Laporan Sistem Fuzzy Target Following pada PioneerP3DX

Hernanda Achmad Priyatna - 5022221114

May 4, 2025

1 Gambaran Umum Sistem

1.1 Obstacle Avoidance

Pada *obstacle avoidance*, digunakan 6 buah sensor ultrasonik dengan dua himpunan fuzzy: *Near* dan *Far*, masing–masing dengan ambang (threshold) di 0.3 m (lihat Gambar 1).

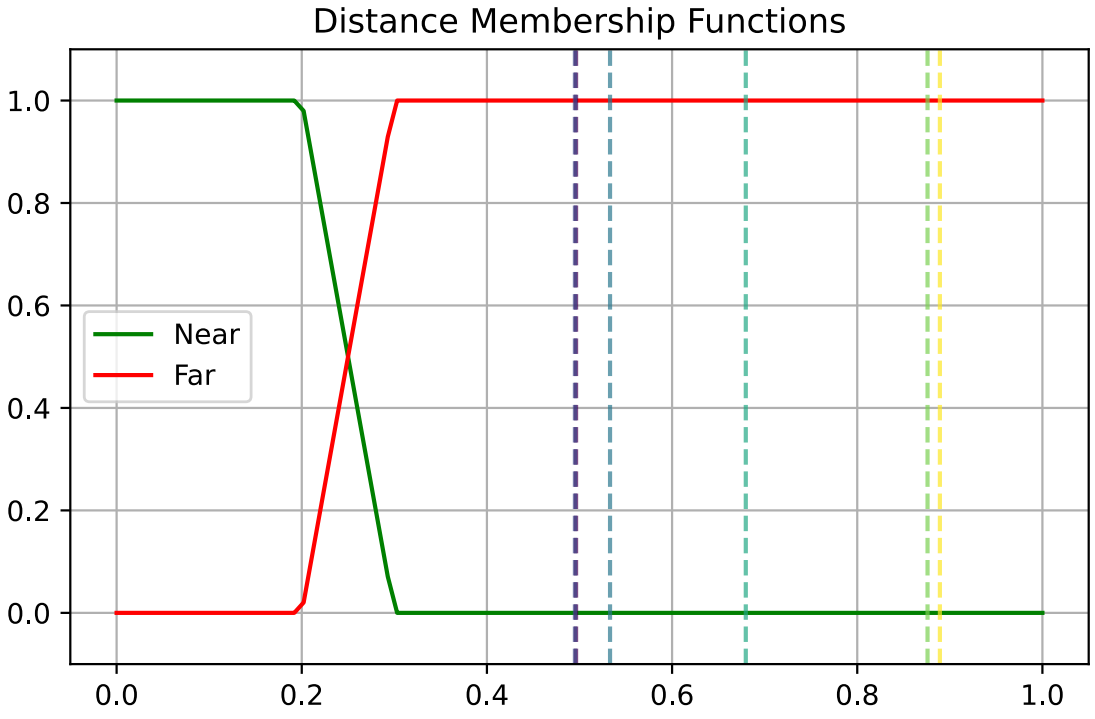

Figure 1: Fungsi Keanggotaan Sensor Ultrasonik (*Near* & *Far*).

Sebagai keluaran, sistem menghasilkan kecepatan linier dan kecepatan sudut (angular). Nilai–nilai *singleton* yang digunakan adalah:

$$\text{singleton\_linear} = \{-0.5,\ 0,\ +0.5\}, \quad \text{singleton\_angular} = \{-0.5,\ 0,\ +0.5\}.$$

Dalam kode Python, dinyatakan sebagai:

```
singleton_outputs_linear  =
  np.array([[-0.5], [0], [0.5]], dtype=float)
singleton_outputs_angular =
  np.array([[-0.5], [0], [0.5]], dtype=float)
```

Tabel–tabel aturan fuzzy untuk *obstacle avoidance* kemudian menjadi:

Table 1: Tabel Aturan Fuzzy untuk Kecepatan Linier

	Left Sensor Near	**Left Sensor Far**
Right Sensor Near	IF Near/Near $\rightarrow -0.5$	IF Far/Near $\rightarrow 0$
Right Sensor Far	IF Near/Far $\rightarrow 0$	IF Far/Far $\rightarrow +0.5$

