POLITECHNIKA RZESZOWSKA im. IGNACEGO ŁUKASIEWICZA

WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI POLITECHNIKI RZESZOWSKIEJ

28.03.2023

Sztuczna Inteligencja

Sprawozdanie nr.2

Temat: STEROWANIE ROBOTEM MOBILNYM KHEPERA PRZY POMOCY LOGIKI ROZMYTEJ- ZADANIE OMIJANIA PRZESZKÓD

Prowadzący:

dr hab. inż. Roman Zajdel, prof. PRz

Wykonali:

2EF-DI, L8

Daniel Kleczyński

Spis treści

1. Wstęp teoretyczny

Celem tego ćwiczenia jest zapoznanie się ze sposobem syntezy systemu ekspertowego, który pełni rolę regulatora. W tym konkretnym przypadku, będzie to rozmyty system ekspertowy wykorzystywany do sterowania robotem mobilnym Khepera, który ma za zadanie omijać przeszkody.

Laboratorium składa się z trzech części. W części I uczestnicy zapoznają się z budową robota Khepera III oraz sposobem jego sterowania za pomocą dedykowanych funkcji języka Python. W części II uczestnicy projektują rozmyty system ekspertowy typu Mamdaniego z wykorzystaniem biblioteki skfuzzy, która będzie realizowaał zadanie omijania przeszkód. Część III laboratorium polega na praktycznej weryfikacji poprawności zrealizowanego systemu w sterowaniu rzeczywistym robotem.

Dodatkowo, w trakcie laboratorium zdecydowano się na zmianę rodzaju systemu ekspertowego z rozmytego typu Mamdaniego na rozmyty system ekspertowy typu Takagi-Sugeno zerowego poziomu.

2. Środowisko programistyczne

Aby móc zacząć tworzyć system ekspertowy, konieczne będzie zainstalowanie pakietu Scikit-fuzzy (skfuzzy), simpful odpowiedzialnego za skonstruowanie systemu Takagi-Sugeno a także przydatnych pakietów numpy i matplotlib, które umożliwią wykonanie kodu z instrukcji do laboratorium. Poniższe zadania zostały wykonane w edytorze kodu Visual Studio Code z zainstalowaną wtyczką Jupyter aby muc korzystać z notatnika. Python został zainstaloway w wersji 3.11.2

3. Przebieg ćwiczenia

Aby móc stworzyć system ekspertowy należało zapoznać się z instrukcją użytkownika robota Khepler III aby poznać jego możliwości.

Należy również zapoznać się z dołączonymi skryptami w języku pytchon dołączonych do instrukcji laboratoryjnej odpowiadają one za stworzenie komunikacji z robotem oraz wysłaniu mu odpowiednio zmienionych informacji z naszego systemu ekspertowego.

Przydatne funkcje z którymi warto się zapoznać aby dobrze wykonać ćwiczenia:

- k3ReadProximitSensors -Odczyt wartości sygnałów czujników zbliżeniowych oraz przeskalowanie czujników 2, 5 oraz 9
- k3ReadAmbientSensors -Odczyt wartości sygnałów czujników pracujących w trybie pomiaru światła zewnętrznego. Polecenie użyteczne przy realizacji zadania podążania do źródła światła
- k3SetSpeed Ustawienie prędkości poszczególnych kół
- k3Stop- Zatrzymanie robota
- k3ReadProximitySensorsLoop -Odczyt sygnałów z czujników zbliżeniowych przez ustaloną liczbę cykli(domyślnie 1000). Stosowane do sporządzenia charakterystyki czujników.

3.1. Część I. Zapoznanie z Robotem Khepera III

Prowadzący zajęcia zademonstrował jak obsługiwać robota kilkoma poleceniami po tym jak został uruchomiony plik hepera3.py gdzie zostało nawiązane połączenie a poleceniem s=k3Init() został utworzony uchwyt do konkretnego robota aby muc móc wprowadzać resztę poleceń. 33

- Dalej zostało sprawdzony stan baterii
- pomiary natężenia światła
- pomiary odległości
- sprawdzenie prędkości kół,
- została ustawiona pewna prędkość kół i robot poruszał się do przodu skręcał lub obracał się w około własnej osi
- Została ustawiano pozycja [x,y] do której robot podążał
- Odczytanie obecnego położenia
- Oraz przerywanie wykonywanego programu

Zostało również przedstawione środowisko po którym robot będzie się poruszać jest to duży kwadrat dookoła którego jest biała ściana i białe oraz czarne przeszkody które można przestawiać (kolor jest istotny na odczyty z czujników).

Przeprowadzono również test odczytów czujników który sprawdzał pomiary w zależności od koloru przeszkody.

Wynik testu był zależy od koloru który dobrze pochłaniał (nie odbijał) promienie z zakresu podczerwieni. Jeśli kolor dobrze absorbuje podczerwień np. kolor czarny odczyt będzie zaniżony co może prowadzić do nie wykrycia przeszkody kiedy ona będzie blisko. Analogicznie działa kolor biały wtedy odczyty są dużo bardziej dokładne wiąże się to z tym że kolor biały odbija więcej promieni podczerwonych które wrażają do czujników.

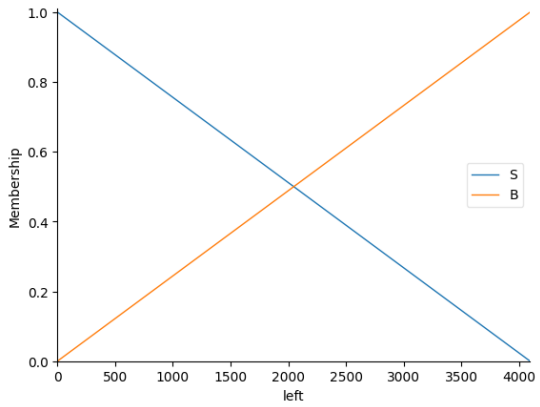
Promienie podczerwone nie muszą pochodzić tylko z czujników robota mogą również być emitowane z innego źródła co może wprowadzać zakłócenia(nie duże wartości jeśli źródło nie jest mocne) bądź być kryterium do naprowadzania robota.

