

Controladores fuzzy de robô móvel

https://github.com/isabelatelles/mc906_collab/p3

ISABELA TELLES FURTADO DOSWALDO

RA 170012

E-mail: i170012@dac.unicamp.br

NATHÁLIA HARUMI KUROMIYA

RA 175188

E-mail: n175188@dac.unicamp.br

Resumo – O trabalho descreve o problema de definição de comportamentos do Pionner P3DX, sendo eles "seguir parede" e "andar até o objetivo". Para a implementação desses comportamentos, foi utilizado algoritmos fuzzy da biblioteca Scikit-Fuzzy, além do simulador V-REP. Com esses comportamentos, explorou-se, além da estrutura básica de fuzzy, a influência de diferentes funções de pertinência e de métodos de defuzzificação para obtenção de valores crisp. Com isso, foi possível obter esses dois comportamentos de forma satisfatória, ainda que com pequenas limitações de otimicidade devido a, principalmente, obstáculos no cenário.

Palavras-chave – Fuzzy, IA, V-REP

I. INTRODUÇÃO

A robótica móvel é a área da robótica que estuda a construção de robôs móveis, que possuem capacidade de mobilidade e autonomia para comportar-se de maneira adequada em um ambiente.

O intuito deste trabalho é explorar melhor essa área utilizando a lógica fuzzy para controlar um robô terrestre, o robô Pionner P3DX, em ambientes desconhecidos por ele e desenvolvidos no simulador V-REP com o auxílio do seguinte conjunto de slides [1].

Para a confecção do trabalho, foram estudados os conceitos da lógica fuzzy, apresentados nos slides [2], [3] e [4]. A motivação para compreender o funcionamento dessa lógica vem do fato de que ela está presente nos mais diversos aparelhos utilizados diariamente por nós, como câmeras digitais, geladeiras, máquinas de lavar roupa, etc. A explicação para o uso amplo da lógica fuzzy em aplicações reais é simples: ela admite valores que variam entre zero e um, que é o que de fato encontramos em problemas reais.

II. SEÇÕES

Este trabalho está separado nas seguintes seções:

- I. Introdução
- II. Seções
- III. Trabalho Proposto
- IV. Materiais e Métodos
 - IV-A. Segue parede à direita
 - IV-B. Andar até o objetivo

- V. Resultados e Discussão
- VI. Conclusões
- Referências

III. TRABALHO PROPOSTO[5]

O trabalho em questão propõe o desenvolvimento de controladores fuzzy independentes, capazes de fazer com que o robô Pionner P3DX: vá para determinado ponto no ambiente e se movimente seguindo a parede à sua direita no ambiente.

Como já mencionado na seção I. Introdução, foram gerados ambientes desconhecidos pelo robô no simulador V-REP, que podem ser encontrados na pasta *scenes*, de forma que seja possível avaliarmos como o robô reage em cada um dos ambientes para cada comportamento configurado.

IV. MATERIAIS E MÉTODOS

A implementação dos controladores fuzzy foi feita utilizando Python 3[6], além das bibliotecas Scikit-Fuzzy[7] e Numpy[8].

Para fazer a leitura das distâncias de obstáculos ao redor do robô foram utilizados os sensores ultrassônicos já embutidos no robô Pionner P3DX. A Figura 1 mostra a disposição desses sensores no robô.

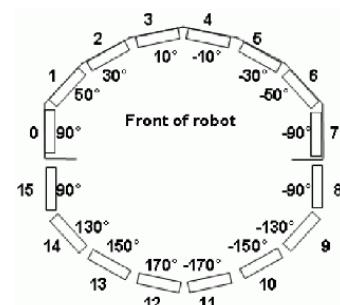


Figura 1. Disposição dos 16 sensores ultrassônicos no robô Pionner P3DX.

A. Segue parede à direita

1) *Variáveis de entrada e saída*: Para que o robô possa seguir a parede à sua direita, o controlador fuzzy admite as seguintes variáveis de **entrada**:

- Diferença entre os valores retornados pelos sensores à sua direita, de números 7 e 8, segundo a disposição mostrada na Figura 1;
- Valores retornados pelos sensores à sua frente, de números 3, 4 e 5, segundo a disposição mostrada na Figura 1.

Essas variáveis permitem que o robô perceba o ambiente ao seu redor, mais especificamente, a distância que ele está de uma parede ou obstáculo à sua frente e como ele está disposto em relação à parede a sua direita.

