Sistem Fuzzy Obstacle Avoidance PioneerP3DX

Figo Arzaki Maulana

28 April 2025

1 Deskripsi Sistem

Video Demo

Sistem ini menggunakan 6 sensor ultrasonik pada robot PioneerP3DX untuk program obstacle avoidance. Data jarak dari setiap sensor difuzzifikasi menjadi dua fuzzy membership: Near dan Far. Hasil fuzzy inference menghasilkan sinyal PWM untuk motor kiri dan kanan. Pada sistem ini digunakan 2 input dan 1 output untuk setiap roda kanan dan kiri. Sistem menggunakan singleton PWM output dengan nilai -5 dan 5 namun pada velocity dilakukan normalisasi kecepatan menjadi -1 hingga 1. Hal ini dilakukan melalui tuning dan memberikan hasil yang lebih bagus daripada menggunakan singleton PWM dengan nilai -1 dan 1.

2 Membership Function

Proses fuzzifikasi menggunakan fungsi segitiga. Implementasi fungsi keanggotaan untuk "Near" dan "Far" terhadap jarak sensor adalah sebagai berikut:

```python
def triangle_membership(x, a, b, c, FD0, FD2):
    if x < a:
        FD = FD0
    elif x >= a and x < b:
        FD = (x - a) / (b - a)
    elif x >= b and x < c:
        FD = (c - x) / (c - b)
    elif x >= c:
        FD = FD2
    return FD

def ultrasound_membership(x):
    near = triangle_membership(x, 0.2, 0.2, 0.3, 1, 0)
    far = triangle_membership(x, 0.2, 0.3, 0.3, 0, 1)
    y = np.array([[near], [far]], dtype=float)
    return y
```

Listing 1: Implementasi Fungsi Keanggotaan Segitiga

Fungsi keanggotaan menggunakan parameter-parameter berikut:

- **Near**: Segitiga dengan a=0.2, b=0.2, c=0.3, FD0=1, FD2=0
- **Far**: Segitiga dengan a=0.2, b=0.3, c=0.3, FD0=0, FD2=1

3 Rule Base dalam Tabel

Basis aturan fuzzy mendefinisikan bagaimana sistem merespons input dari sensor. Terdapat tabel aturan terpisah untuk motor kiri dan kanan:

3.1 Aturan Motor Kiri

Sensor Kanan	Sensor Kiri	
	Near	Far
Near	IF Near/Near → -5	IF Far/Near → -5
Far	IF Near/Far → +5	IF Far/Far → +5

Table 1: Tabel Aturan Fuzzy untuk Motor Kiri

3.2 Aturan Motor Kanan

Sensor Kanan	Sensor Kiri	
	Near	Far
Near	IF Near/Near → -5	IF Far/Near → +5
Far	IF Near/Far → -5	IF Far/Far → +5

Table 2: Tabel Aturan Fuzzy untuk Motor Kanan

Dalam kode, aturan-aturan ini diimplementasikan sebagai matriks:

```
rule_table_left = np.array([[0, 0], [1, 1]], dtype=int)
rule_table_right = np.array([[0, 1], [0, 1]], dtype=int)
singleton_PWM_outputs = np.array([[-5], [5]], dtype=float)
```

Listing 2: Implementasi Tabel Aturan

Di mana indeks 0 sesuai dengan nilai -5 dan indeks 1 sesuai dengan nilai +5 dalam nilai output singleton.

4 Defuzzifikasi dan Plot Output Crisp

Setelah inferensi, output didefuzzifikasi dengan metode weighted average. Implementasi melibatkan perhitungan output crisp berdasarkan evaluasi aturan:

```
defuzz_table = []
for idx in range(3):
    left_idx = idx
    right_idx = len(output_singleton) - idx - 1

    left_memberships = output_singleton[left_idx]
    right_memberships = output_singleton[right_idx]

    num_left = 0
    den_left = 0
    num_right = 0
    den_right = 0

    for r in range(2):  # near/far
        for l in range(2):  # near/far
            tab_idx_left = rule_table_left[r][l]
            tab_idx_right = rule_table_right[r][l]

            fd1andfd2 = float(min(left_memberships[l], right_memberships[r]))

            num_left += fd1andfd2 * singleton_PWM_outputs[tab_idx_left]
            den_left += fd1andfd2
            num_right += fd1andfd2 * singleton_PWM_outputs[tab_idx_right]
            den_right += fd1andfd2