3.2. Synteza sytemu ekspertowego

W tym punkcie zostanie omówione, jak stworzyć system ekspertowy dla robota, który będzie w stanie podejmować decyzje na podstawie analizy swojego otoczenia i podejmować odpowiednie działania. Do stworzenia takiego systemu potrzebny jest zestaw reguł i wiedzy, które będą decydować, jak robot powinien reagować na określone sytuacje. Istotne jest, aby reguły te były zdefiniowane w sposób jasny i precyzyjny, aby robot mógł ich poprawnie użyć i wykonać odpowiednie zadania. W tym punkcie zostanie omówione, jak zaprojektować system ekspertowy, który umożliwi skuteczne i inteligentne zachowanie robota.

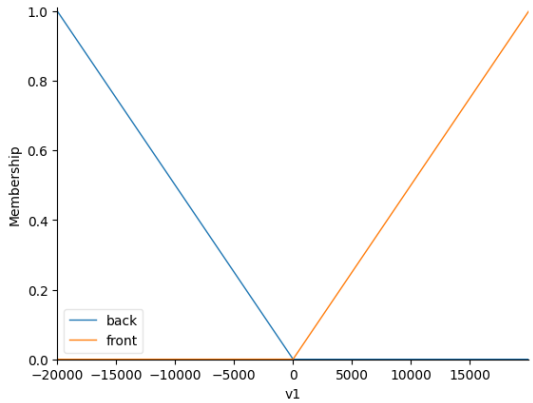
Do stworzenie takiego systemu użyjemy logiki rozmytej, systemu Mamdani'ego. Pierwsze co trzeba wykonać to zastanowić się co będzie naszym wejściem i co oczekujemy na wyjściu.

Wiemy że Khepera III posiada czajniki mierzące odległość można zastosować jako wejście gdzie będą dwie funkcje gdzie przynależność będzie określała czy robot jest blisko danego czajnika lub daleko. Dziedziną tych funkcji będzie zakres wartości jak otrzymujemy z czujników robota [0, 4095]. Zaproponowana funkcje przynależności dla każdego z czujników gdzie B oznacza że przeszkoda jest blisko czajnika natomiast S daleko .



Rysunek 1 funkcje przynależności jednego z czujników (left)

Wyjściami będą serwomechanizmy które mogą się obracać w przód i w tył w tym przypadku użyjemy wartości z przedziału –[20 000, 20 000].



Rysunek 2 funkcje wyjściowe dla jednego z serwomechanizmów

Kolejnym krokiem jest stworzenie reguł w tym celu skorzystano z wiedzy eksperckiej dzięki której otrzymaliśmy reguły:

Tabela 1 reguły do systemu omijania przeszkód

Lp.	left	front	right	VI	Vr
1	S	S	S	front	front
2	S	S	B	back	front
3	S	B	S	back	front
4	S	B	B	back	front
5	B	S	S	front	back
6	B	B	S	front	back
7	B	S	B	front	front
8	B	B	B	back	front

Mając wszystkie potrzebne elementy do stroszenia systemu rozmytego oraz wiedzy która funkcja musi zostać przysłonięta w bibliotece khepera3 jest to k3FuzzyAvoidDef() został stworzony kod.

```
def k3FuzzyAvoidDef():
    MaxProximitiSignal = 4096
    MaxSpeed = 20000
    left = ctrl.Antecedent(np.arange(0,MaxProximitiSignal,1), 'left')
    front = ctrl.Antecedent(np.arange(0,MaxProximitiSignal,1), 'front')
    right = ctrl.Antecedent(np.arange(0,MaxProximitiSignal,1), 'right')
    vl = ctrl.Consequent(np.arange(0,MaxSpeed,1), 'v1')
    vr = ctrl.Consequent(np.arange(0,MaxSpeed,1), 'v2')

    left['S'] = fuzz.trimf(left.universe, [0, 0, MaxProximitiSignal])
    left['B'] = fuzz.trimf(left.universe, [0, MaxProximitiSignal,
MaxProximitiSignal])

    front['S'] =fuzz.trimf(left.universe, [0, 0, MaxProximitiSignal])
    front['B'] =fuzz.trimf(left.universe, [0, MaxProximitiSignal,
MaxProximitiSignal])

    right['S'] =fuzz.trimf(left.universe, [0, 0, MaxProximitiSignal])
    right['B'] =fuzz.trimf(left.universe, [0, MaxProximitiSignal,
MaxProximitiSignal])

    vl['back'] = fuzz.trimf(left.universe, [-MaxSpeed, -MaxSpeed, 0])
    vl['front'] = fuzz.trimf(left.universe, [0,MaxSpeed, MaxSpeed])

    vr['back'] = fuzz.trimf(left.universe, [-MaxSpeed, -MaxSpeed, 0])
    vr['front'] = fuzz.trimf(left.universe, [0,MaxSpeed, MaxSpeed])

    # rules definition
    # Rule in a fuzzy control system, connecting antecedent(s) to
consequent(s)
    rule1 =ctrl.Rule(antecedent=(left['S'] & front['S'] &
right['S']),consequent=(vl['front'], vr['front']) )
    rule2 =ctrl.Rule(antecedent=(left['S'] & front['S'] &
right['B']),consequent=(vl['back'], vr['front']) )
    rule3 =ctrl.Rule(antecedent=(left['S'] & front['B'] &
right['S']),consequent=(vl['back'], vr['front']) )
    rule4 =ctrl.Rule(antecedent=(left['S'] & front['B'] &
right['B']),consequent=(vl['back'], vr['front']) )
    rule5 =ctrl.Rule(antecedent=(left['B'] & front['S'] &
right['S']),consequent=(vl['front'], vr['back']) )
    rule6 =ctrl.Rule(antecedent=(left['B'] & front['B'] &
right['S']),consequent=(vl['front'], vr['back']) )
    rule7 =ctrl.Rule(antecedent=(left['B'] & front['S'] &
right['B']),consequent=(vl['front'], vr['front']) )
    rule8 =ctrl.Rule(antecedent=(left['B'] & front['B'] &
right['B']),consequent=(vl['back'], vr['front']) )
```

```
    # The system is initialized and populated with a set of fuzzy Rules
    avoid_ctr =
ctrl.ControlSystem([rule1,rule2,rule3,rule4,rule5,rule6,rule7,rule8])
    # Calculate results from a ControlSystem
    avoid_sym = ctrl.ControlSystemSimulation(avoid_ctr)
    return avoid_sym
```