Table 2: Tabel Aturan Fuzzy untuk Kecepatan Sudut (Angular)

	Left Sensor Near	**Left Sensor Far**
Right Sensor Near	IF Near/Near $\rightarrow 0$	IF Far/Near $\rightarrow -0.5$
Right Sensor Far	IF Near/Far $\rightarrow +0.5$	IF Far/Far $\rightarrow 0$

Persamaan dalam array Python:

```
rule_table_linear  = np.array([[0, 1],
                               [1, 2]], dtype=int)
rule_table_angular = np.array([[1, 0],
                               [2, 1]], dtype=int)
```

di mana indeks 0,1,2 merujuk ke posisi dalam `singleton_outputs_*`.

1.2 Target Following

Pada bagian ini, kendali *target following* dilakukan dengan membagi komponen kendali menjadi dua, yaitu **error linear** dan **error angular**. Error linear dihitung sebagai jarak antara posisi robot dan posisi target menggunakan rumus Pythagoras, sedangkan error angular dihitung dari selisih antara arah menuju target dengan orientasi robot saat ini. Untuk melakukan proses fuzzifikasi, kedua error tersebut dibagi menjadi beberapa fungsi keanggotaan seperti ditunjukkan pada Gambar 2 dan Gambar 3.

Pada *target following*, sistem saya dibagi menjadi dua komponen:

- **Linear error** (e_d): jarak antara posisi robot dan target, dihitung dengan rumus Pythagoras
$$e_d = \sqrt{(x_{\text{ref}} - x_{\text{act}})^2 + (y_{\text{ref}} - y_{\text{act}})^2}.$$
- **Angular error** (e_θ): selisih sudut arah target dan orientasi robot,
$$\theta_{\text{target}} = \text{atan2}(y_{\text{ref}} - y_{\text{act}},\, x_{\text{ref}} - x_{\text{act}}),$$
$$e_\theta = \text{wrap}\big(\theta_{\text{target}} - \theta_{\text{robot}}\big),$$
dengan safety untuk memastikan nilai error dalam rentang $e_\theta \in [-\pi, \pi]$ (normalisasi theta).