```

```
    crisp_left = num_left / den_left if den_left > 0 else 0
    crisp_right = num_right / den_right if den_right > 0 else 0
    crisp_out.append([crisp_left, crisp_right])
```

Listing 3: Implementasi Defuzzifikasi

5 Konfigurasi Sensor dan Pembebanan

Robot menggunakan 6 sensor ultrasonik dengan indeks sensor [1,2,3,4,5,6]. Dari keenam sensor tersebut, data diambil secara berpasangan mulai dari sensor terluar hingga sensor terdalam:

- Sensor pasang 1 (indeks 1 & 6) dengan bobot 1.5
- Sensor pasang 2 (indeks 2 & 5) dengan bobot 2.5
- Sensor pasang 3 (indeks 3 & 4) dengan bobt 3.5

Bobot ini memengaruhi perhitungan weighted crisp sebelum normalisasi ke rentang [-1, 1]. Nilai tersebut ditentukan berdasarkan urgensi deteksi obstacle:

- Sensor yang berada di samping ketika mendeteksi obstacle berarti akan membuat robot bersenggolan atau hanya terserempet sehingga urgensinya tidak terlalu besar.
- Pasangan sensor yang ada di tengah memiliki urgensi yang sangat besar karena robot pasti akan bertabrakan jika lurus.

```
weights = [1.5, 2.5, 3.5]
weighted_crisp = [0, 0]
for i in range(len(crisp_out)):
    weighted_crisp[0] += crisp_out[i][0] * weights[i]
    weighted_crisp[1] += crisp_out[i][1] * weights[i]
weighted_crisp[0] /= sum(weights)
weighted_crisp[1] /= sum(weights)

max_velocity = 4
weighted_crisp[0] = max(-1, min(1, weighted_crisp[0] / max_velocity))
weighted_crisp[1] = max(-1, min(1, weighted_crisp[1] / max_velocity))
```

Listing 4: Perhitungan Weighted Crisp Output

6 Visualisasi

Sistem ini menyertakan visualisasi real-time dari fungsi keanggotaan dan pembacaan sensor:

```
plt.ion()
fig, ax = plt.subplots(figsize=(10, 6))
plt.title('Distance Membership Functions')
plt.xlabel('Distance (m)')
plt.ylabel('Membership Value')
plt.grid(True)
plt.xlim([0, 1])
plt.ylim([0, 1.1])

ax.plot(dis_eval, near_vals, 'b-', label='Near')
ax.plot(dis_eval, far_vals, 'g-', label='Far')
ax.fill_between(dis_eval, near_vals, alpha=0.2, color='blue')
ax.fill_between(dis_eval, far_vals, alpha=0.2, color='green')
plt.legend()
```