Kod 1 kod do systemu ekpertowego do omijania przeszkód

3.3. Przerobienie systemu aby robot podążał za przeszkodą ruchomą.

Aby robo podążał za przeszkodą należy zmienić reguły. Można wsnuć wniosek że to właśnie reguły głównym stopniu charakteryzują nam zachowanie robota a definiowanie funkcji dziedzin wybieranie systemów itp. Pomagają tylko w kompatybilności ze sprzętem.

Korzystając z wiedzy eksperckiej zostały stworzone kolejne reguły.

Tabela 2 reguły do systemu podążania za przeszkodą

Lp.	left	front	right	Vl	Vr
1	S	S	S	front	front
2	S	S	B	front	back
3	S	B	S	front	front
4	S	B	B	front	back
5	B	S	S	back	front
6	B	B	S	back	front
7	B	S	B	front	back
8	B	B	B	front	front

(wybieramy przeszkodę po prawej)

```
rule1 =ctrl.Rule(antecedent=(left['S'] & front['S'] &
right['S']),consequent=(vl['front'], vr['front']) )
    rule2 =ctrl.Rule(antecedent=(left['S'] & front['S'] &
right['B']),consequent=(vl['front'], vr['back']) )
    rule3 =ctrl.Rule(antecedent=(left['S'] & front['B'] &
right['S']),consequent=(vl['front'], vr['front']) )
    rule4 =ctrl.Rule(antecedent=(left['S'] & front['B'] &
right['B']),consequent=(vl['front'], vr['back']) )
    rule5 =ctrl.Rule(antecedent=(left['B'] & front['S'] &
right['S']),consequent=(vl['back'], vr['front']) )
    rule6 =ctrl.Rule(antecedent=(left['B'] & front['B'] &
right['S']),consequent=(vl['back'], vr['front']) )
    rule7 =ctrl.Rule(antecedent=(left['B'] & front['S'] &
right['B']),consequent=(vl['front'], vr['back']) )
    rule8 =ctrl.Rule(antecedent=(left['B'] & front['B'] &
right['B']),consequent=(vl['front'], vr['front']) )
```

Kod 2 reguły do systemu podążania za przeszkodą

3.4. Realizacja sytemu podążania za źródłem światła

W tym przypadku należy zmienić funkcje od odczytywania sensorów z k3ReadProximitySensors(s) na k3ReadAmbientSensors(s) która mierzy natężenie światła z otocznia. Charakterystyka tej funkcji jest odwrotna do k3ReadProximitySensors(s) duże wartości świadczą o małym natężeniu światła co można utożsamiać z dużą odległością źródła światła od robota a małe wartości o dużym natężeniu czyli bliskiej odległości źródła światła. Mając taką wiedzę możemy wysnuć wniosek że reguły nie będą się różnić od tych z zadania podążania za celem lecz zmieni się kolejność funkcji przynależności dla czujników.

Zmiana w kodzie została tylko zamienione nazwy B z S:

```
    left['B'] = fuzz.trimf(left.universe, [0, 0, MaxProximitiSignal])
    left['S'] = fuzz.trimf(left.universe, [0, MaxProximitiSignal,
MaxProximitiSignal])

    front['B'] =fuzz.trimf(front.universe, [0, 0, MaxProximitiSignal])
    front['S'] =fuzz.trimf(front.universe, [0, MaxProximitiSignal,
MaxProximitiSignal])

    right['B'] =fuzz.trimf(right.universe, [0, 0, MaxProximitiSignal])
    right['S'] =fuzz.trimf(right.universe, [0, MaxProximitiSignal,
MaxProximitiSignal])
```

Kod 3 zmiana definicji funkcji zmiennych wejściowych

oraz

```
def k3FuzzyAvoidLoop(s):
    avoid_sym = khep.k3FuzzyAvoidDef()
    res = False
    iter = 0
    while iter <= 1000:
        sens = khep.k3ReadAmbientSensors(s)
        print(sens)
        val_left = max(sens[1],sens[2])
        val_front = max(sens[3],sens[4])
        val_right = max(sens[5],sens[6])
        avoid_sym = k3FuzzyAvoidEval(avoid_sym, val_left, val_front,
val_right)
        khep.k3SetSpeed(s,avoid_sym.output['vl'],avoid_sym.output['vr'])
        iter += 1
```

Kod 4 zmiana funkcji dokonującej pomiary

3.5. Zmiana systemu na Takagi-Sugeno zerowego rzędu, część I

Nie biblioteka skfuzzy nie posiada modelu Takagi-Sugeno dlatego użyto simpful i posłuży do skonstruowania rozmytego systemu ekspertowego typu Takagi-Sugeno zerowego poziomu, który miał za zadanie omijać przeszkody. Simpful dostarcza narzędzia do budowy modeli wnioskowania rozmytego, umożliwiając użytkownikowi tworzenie zbiorów rozmytych, funkcji przynależności oraz reguł wnioskowania. Dzięki temu biblioteka ta pozwala na tworzenie skutecznych systemów ekspertowych opartych na logice rozmytej.

