : Captures and returns the current image as a 2D list of RGB tuples.

These functions are critical for managing sensor states and retrieving data for navigation decisions.

Sensor Management

To optimize power consumption and prevent interference, sensors are activated only when needed and deactivated immediately after use. For example:

- **Ultrasound Sensor**:

```
set_distance(True)  # Activate the ultrasound sensor
# Perform operations, e.g., get_distance()
set_distance(False)  # Deactivate the sensor
```

- **Camera Sensor**:

```
set_camera("8x6")  # Activate and set resolution as 8 x 6
# Perform operations, e.g., get_camera()
set_camera()  # Deactivate the sensor
```

This approach ensures efficient resource use and extends the operational lifespan of the robot.

Ultrasound Sensor Implementation

Data Collection

The ultrasound sensor measures distances to potential obstacles, with data collected at regular intervals and logged with timestamps for analysis. The following code demonstrates this process:

```
import pandas as pd
from datetime import datetime
import time

set_distance(True)  # Activate the ultrasound sensor

data = []
for i in range(20):
    distance = get_distance()  # Retrieve distance measurement
    timestamp = datetime.now().strftime('%Y/%m/%d %H:%M:%S')
    data.append([timestamp, distance])
    time.sleep(2)  # Wait for 2 seconds before the next measurement
df = pd.DataFrame(data, columns=["Time", "Distance (cm)"])
print(df)
set_distance(False)  # Deactivate the sensor
```

This script collects 20 distance measurements at 2-second intervals, stores them in a Pandas DataFrame with timestamps, and prints the DataFrame for review.

Navigation Logic

The ultrasound sensor's navigation logic checks for obstacles within a 10 cm threshold, directing the robot to turn right if an obstacle is detected or move forward if the path is clear:

```
def ultrasound_race():
    distance = get_distance()
    if distance < 10:  # Threshold set to 10 cm
        print("Obstacle detected! Turning right.")
        right()
    else:
        print("Path clear. Moving forward.")
        forward()
```

This function ensures the robot responds appropriately to obstacles detected by the ultrasound sensor.

Camera Sensor Implementation

Image Data Collection

The camera sensor captures images of the environment, which are analyzed to inform navigation decisions. Images are collected at regular intervals and logged with timestamps:

```
import pandas as pd
from datetime import datetime
import time

set_camera("8x6")  # Activate camera and set resolution to 8x6
data = []
for i in range(20):
    image = get_camera()  # Capture image
    timestamp = datetime.now().strftime('%Y/%m/%d %H:%M:%S')
    data.append([timestamp, image])
    time.sleep(2)  # Wait for 2 seconds before the next capture
df = pd.DataFrame(data, columns=["Time", "Image Data"])
print(df)
set_camera()  # Deactivate the camera
```

This script captures 20 images at 2-second intervals, stores them in a Pandas DataFrame with timestamps, and prints the DataFrame.

Image data collection implementation

The camera will film an image with a grid of pixels. In a color image, each pixel contains 3 values: red, green and blue. In this task, we are trying to complete camera_race() function to let the robot to make decisions based on camera sensors. Following are the things we may need:

- set_camera("8x6") --> set the mode of camera
- get_camera() --> This function will return a matrix of list indicating the different values for each pixel including three colors **(If the color is dark that means the obstacle is close, the values will be lower --> usually we use 100 as seperation)**
- In other words, we could access the pixel value of row i, column j and in channel k (k = 0 for red, k = 1 for green, 2 for blue) with image[i][j][k] where image = get_camera()
- For example, in the folowing codes, we are asking it to return us the values of 3 channals of the first row with the second column as circled in the figure

```
set_camera("8x6") # set the camera to the correct resolution
for i in range(10):
    image = get_camera() # get the camera image
    print('red', image[0][1][0], 'green', image[0][1][1], 'blue', image[0][1]
```

```
    [2])
    # red 205 green 204 blue 207
```

Navigation Logic

The camera-based navigation logic analyzes specific regions of the captured 8x6 image to detect obstacles by identifying dark areas (potential obstacles). The image is divided into three regions:

- **Left Side Region**: Pixels in rows 2 to 4 and columns 0 to 5, corresponding to the left part of the robot's field of view.
- **Right Side Region**: Pixels in rows 2 to 4 and columns 2 to 7, corresponding to the right part of the robot's field of view.
- **Forward Region**: Pixels in rows 2 to 4 and all columns, representing the area directly ahead.