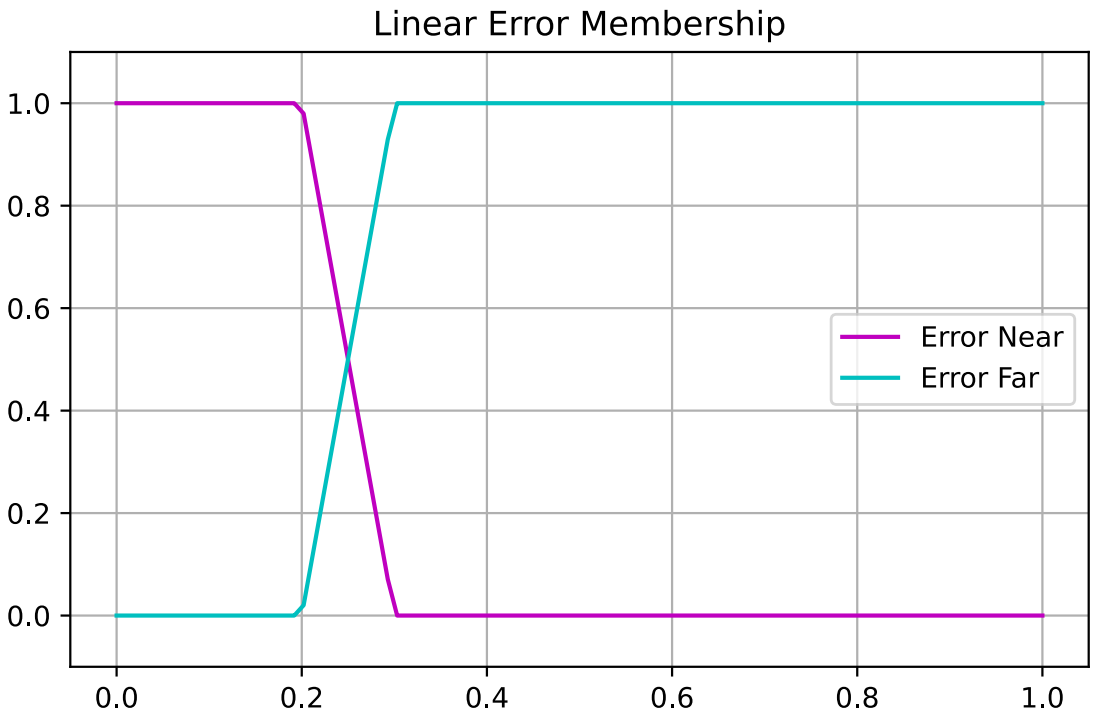

Figure 2: Fungsi Keanggotaan Error Linear

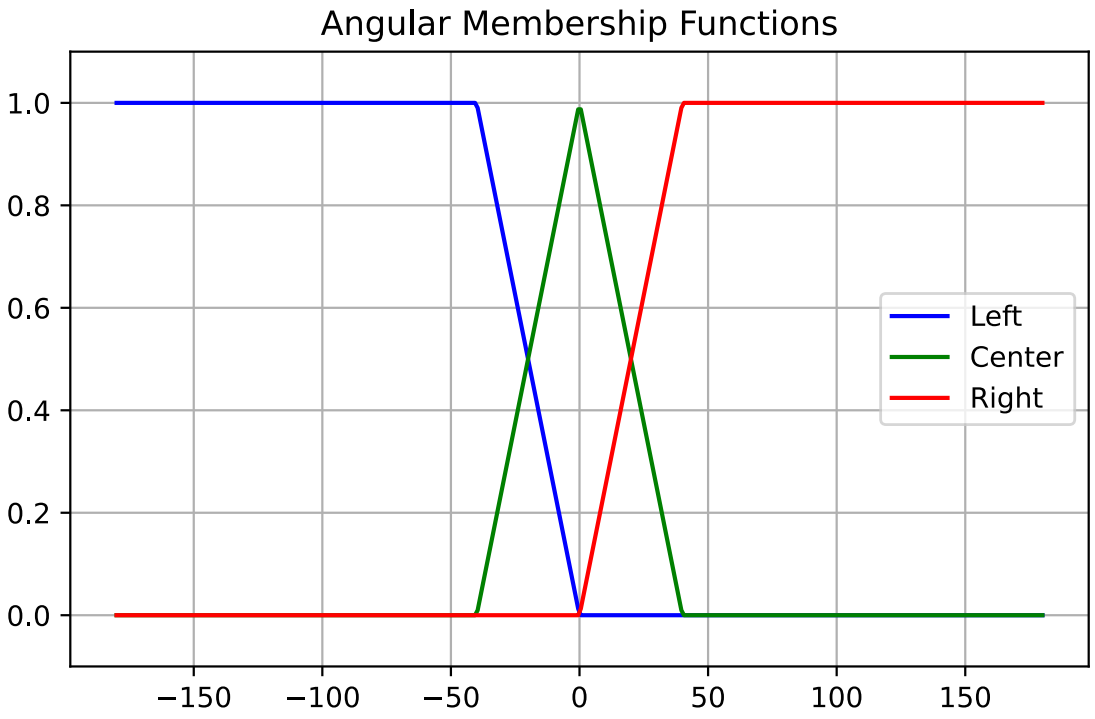

Figure 3: Fungsi Keanggotaan Error Angular

Kode Python untuk memperoleh e_d dan e_θ:

```
position, _ = getTargetPose(disk_handle)
x_ref, y_ref, _ = position

position, theta_robot = getPose()
x_act, y_act, _ = position

theta_target = np.arctan2(y_ref - y_act, x_ref - x_act) * 180 / np.pi
theta_robot = theta_robot * 180 / np.pi

current_pose = (position[0], position[1])

while theta_target > 180:
    theta_target -= 360
while theta_target < -180:
    theta_target += 360

while theta_robot > 180:
    theta_robot -= 360
while theta_robot < -180:
    theta_robot += 360

error_linear = np.sqrt((x_ref - x_act) ** 2 + (y_ref - y_act) ** 2)
error_theta = theta_target - theta_robot
```