Omawianie kodu:

Poniższy przykład prezesuje problem omijania przeszkód 3.2.

- Stworzenie uchwytu do Systemu rozmytego oraz definiowanie funkcji przynależności zmiennych wejściowych

```
FS = sf.FuzzySystem()
MaxProximitiSignal = 4096
MaxSpeed = 20000

L_S = sf.FuzzySet(points=[[0., 1.],  [MaxProximitiSignal, 0.]], term="S")
L_B = sf.FuzzySet(points=[[0., 0.], [MaxProximitiSignal,1]], term="B")
FS.add_linguistic_variable("left", sf.LinguisticVariable([L_S, L_B],
concept="left"))

R_S = sf.FuzzySet(points=[[0., 1.],  [MaxProximitiSignal, 0.]], term="S")
R_B = sf.FuzzySet(points=[[0., 0.], [MaxProximitiSignal,1]], term="B")
FS.add_linguistic_variable("right", sf.LinguisticVariable([R_S, R_B],
concept="right"))

F_S= sf.FuzzySet(points=[[0., 1.],  [MaxProximitiSignal, 0.]], term="S")
F_B= sf.FuzzySet(points=[[0., 0.], [MaxProximitiSignal,1]], term="B")
FS.add_linguistic_variable("front", sf.LinguisticVariable([F_S, F_B],
concept="front"))
```

Kod 5 T-S definiowanie funckji zmiennych wejściowych

- Zefinowanie stałych wartości wyjściowych

```
FS.set_crisp_output_value("left_while_front", MaxSpeed)
FS.set_crisp_output_value("left_while_back", -MaxSpeed)

FS.set_crisp_output_value("right_while_front", MaxSpeed)
FS.set_crisp_output_value("right_while_back", -MaxSpeed)
```

Kod 6 T-S ustawianie zmiennych wejściowych

- Definiowanie reguł oraz dodanie ich do systemu

```
# Define fuzzy rules
R1 = "IF (left IS S) AND (front IS S) AND (right IS S) THEN (vl IS
left_while_front)  "
R2 = "IF (left IS S) AND (front IS S) AND (right IS B) THEN (vl IS
left_while_back)  "
R3 = "IF (left IS S) AND (front IS B) AND (right IS S) THEN (vl IS
left_while_back)   "
R4 = "IF (left IS S) AND (front IS B) AND (right IS B) THEN (vl IS
left_while_back)   "
R5 = "IF (left IS B) AND (front IS S) AND (right IS S) THEN (vl IS
left_while_front) "
R6 = "IF (left IS B) AND (front IS B) AND (right IS S) THEN (vl IS
left_while_front) "
R7 = "IF (left IS B) AND (front IS S) AND (right IS B) THEN (vl IS
left_while_front)  "
R8 = "IF (left IS B) AND (front IS B) AND (right IS B) THEN (vl IS
left_while_back)   "

R9= "IF (left IS S) AND (front IS S) AND (right IS S) THEN   (vr IS
right_while_front)"
R10 = "IF (left IS S) AND (front IS S) AND (right IS B) THEN (vr IS
right_while_front)"
R11 = "IF (left IS S) AND (front IS B) AND (right IS S) THEN  (vr IS
right_while_front)"
R12 = "IF (left IS S) AND (front IS B) AND (right IS B) THEN  (vr IS
right_while_front)"
R13 = "IF (left IS B) AND (front IS S) AND (right IS S) THEN (vr IS
right_while_back)"
R14 = "IF (left IS B) AND (front IS B) AND (right IS S) THEN (vr IS
right_while_back)"
R15 = "IF (left IS B) AND (front IS S) AND (right IS B) THEN  (vr IS
right_while_front)"
R16 = "IF (left IS B) AND (front IS B) AND (right IS B) THEN  (vr IS
right_while_front)"
FS.add_rules([R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R11, R12 , R13
,R14, R15, R16])
```

Kod 7 T-S Definiowanie reguł oraz dodanie ich do systemu

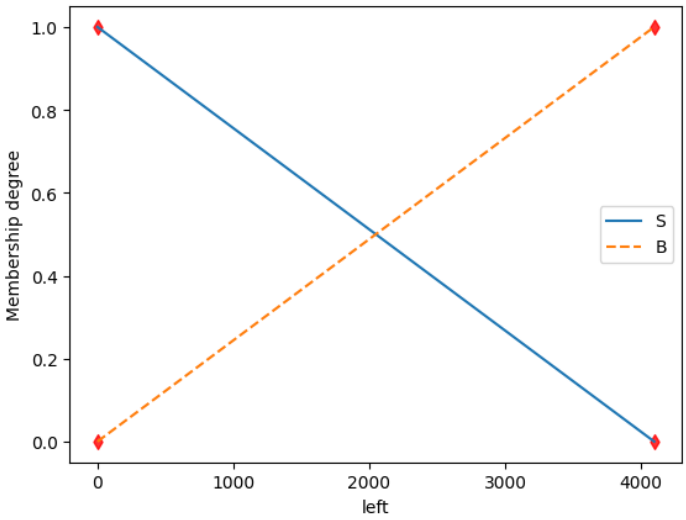
- Dodanie wartości na wejścia systemu, ewaluowanie systemu Takagi-Sugeno zerowego rzędu oraz wyrysowanie kilku funkcji dla zmiennej left

```
FS.set_variable("left", 100)
FS.set_variable("front", 100)
FS.set_variable("right", 100)

outputs=FS.Sugeno_inference(["vr","vl"])
print(outputs['vr'])
print(outputs['vl'])

FS.plot_variable("left")
```