A pixel is considered "dark" if all its RGB values are less than 100. If more than 30% of the pixels in a region are dark, an obstacle is detected in that region. The navigation logic is implemented as follows:

```
def camera_race():
    image = get_camera()
    # Check region for right: rows 2,3,4; columns 0,1,2,3,4,5
    right_region = [image[r][c] for r in [2,3,4] for c in [0,1,2,3]]
    # Check region for left: rows 2,3,4; columns 2,3,4,5,6,7
    left_region = [image[r][c] for r in [2,3,4] for c in [4,5,6,7]]
    # Check region for backward: rows 2,3,4; columns 0,1,2
    backward_region = [image[r][c] for r in [2,3,4] for c in [2,3,4,5]]

    # Helper to check if at least 30% of pixels in region are "dark" (all
channels < 100)
    def is_most_dark(region, threshold=0.4):
        dark_count = sum(
            sum(channel < 90 for channel in pixel) for pixel in region
        )
        total_channels = len(region) * len(region[0]) if region else 1
        return dark_count / total_channels > threshold
    if is_most_dark(backward_region, threshold=0.4):
        print("No clear path, gobackward")
        backward()
    elif is_most_dark(right_region, threshold=0.3):
        print("Obstacle on left, turn right")
        right()
    elif is_most_dark(left_region, threshold=0.3):
        print("Obstacle on right, turn left")
        left()
    else:
```

```
            print("Path clear, go forward")
            forward()
```

This logic enables the robot to navigate around obstacles based on visual inputs, with decisions based on the presence of obstacles in specific regions.

System Testing

Ultrasound Testing

The ultrasound navigation logic is validated through repeated execution of the `ultrasound_race` function:

```
if __name__ == '__main__':
    set_distance(True)  # Activate ultrasound sensor
    for _ in range(30):
        ultrasound_race()
    set_distance(False)  # Deactivate sensor
```

This test runs the ultrasound-based navigation logic 30 times, ensuring the robot responds correctly to obstacles detected by the ultrasound sensor.

Camera Testing

The camera navigation logic is tested with an extended number of iterations to validate its performance:

```
if __name__ == '__main__':
    set_camera("8x6")  # Activate camera and set resolution
    for _ in range(50):
        camera_race()
    set_camera()  # Deactivate camera
```

This test runs the camera-based navigation logic 50 times, allowing the robot to navigate based on visual inputs from the camera.

Testing Protocol

To ensure the reliability and effectiveness of the navigation system, the following testing protocol is recommended:

1. **Start with Simple Environments**: Begin testing in controlled settings with minimal obstacles to verify basic functionality.
2. **Gradually Increase Complexity**: Introduce more complex environments with varying obstacle density and types to assess adaptability.
3. **Monitor Outputs**: Observe console output for real-time decision-making and review data logs for post-test analysis.
4. **Adjust Thresholds**: Fine-tune thresholds (e.g., 10 cm for ultrasound, 30% darkness for camera) based on real-world performance to optimize navigation accuracy.

This iterative approach supports continuous improvement and adaptation to diverse operational conditions.

Troubleshooting Guide

The following table outlines common issues, their possible causes, and solutions to assist in maintaining system performance:

Issue	Possible Cause	Solution
Sensor not detecting obstacles	Sensor not activated, physical obstruction	Verify sensor activation by calling is_connected()
Inaccurate distance measurements	Sensor miscalibration, environmental interference	Calibrate sensor, eliminate interference sources (e.g., reflective surfaces)
Image analysis issues	Incorrect resolution, poor lighting	Set correct resolution, adjust lighting, modify darkness threshold
Robot not responding to obstacles	Incorrect threshold settings, logic errors	Review and adjust thresholds, debug navigation logic

Conclusion

The enhanced robot navigation system integrates advanced sensor technologies and data management techniques to achieve autonomous movement with effective obstacle avoidance. By leveraging ultrasound and camera sensors, structured data logging, and optimized sensor management, the system provides a robust solution for navigating complex environments. Future enhancements may include machine learning algorithms for advanced obstacle detection and path planning, further improving the system's capabilities.