Semua error ini kemudian difuzzifikasi ke dalam himpunan:

$$\text{Near, Far} \quad \text{(linear)}, \qquad \text{Left, Center, Right} \quad \text{(angular)}.$$

Singleton dan rule table untuk *target following* adalah:

```
singleton_error_linear  = np.array([[0.0], [0.5]],       dtype=float)
singleton_error_angular = np.array([[+0.5], [0.0], [-0.5]], dtype=float)

rule_table_error_linear  = np.array([[0, 0, 0],
                                     [1, 1, 1]], dtype=int)
rule_table_error_angular = np.array([[0, 1, 2],
                                     [0, 1, 2]], dtype=int)
```

Table 3: Tabel Aturan Fuzzy untuk Output Linear (Target Following)

	Left	**Center**	**Right**
Near	IF Near/Left → 0.0	IF Near/Center → 0.0	IF Near/Right → 0.0
Far	IF Far/Left → 0.5	IF Far/Center → 0.5	IF Far/Right → 0.5

Table 4: Tabel Aturan Fuzzy untuk Output Sudut (Target Following)

	Left	**Center**	**Right**
Near	IF Near/Left $\rightarrow +0.5$	IF Near/Center $\rightarrow 0.0$	IF Near/Right $\rightarrow -0.5$
Far	IF Far/Left $\rightarrow +0.5$	IF Far/Center $\rightarrow 0.0$	IF Far/Right $\rightarrow -0.5$

2 Defuzzifikasi dan Plot Crisp

Setelah proses fuzzifikasi dan pembentukan *rule table*, tahap selanjutnya adalah melakukan defuzzifikasi untuk memperoleh nilai keluaran berupa *crisp output*, yaitu nilai kecepatan linear dan kecepatan angular yang akan digunakan oleh robot.
Pada sistem ini, metode defuzzifikasi yang digunakan adalah Weighted Average yang sesuai dengan materi untuk mendapatkan nilai kontinyu.
Untuk setiap kombinasi kondisi input (contohnya: *Near* dan *Far*), sistem mengacu pada tabel aturan fuzzy yang telah dibuat sebelumnya untuk menentukan output yang sesuai.
Nilai crisp kemudian dihitung berdasarkan rumus:

$$\text{Crisp Output} = \frac{\sum_{i=1}^{n} \mu_i \cdot z_i}{\sum_{i=1}^{n} \mu_i} \tag{1}$$

dengan:

- μ_i = derajat keanggotaan (membership value) dari aturan ke-i
- z_i = nilai singleton atau nilai output dari aturan ke-i
- n = jumlah aturan yang aktif

Operasi logika AND digunakan untuk menggabungkan dua kondisi fuzzy pada obstacle avoidance dan target following.
Setelah proses defuzzifikasi, diperoleh nilai *crisp output* untuk kecepatan linear dan angular baik dari komponen *obstacle avoidance* maupun *target following*. Selanjutnya, sistem menggunakan mekanisme **decision making** untuk menentukan output akhir yang akan digunakan oleh robot.
Keputusan ini ditentukan berdasarkan pengecekan semua sensor ultrasonik untuk mendeteksi keberadaan rintangan di sekitar robot. Jika salah satu dari sensor yang dicek masih memiliki derajat keanggotaan *Near* lebih dari 0.3, maka variabel `is_there_near` akan diset ke `True`, yang berarti rintangan masih terdeteksi dalam jarak dekat.
Proses pengecekan ini dilakukan melalui potongan kode berikut:

```
for sensor_idx, sensor in enumerate(sensor_being_checked):
    distance = obj_distance[sensor]
    distance = max(0.0, min(1.0, distance))
    output = dis_MF(distance)

    if output[0][0] > 0.3:
        is_there_near = True
```