Kod 8 T-S ustawianie wartości wejściowych testy

Output:

```
18331.943286071728
16592.844974446336
```



- Przeprowadzenie zgodności z tabelą reguł

```
dic={"S":0, "B":4000}
TR=np.array([[dic["S"],dic["S"],dic["S"]],  # 1 S   S   S   front   front
             [dic["S"],dic["S"],dic["B"]],  # 2 S   S   B   back    front
             [dic["S"],dic["B"],dic["S"]],  # 3 S   B   S   back    front
             [dic["S"],dic["B"],dic["B"]],  # 4 S   B   B   back    front
             [dic["B"],dic["S"],dic["S"]],  # 5 B   S   S   front   back
             [dic["B"],dic["B"],dic["S"]],  # 6 B   B   S   front   back
             [dic["B"],dic["S"],dic["B"]],  # 7 B   S   B   front   front
```

```
            [dic["B"],dic["B"],dic["B"]]]  # 8 B   B   B   back    front

for i in TR:
    FS.set_variable("left", i[0])
    FS.set_variable("front", i[1])
    FS.set_variable("right", i[2])
    outputs=FS.Sugeno_inference(["vr","vl"])
    print(f"Regula {i}  : vl={outputs['vl']} , vr={outputs['vr']}")
```

Kod 9 T-S Testowanie zgodności z tabelą reguł

Output:

```
Regula [   0    0    0]        : vl=20000.0 ,    vr=20000.0
Regula [   0    0 4096]        : vl=-20000.0 ,   vr=20000.0
Regula [   0 4096    0]        : vl=-20000.0 ,   vr=20000.0
Regula [   0 4096 4096]        : vl=-20000.0 ,   vr=20000.0
Regula [4096    0    0]        : vl=20000.0 ,    vr=-20000.0
Regula [4096 4096    0]        : vl=20000.0 ,    vr=-20000.0
Regula [4096    0 4096]        : vl=20000.0 ,    vr=20000.0
Regula [4096 4096 4096]        : vl=-20000.0 ,   vr=20000.0
```

Tabela dla porównania:

Lp.	left	front	right	VI	Vr
1	S	S	S	front	front
2	S	S	B	back	front
3	S	B	S	back	front
4	S	B	B	back	front
5	B	S	S	front	back
6	B	B	S	front	back
7	B	S	B	front	front
8	B	B	B	back	front

Można przyjąć pewne uproszczenia aby sprawdzić poprawność działania systemu na skrajnych wartościach:

- S= 0 przynależność do zbioru gdzie przeszkoda jest bardzo daleko jest największa
- B= 4096 przynależność do zbiory gdzie przeszkoda jest blisko jest największa
- front = 20 000
- back = -20 000

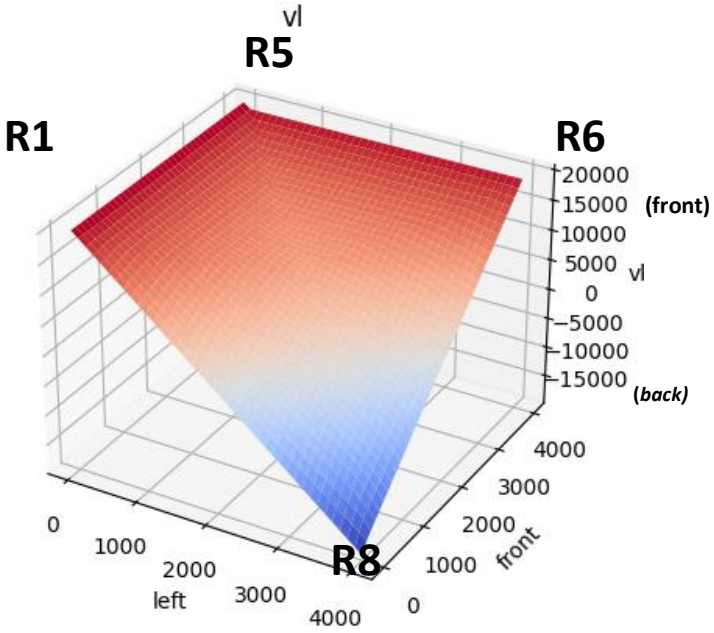
jeśli tak przyjmiemy i porównamy wyniki to rozkład wartości dla reguły zgadza się z tabelą zobaczymy że system działa poprawnie.

- Można również pokazać jak działa system nie tylko w skrajnych wartościach niestety mimo wszytko nie uda pokazać się całej przestrzeni możliwych rozwiązań ponieważ nie jesteśmy w stanie tworzyć wykresów więcej niż 3 wymiarowe dlatego można posłużyć się pewnymi zależnościami i np. przyjąć na wejście jednego z czujników max lub min wartość i rozpatrywać tylko jedno wyjście:

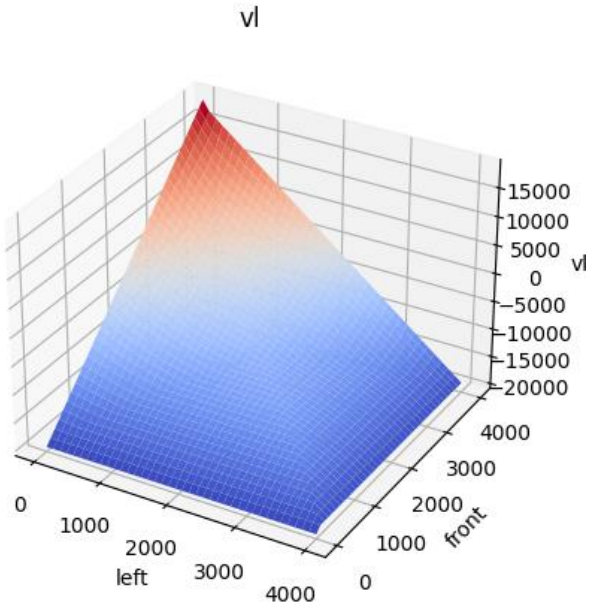
```
import numpy as np
import matplotlib.pyplot as plt
#from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline

plt_left= np.linspace(0, 4000, 40)
plt_front = np.linspace(0, 4000, 40)
X, Y = np.meshgrid(plt_left, plt_front)

vl=np.zeros((40,40))
for i in range(40):
    for j in range(40):
        FS.set_variable("left", plt_left[i])
        FS.set_variable("front", plt_front[j])
        FS.set_variable("right", 0)
        outputs=FS.Sugeno_inference(["vr","vl"])
        vl[i][j]=outputs['vl']


# Tworzenie wykresu 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, vl, cmap='coolwarm')

# Ustawienia osi
ax.set_title('vl')
ax.set_xlabel('left')
ax.set_ylabel('front')
ax.set_zlabel('vl')

# Pokazanie wykresu
plt.show()
```

Output:

Wartość right = min



Wartość right = max



Można zauważyć że skrajne rogi wykresów spełniają poszczególne reguły a powierzchnie miedzy nimi są możliwymi wartościami na wyjściu vl systemu.

3.6. Zmiana systemu na Takagi-Sugeno zerowego rzędu, część II

Zostały tylko zmienione reguły

```
# Define fuzzy rules
R1 = "IF (left IS S) AND (front IS S) AND (right IS S) THEN (vl IS
left_while_front)  "
R2 = "IF (left IS S) AND (front IS S) AND (right IS B) THEN (vl IS
left_while_front)  "
R3 = "IF (left IS S) AND (front IS B) AND (right IS S) THEN (vl IS
left_while_front)   "
R4 = "IF (left IS S) AND (front IS B) AND (right IS B) THEN (vl IS
left_while_front)   "
R5 = "IF (left IS B) AND (front IS S) AND (right IS S) THEN (vl IS
left_while_back) "
R6 = "IF (left IS B) AND (front IS B) AND (right IS S) THEN (vl IS
left_while_back) "
R7 = "IF (left IS B) AND (front IS S) AND (right IS B) THEN (vl IS
left_while_front)  "
R8 = "IF (left IS B) AND (front IS B) AND (right IS B) THEN (vl IS
left_while_front)   "

R9= "IF (left IS S) AND (front IS S) AND (right IS S) THEN   (vr IS
right_while_front)"
R10 = "IF (left IS S) AND (front IS S) AND (right IS B) THEN (vr IS
right_while_back)"
R11 = "IF (left IS S) AND (front IS B) AND (right IS S) THEN  (vr IS
right_while_front)"
R12 = "IF (left IS S) AND (front IS B) AND (right IS B) THEN  (vr IS
right_while_back)"
R13 = "IF (left IS B) AND (front IS S) AND (right IS S) THEN (vr IS
right_while_front)"
R14 = "IF (left IS B) AND (front IS B) AND (right IS S) THEN (vr IS
right_while_front)"
R15 = "IF (left IS B) AND (front IS S) AND (right IS B) THEN  (vr IS
right_while_back)"
R16 = "IF (left IS B) AND (front IS B) AND (right IS B) THEN  (vr IS
right_while_front)"

FS2.add_rules([R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R11, R12 , R13
,R14, R15, R16])

dic={"S":0, "B":4096}
TR=np.array([[dic["S"],dic["S"],dic["S"]],  # 1 S   S   S   front   front
             [dic["S"],dic["S"],dic["B"]],  # 2 S   S   B   back    front
             [dic["S"],dic["B"],dic["S"]],  # 3 S   B   S   back    front
```

```
            [dic["S"],dic["B"],dic["B"]],  # 4 S   B   B   back    front
            [dic["B"],dic["S"],dic["S"]],  # 5 B   S   S   front   back
            [dic["B"],dic["B"],dic["S"]],  # 6 B   B   S   front   back
            [dic["B"],dic["S"],dic["B"]],  # 7 B   S   B   front   front
            [dic["B"],dic["B"],dic["B"]]])  # 8    B   B   B   back    front

for i in TR:
    FS2.set_variable("left", i[0])
    FS2.set_variable("front", i[1])
    FS2.set_variable("right", i[2])
    outputs=FS2.Sugeno_inference(["vr","vl"])
    print(f"Regula {i}  : vl={outputs['vl']} , vr={outputs['vr']}")
```

Kod 10 zmiana kodu aby robot podążał za przeszkodą

Output:

```
Regula [   0    0    0]  : vl=20000.0 , vr=20000.0
Regula [   0    0 4096]  : vl=20000.0 , vr=-20000.0
Regula [   0 4096    0]  : vl=20000.0 , vr=20000.0
Regula [   0 4096 4096]  : vl=20000.0 , vr=-20000.0
Regula [4096    0    0]  : vl=-20000.0 , vr=20000.0
Regula [4096 4096    0]  : vl=-20000.0 , vr=20000.0
Regula [4096    0 4096]  : vl=20000.0 , vr=-20000.0
Regula [4096 4096 4096]  : vl=20000.0 , vr=20000.0
```

Tabela dla porównania:

Lp.	left	front	right	Vl	Vr
1	S	S	S	front	front
2	S	S	B	front	back
3	S	B	S	front	front
4	S	B	B	front	back
5	B	S	S	back	front
6	B	B	S	back	front
7	B	S	B	front	back
8	B	B	B	front	front

3.7. Zmiana systemu na Takagi-Sugeno zerowego rzędu, część III

A przerobi system do podążania za światłem należy postąpi analogicznie jak w podpunkcie 3.4.

