Project: Particle Filter-Based Robot Localization

November 12, 2023

Abstract

Based on the information from the "Particle Filter" slides, here is a project idea for students involving the implementation of a Particle Filter for localization and navigation using Python. The project is designed to be straightforward enough for students with some programming experience, yet challenging enough to provide a comprehensive understanding of Particle Filters in a practical scenario.

1 Project Description

In this project, students will implement a Particle Filter to estimate the position of a robot moving in a two-dimensional space. The robot's environment will be represented as a grid, where each cell can be either an obstacle or free space. The robot will have access to a simple sensor that provides noisy measurements of its distance to the nearest obstacle in its front, left, right, and back directions.

1.1 Objectives

- **Implement a Particle Filter:** Students will develop a Particle Filter to estimate the robot's location based on sensor readings and a map of the environment.
- **Simulate Robot Movement:** Create a simulation where the robot moves a certain number of steps in the environment, making random turns and moves.
- **Sensor Data Simulation:** Generate simulated sensor data based on the robot's actual position and the map.
- **Visualization:** Implement real-time visualization of the particle cloud and the estimated position of the robot in comparison to its actual position.

1.2 Implementation Approaches

Basic Python Implementation: - Use standard Python libraries ('numpy', 'matplotlib' for visualization). - Represent the map as a 2D array, the robot's position as coordinates, and particles as objects with position and weight attributes. - Implement particle resampling, motion update, and measurement update functions.

Object-Oriented Approach: - Define classes for the Robot, Particle, and Map. - Implement methods for movement, sensing, and updating in each class. - Use inheritance to showcase different types of particles or robots, if desired.

Advanced Visualization with Pygame: - Utilize the 'pygame' library for more interactive and sophisticated visualization. - Allow real-time interaction, e.g., manually controlling the robot's movement or altering the environment.

2 Example Template

Import Necessary Libraries

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
```

Define the Robot and Particle Classes

```python
class Robot:
    def __init__(self, x, y, orientation):
        self.x = x
        self.y = y
        self.orientation = orientation  # in degrees

    def move(self, delta_x, delta_y, delta_orientation):
        self.x += delta_x
        self.y += delta_y
        self.orientation = (self.orientation + delta_orientation) % 360

    # Simulate sensor reading based on robot's position
    def sense(self, environment_map):
        # Implement sensor reading logic here
        pass

class Particle:
    def __init__(self, x, y, orientation, weight):
        self.x = x
        self.y = y
        self.orientation = orientation
        self.weight = weight

    def move(self, delta_x, delta_y, delta_orientation):
        # Add noise to movement
        self.x += delta_x + np.random.normal(0, 0.1)
        self.y += delta_y + np.random.normal(0, 0.1)
        self.orientation = (self.orientation + delta_orientation) % 360 + np.random.normal(0, 5)

    # Update weight based on measurement
    def update_weight(self, measurement, robot_measurement):
        # Implement weight updating logic here
        pass
```

Initialize Robot and Particles

```python
robot = Robot(50, 50, 0)
particles = [Particle(np.random.randint(100), np.random.randint(100), np.random.randint(360), 1.0) for _ in range(1000)]
```

Particle Filter Algorithm

```python
def particle_filter(particles, robot, environment_map, move_command):
    # Move the robot and particles
    robot.move(*move_command)
    for particle in particles:
        particle.move(*move_command)

    # Update particles' weights based on sensor reading
    robot_measurement = robot.sense(environment_map)
    for particle in particles:
        particle_measurement = particle.sense(environment_map)  # Particle's sense method not shown
        particle.update_weight(particle_measurement, robot_measurement)

    # Resampling
    weights = np.array([particle.weight for particle in particles])
    weights /= np.sum(weights)  # Normalize weights
    indices = np.random.choice(range(len(particles)), size=len(particles), p=weights)
    resampled_particles = [particles[i] for i in indices]

    return resampled_particles
```

Visualization using Matplotlib

```python
def update(frame_number):
    global particles, robot
    move_command = (1, 0, 10)  # Example move command
    particles = particle_filter(particles, robot, environment_map, move_command)

```

```python
    # Clear current plot
    plt.cla()

    # Plot particles
    xs, ys = zip(*[(particle.x, particle.y) for particle in particles])
    plt.scatter(xs, ys, color='blue', s=1)

    # Plot robot
    plt.scatter(robot.x, robot.y, color='red', s=10)

    plt.xlim(0, 100)
    plt.ylim(0, 100)
    plt.title("Particle Filter Robot Localization")

fig = plt.figure()
ani = FuncAnimation(fig, update, frames=10, interval=1000)
plt.show()
```

Note:

- This code provides a basic framework and requires further development to fully simulate the environment, sensor readings, and particle weight updates.
- The move and sense methods for the Robot and Particle classes should be tailored to the specific problem and sensor model.
- The visualization updates the particles and robot position at each step, illustrating the working of the particle filter.

This implementation serves as a foundational guideline, and students are encouraged to build upon it, refining and adding complexity as needed for their specific project requirements.

3 Expected Outcomes

- - Understand the concept and application of Particle Filters in localization.
- - Gain experience in simulating robot movement and sensor readings.
- - Develop skills in probabilistic reasoning and algorithm implementation.

4 Evaluation Criteria

- - Accuracy of the localization (how close the estimated position is to the actual position).
- - Efficiency of the implementation (number of particles used vs. accuracy).
- - Quality of the visualization and ease of understanding the Particle Filter process.

This project provides a balance of theoretical understanding and practical application, making it an excellent exercise for students to grasp the fundamentals of Particle Filters in robotics.