Berdasarkan nilai `is_there_near`, maka sistem akan memilih kecepatan output robot dengan pembobotan penuh. Jika nilai `True`, maka robot mengutamakan hasil dari *obstacle avoidance*, dan sebaliknya jika `False`, maka robot mengutamakan hasil dari *target following*. Pemilihan ini ditunjukkan melalui kode berikut:

```
if is_there_near:
    print("Obstacle Avoidance", weighted_crisp[0], weighted_crisp[1])
    output_velocity = [weighted_crisp[0] / 5, weighted_crisp[1] / 5]
else:
    print("Target Following", crisp_out[0][0], crisp_out[0][1])
    output_velocity = [crisp_out[0][0] / 5, crisp_out[0][1] / 5]
```

Pembagian nilai output velocity dengan 5 agar robot tidak bergerak terlalu cepat mengingat tempat simulasi yang kecil dan terbatas.

2.1 Inverse Kinematic

Setelah menentukan nilai kecepatan linear dan angular hasil dari proses *decision making*, tahap selanjutnya adalah mengonversi kecepatan tersebut menjadi kecepatan roda kiri dan kanan menggunakan metode **inverse kinematic**. Proses ini diperlukan agar perintah kecepatan dapat diterapkan langsung pada aktuator robot diferensial Pioneer P3DX. Implementasi inverse kinematic yang digunakan ditunjukkan pada potongan kode berikut:

```
def inverseKinematic(v, w):
    R = 97.5e-3  # Jari-jari roda (meter)
    L = 381e-3   # Jarak antara roda kiri dan kanan (meter)
    T = np.array([[R/2, R/2],
                  [R/(2*L), -R/(2*L)]])
    T_inv = np.linalg.inv(T)
    return np.dot(T_inv, np.array([v, w]))
```

Fungsi ini akan menghasilkan dua buah nilai, yaitu kecepatan roda kiri dan roda kanan, yang kemudian dikirim ke aktuator untuk menggerakkan robot sesuai hasil kontrol.

2.2 Perekaman Pergerakan Robot

Gambar berikut menunjukkan lintasan atau *path* yang dihasilkan oleh pergerakan robot saat menjalankan kombinasi antara **target following** dan **obstacle avoidance**. Dari visualisasi ini, dapat diamati bagaimana robot secara dinamis menyesuaikan arah dan kecepatan untuk menghindari rintangan sekaligus menuju target dengan tetap mempertahankan kestabilan gerak.