Zostało również zamienione S z B: a

```
L_S = sf.FuzzySet(points=[[0., 1.],  [MaxProximitiSignal, 0.]], term="B")
L_B = sf.FuzzySet(points=[[0., 0.], [MaxProximitiSignal,1]], term="S")
FS2.add_linguistic_variable("left", sf.LinguisticVariable([L_S, L_B],
concept="left"))

R_S = sf.FuzzySet(points=[[0., 1.],  [MaxProximitiSignal, 0.]], term="B")
R_B = sf.FuzzySet(points=[[0., 0.], [MaxProximitiSignal,1]], term="S")
FS2.add_linguistic_variable("right", sf.LinguisticVariable([R_S, R_B],
concept="right"))

F_S= sf.FuzzySet(points=[[0., 1.],  [MaxProximitiSignal, 0.]], term="B")
F_B= sf.FuzzySet(points=[[0., 0.], [MaxProximitiSignal,1]], term="S")
FS2.add_linguistic_variable("front", sf.LinguisticVariable([F_S, F_B],
concept="front"))
```

Kod 11 zmiana kodu realizująca podążanie za światłem

Aby to przetestować tabela rebół będzie taka sama tylko wartości maksymalnych przynależności będzie inna ponieważ jest charakterystyka odczytywanych wartości z czujników

```
dic={"S":4096, "B":0}
```

Kod 12 zmiana kodu realizująca podążanie za światłem

Output:

```
Regula [4096 4096 4096]  : vl=20000.0 , vr=20000.0
Regula [4096 4096    0]  : vl=20000.0 , vr=-20000.0
Regula [4096    0 4096]  : vl=20000.0 , vr=20000.0
Regula [4096    0    0]  : vl=20000.0 , vr=-20000.0
Regula [   0 4096 4096]  : vl=-20000.0 , vr=20000.0
Regula [   0    0 4096]  : vl=-20000.0 , vr=20000.0
Regula [   0 4096    0]  : vl=20000.0 , vr=-20000.0
Regula [   0    0    0]  : vl=20000.0 , vr=20000.0
```

Tabela dla porównania:

Lp.	left	front	right	Vl	Vr
1	S	S	S	front	front
2	S	S	B	front	back
3	S	B	S	front	front
4	S	B	B	front	back
5	B	S	S	back	front
6	B	B	S	back	front
7	B	S	B	front	back
8	B	B	B	front	front

4. Ćwiczenie III realizacja podążania do celu.

4.1. Część I

Moja implementacja realizacji jadu do punktu o współrzędnych x,y

KOD:

```
# obliczanie korekty kąnta aby robot był zwrócony do celu
def correctAngle(s,x ,y,chi_C): # chi_C - obecny kąt , x,y - pozycja docelowa
    x_c, y_c = khep.k3ReadPosition(s)

    psi = lambda x, y : chi_c - np.arctan((y_c-y)/(x_c-x))

    return  psi(x,y)

#  oblicznie nakrutszej drogi do celu
def calculateDistance(s,x,y,chi_c):
    x_c, y_c = khep.k3ReadPosition(s)
    d = lambda x, y : np.sqrt((x_c-x)**2 + (y_c-y)**2)
    return d(x,y)


def goTo(s,x,y,chi_c):

    #funkcja obracająca robota
    #rotateRobot(s,correctAngle(s,x,y,chi_c)) #obrót w stronę celu


khep.setspeed(calculateDistance(s,x,y,chi_c),calculateDistance(s,x,y,chi_c)) #
jazda prosto
```

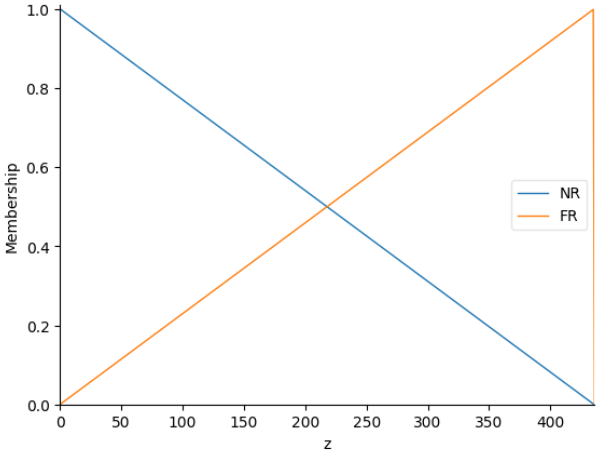
4.2. Część II

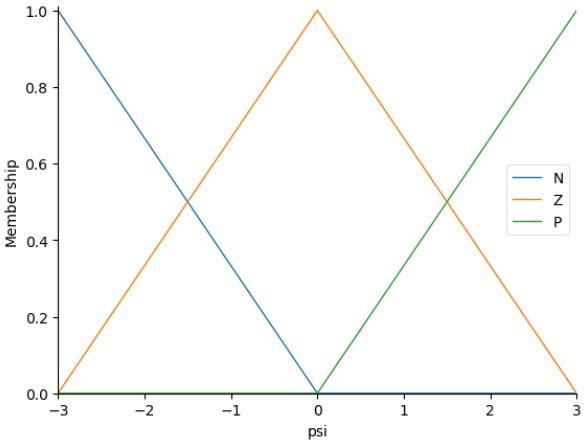
KOD:

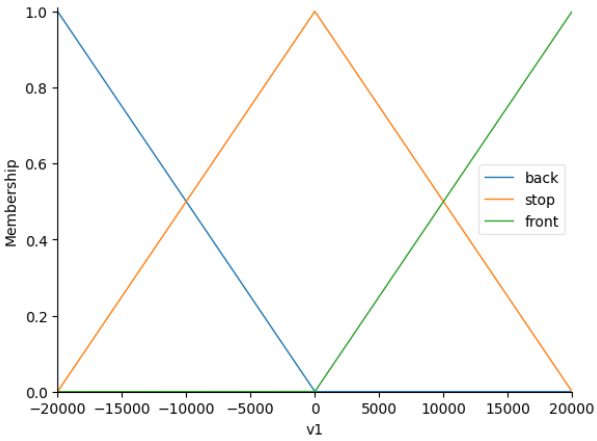
```
def k3FuzzyGoalDef():
    R           = 44 #[mm]
    MaxZ        = 7*R*np.sqrt(2)
    MaxSpeed    = 20000
    MaxPsi      = 3
    z           = ctrl.Antecedent(np.arange(0,MaxZ+1,1), 'z')
    psi         = ctrl.Antecedent(np.arange(-MaxPsi,MaxPsi+1,1), 'psi')
    vl          = ctrl.Consequent(np.arange(-MaxSpeed,MaxSpeed+1,1), 'v1')
    vr          = ctrl.Consequent(np.arange(-MaxSpeed,MaxSpeed+1,1), 'v2')
```

```
    z['NR']   = fuzz.trimf(z.universe, [0,    0, MaxZ])
    z['FR']   = fuzz.trimf(z.universe, [0,   MaxZ , MaxZ])

    psi['N']  = fuzz.trimf(psi.universe, [-MaxPsi,   -MaxPsi , 0])
    psi['Z']  = fuzz.trimf(psi.universe, [-MaxPsi,   0 , MaxPsi])
    psi['P']  = fuzz.trimf(psi.universe, [0,   MaxPsi , MaxPsi])

    vl['back']  = fuzz.trimf(vl.universe, [-MaxSpeed,   -MaxSpeed , 0])
    vl['stop']  = fuzz.trimf(vl.universe, [-MaxSpeed,   0 , MaxSpeed])
    vl['front'] = fuzz.trimf(vl.universe, [0,   MaxSpeed , MaxSpeed])

    vr['back']  = fuzz.trimf(vr.universe, [-MaxSpeed,   -MaxSpeed , 0])
    vr['stop']  = fuzz.trimf(vr.universe, [-MaxSpeed,   0 , MaxSpeed])
    vr['front'] = fuzz.trimf(vr.universe, [0,   MaxSpeed , MaxSpeed])

    z.view()
    psi.view()
    vl.view()

    # rules definition
    # Rule in a fuzzy control system, connecting antecedent(s) to
consequent(s)
    rule1 = ctrl.Rule(antecedent=(z['NR'] & psi['N']), consequent=(vl['back'],
vr['front']) )

    rule2 = ctrl.Rule(antecedent=(z['NR'] & psi['Z']), consequent=(vl['stop'],
vr['stop']) )

    rule3 = ctrl.Rule(antecedent=(z['NR'] & psi['P']),
consequent=(vl['front'], vr['back']) )

    rule4 = ctrl.Rule(antecedent=(z['FR'] & psi['N']), consequent=(vl['back'],
vr['front']) )

    rule5 = ctrl.Rule(antecedent=(z['FR'] & psi['Z']),
consequent=(vl['front'], vr['front']) )

    rule6 = ctrl.Rule(antecedent=(z['FR'] & psi['P']),
consequent=(vl['front'], vr['back']) )


    # The system is initialized and populated with a set of fuzzy Rules
    goal_ctr = ctrl.ControlSystem([rule1,rule2,rule3,rule4,rule5,rule6])
    # Calculate results from a ControlSystem
    goal_sym = ctrl.ControlSystemSimulation(goal_ctr)
    return goal_sym
```

Output:


Membership
z
NR
FR


Membership
psi
N
Z
P


Membership
v1
back
stop
front

4.3. Część III

Została podjęta próba napinania symulatora dla naszego sytemu rozmytego niestety nie został on ukończony co za tym idzie nie można było wkreślić dróg dla robota

KOD:

```
class Symu:
    def __init__(self, MPP=2, chiGL=0, startPosition=[0, 0]):
        self.MPP = MPP
        self.chiGL = chiGL
        self.position = startPosition

    #obliczenie prędkości liniowej
    def vlinear(self, vl, vr):
        return [vl * self.R, vr * self.R]

    def setPosition(self, x, y):
        self.position[0] = x
        self.position[1] = y

    def getPosition(self):
        return self.position

    def setAngle(self, angle):
        self.chiGL = angle

    def setSpeed(self, vl, vr):
        2+2
        vl_lin, vr_lin = self.vlinear(vl, vr)


        # zwrotu wektorów poszczególnych kół
        vxL = vl_lin * np.cos(self.chiGL)
        vyL = vl_lin * np.sin(self.chiGL)
        vxR = vr_lin * np.cos(self.chiGL)
        vyR = vr_lin * np.sin(self.chiGL)

        # obliczenie nowej pozycji
        new_x = ((vxL + vxR) / 2 + self.position[0])
        new_y = ((vyL + vyR) / 2 + self.position[1])

        # obliczenie nowego kąta względem osi x
        psi = lambda x, y: self.chiGL - np.arctan((self.position[0] - y) /
(self.position[1] - x))
```