Listing 5: Pengaturan Visualisasi

A Kode Lengkap

```python
import numpy as np
import matplotlib.pyplot as plt
from coppeliasim_zmqremoteapi_client import RemoteAPIClient

def triangle_membership(x, a, b, c, FD0, FD2):
    if x < a:
        FD = FD0
    elif x >= a and x < b:
        FD = (x - a) / (b - a)
    elif x >= b and x < c:
        FD = (c - x) / (c - b)
    elif x >= c:
        FD = FD2
    return FD

def ultrasound_membership(x):
    near = triangle_membership(x, 0.2, 0.2, 0.3, 1, 0)
    far = triangle_membership(x, 0.2, 0.3, 0.3, 0, 1)
    y = np.array([[near], [far]], dtype=float)
    return y

def getSensorsHandle(sim):
    sensorsHandle = []
    for i in range(16):
        sensorHandle = sim.getObject('/PioneerP3DX/ultrasonicSensor[' + str(i) +
         ']')
        sensorsHandle.append(sensorHandle)
    _, _, _, _, _ = sim.handleProximitySensor(sim.handle_all)
    return sensorsHandle

def getDistances(sim, sensorsHandle):
    Distances = []
    for i in range(16):
        detectionState, _, detectedPoint, _, _ = sim.readProximitySensor(
    sensorsHandle[i])
        distanceValue = detectedPoint[2]
        if detectionState == False:
            distanceValue = 1.0
        Distances.append(distanceValue)
    return Distances

def getMotorsHandle(sim):
    motorRightHandle = sim.getObject('/PioneerP3DX/rightMotor')
    motorLeftHandle = sim.getObject('/PioneerP3DX/leftMotor')
    return motorLeftHandle, motorRightHandle

def setRobotMotion(sim, motorsHandle, veloCmd):
    _ = sim.setJointTargetVelocity(motorsHandle[0], veloCmd[0])
    _ = sim.setJointTargetVelocity(motorsHandle[1], veloCmd[1])

dis_eval = np.linspace(0, 1, 100)
near_vals = []
far_vals = []
for dis in dis_eval:
    membership = ultrasound_membership(dis)
    near_vals.append(membership[0][0])
    far_vals.append(membership[1][0])

plt.ion()
fig, ax = plt.subplots(figsize=(10, 6))
plt.title('Distance Membership Functions')
```

```python
plt.xlabel('Distance (m)')
plt.ylabel('Membership Value')
plt.grid(True)
plt.xlim([0, 1])
plt.ylim([0, 1.1])

ax.plot(dis_eval, near_vals, 'b-', label='Near')
ax.plot(dis_eval, far_vals, 'g-', label='Far')
ax.fill_between(dis_eval, near_vals, alpha=0.2, color='blue')
ax.fill_between(dis_eval, far_vals, alpha=0.2, color='green')
plt.legend()

dot0_near, = ax.plot([], [], 'ro', markersize=8, label='S0')
dot0_far, = ax.plot([], [], 'ro', markersize=8, fillstyle='none', label='S0')
dot2_near, = ax.plot([], [], 'go', markersize=8, label='S2')
dot2_far, = ax.plot([], [], 'go', markersize=8, fillstyle='none', label='S2')
dot5_near, = ax.plot([], [], 'bo', markersize=8, label='S5')
dot5_far, = ax.plot([], [], 'bo', markersize=8, fillstyle='none', label='S5')
dot7_near, = ax.plot([], [], 'mo', markersize=8, label='S7')
dot7_far, = ax.plot([], [], 'mo', markersize=8, fillstyle='none', label='S7')

ax.legend(loc='upper right')

text0 = ax.text(0.05, 0.95, '', transform=ax.transAxes)
text2 = ax.text(0.05, 0.90, '', transform=ax.transAxes)
text5 = ax.text(0.05, 0.85, '', transform=ax.transAxes)
text7 = ax.text(0.05, 0.80, '', transform=ax.transAxes)
text_wheels = ax.text(0.05, 0.75, '', transform=ax.transAxes)

plt.draw()
plt.pause(0.01)
print("Program Started")

client = RemoteAPIClient()
sim = client.require("sim")
sim.setStepping(False)
sim.startSimulation()

sensors_handle = getSensorsHandle(sim)
motors_handle = getMotorsHandle(sim)
wheels_velo = [0.0, 0.0]
singleton_PWM_outputs = np.array([[-5], [5]], dtype=float)
weights = [0.5, 1.0, 1.5]
sensor_being_checked = [1, 2, 3, 4, 5, 6]
rule_table_left = np.array([[0, 0], [1, 1]], dtype=int)
rule_table_right = np.array([[0, 1], [0, 1]], dtype=int)
lt = ["near", "far"]