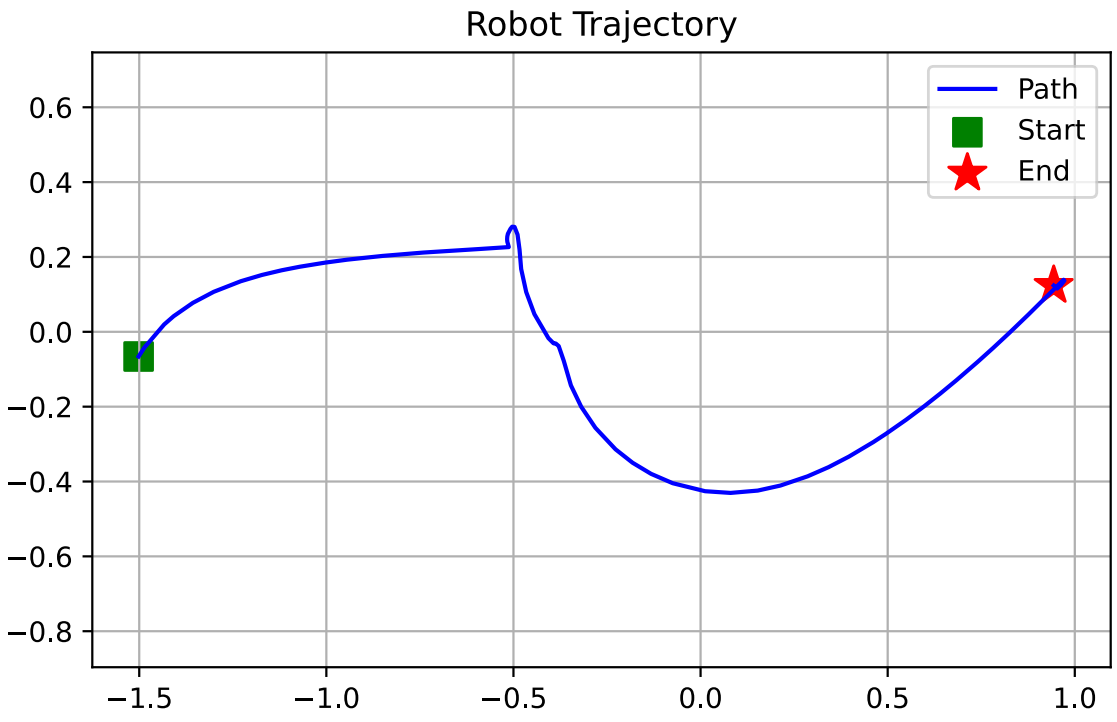

Figure 4: Record Lokasi Robot

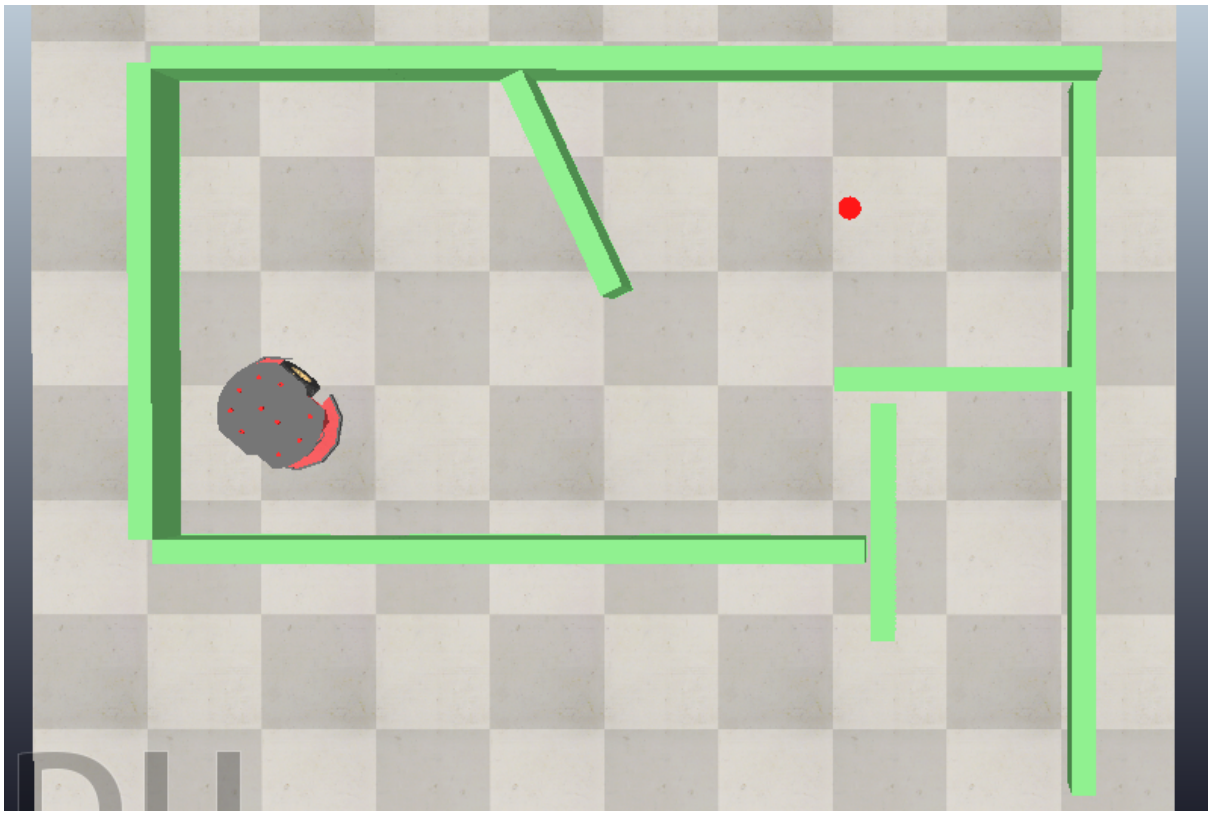

Figure 5: Robot pada Simulasi dan Target Lokasi