stop_after = 30000
time_start = sim.getSimulationTime()

while True:
    t_now = sim.getSimulationTime() - time_start
    obj_distance = getDistances(sim, sensors_handle)
    d0, d2, d5, d7 = obj_distance[0], obj_distance[2], obj_distance[5],
    obj_distance[7]
    d0_clip = min(d0, 1.0)
    d2_clip = min(d2, 1.0)
    d5_clip = min(d5, 1.0)
    d7_clip = min(d7, 1.0)

    s0_membership = ultrasound_membership(d0_clip)
    s0_near, s0_far = s0_membership[0][0], s0_membership[1][0]
    s2_membership = ultrasound_membership(d2_clip)
```

```python
    s2_near, s2_far = s2_membership[0][0], s2_membership[1][0]
    s5_membership = ultrasound_membership(d5_clip)
    s5_near, s5_far = s5_membership[0][0], s5_membership[1][0]
    s7_membership = ultrasound_membership(d7_clip)
    s7_near, s7_far = s7_membership[0][0], s7_membership[1][0]

    dot0_near.set_data([d0_clip], [s0_near])
    dot0_far.set_data([d0_clip], [s0_far])
    dot2_near.set_data([d2_clip], [s2_near])
    dot2_far.set_data([d2_clip], [s2_far])
    dot5_near.set_data([d5_clip], [s5_near])
    dot5_far.set_data([d5_clip], [s5_far])
    dot7_near.set_data([d7_clip], [s7_near])
    dot7_far.set_data([d7_clip], [s7_far])

    text0.set_text(f'S0: {d0:.2f}m (N: {s0_near:.2f}, F: {s0_far:.2f})')
    text2.set_text(f'S2: {d2:.2f}m (N: {s2_near:.2f}, F: {s2_far:.2f})')
    text5.set_text(f'S5: {d5:.2f}m (N: {s5_near:.2f}, F: {s5_far:.2f})')
    text7.set_text(f'S7: {d7:.2f}m (N: {s7_near:.2f}, F: {s7_far:.2f})')

    output_singleton = []
    crisp_out = []

    for sensor_idx, sensor in enumerate(sensor_being_checked):
        distance = obj_distance[sensor]
        distance = max(0.0, min(1.0, distance))
        output = ultrasound_membership(distance)
        output_singleton.append([output[0][0], output[1][0]])

    defuzz_table = []
    for idx in range(3):
        left_idx = idx
        right_idx = len(output_singleton) - idx - 1

        left_memberships = output_singleton[left_idx]
        right_memberships = output_singleton[right_idx]

        num_left = 0
        den_left = 0
        num_right = 0
        den_right = 0

        for r in range(2):  # near/far
            for l in range(2):  # near/far
                tab_idx_left = rule_table_left[r][l]
                tab_idx_right = rule_table_right[r][l]

                fd1andfd2 = float(min(left_memberships[l], right_memberships[r]))

                num_left += fd1andfd2 * singleton_PWM_outputs[tab_idx_left][0]
                den_left += fd1andfd2
                num_right += fd1andfd2 * singleton_PWM_outputs[tab_idx_right][0]
                den_right += fd1andfd2

        crisp_left = num_left / den_left if den_left > 0 else 0
        crisp_right = num_right / den_right if den_right > 0 else 0
        crisp_out.append([crisp_left, crisp_right])

    weighted_crisp = [0, 0]
    for i in range(len(crisp_out)):
        weighted_crisp[0] += crisp_out[i][0] * weights[i]
        weighted_crisp[1] += crisp_out[i][1] * weights[i]
```

```
    weighted_crisp[0] /= sum(weights)
    weighted_crisp[1] /= sum(weights)

    max_velocity = 2
    weighted_crisp[0] = max(-1, min(1, weighted_crisp[0] / max_velocity))
    weighted_crisp[1] = max(-1, min(1, weighted_crisp[1] / max_velocity))

    wheels_velo = weighted_crisp[0], weighted_crisp[1]
    text_wheels.set_text(f'Wheels: L={wheels_velo[0]:.2f}, R={wheels_velo[1]:.2f}')

    setRobotMotion(sim, motors_handle, wheels_velo)
    fig.canvas.draw_idle()
    plt.pause(0.01)

    if sim.getSimulationTime() - time_start > stop_after:
        break

sim.stopSimulation()
print("\nProgram Ended\n")
```

Listing 6: Implementasi Lengkap Fuzzy Obstacle Avoidance