A função de pertinência modelada para a primeira variável de entrada mencionada pode ser vista na Figura 2. Na Figura 3 é possível observar a mesma função, variando apenas sua forma para gaussiana. Ambas as figuras representam a diferença entre as distâncias dadas pelo sensores à direita, 7 e 8, que pode variar entre negativa, zero e positiva. Se essa diferença é negativa, ou seja, a distância dada pelo sensor 7 é menor do que a distância dada pelo sensor 8, o robô está tendendo para a esquerda em relação à parede direita mais próxima. Se essa diferença, por sua vez, é zero, ou seja, a distância dada pelo sensor 7 é igual a distância dada pelo sensor 8, o robô está alinhado paralelamente à parede direita mais próxima. Mas, se essa diferença é positiva, ou seja, a distância dada pelo sensor 7 é maior do que a distância dada pelo sensor 8, o robô está tendendo para a direita em relação à parede direita mais próxima.

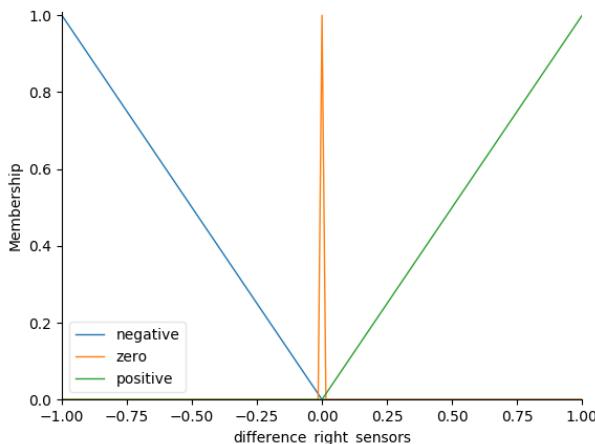


Figura 2. Função triangular de pertinência da diferença entre as distâncias dadas pelo sensores à sua direita.

Já as funções de pertinência modeladas para a segunda, terceira e quarta variáveis de entrada mencionadas são iguais. Elas consistem em uma função, vista na Figura 4, que considera se o robô está perto ou longe da parede à sua frente. A Figura 5 corresponde apenas a uma modificação na forma da função de pertinência da Figura 2, utilizada na configuração final.

As variáveis de saída, por sua vez, fazem com que o robô, de fato, se comporte da maneira esperada, ou seja,

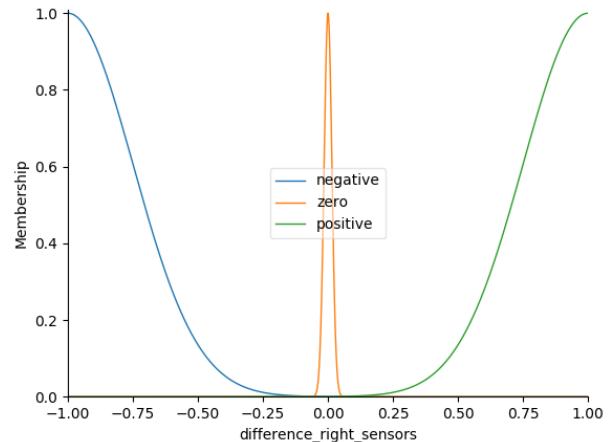


Figura 3. Função gaussiana de pertinência da diferença entre as distâncias dadas pelo sensores à sua direita.

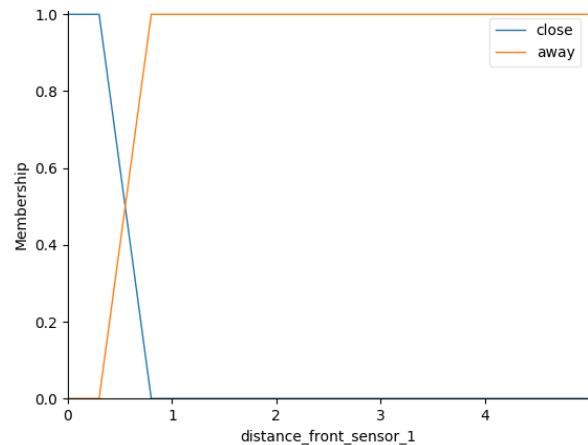


Figura 4. Função trapezoidal de pertinência das distâncias dadas pelo sensores à sua frente.

locomovendo-se sempre paralelamente à parede direita. São essas:

- Velocidade, em rad/s, da roda esquerda do robô;
- Velocidade, em rad/s, da roda direita do robô.

A função de pertinência adotada para cada velocidade é a mesma, e pode ser visualizada na Figura 6.

2) *Sistemas de regras e modelo de inferência:* A partir das variáveis de entrada, saída e suas funções de pertinência, é possível a implementação de um sistema de regras para o controlador fuzzy. Para simplificar, denominaremos a "diferença entre os sensores 7 e 8" de DIF, "qualquer uma das distâncias dadas pelos sensores, que serão mencionados, até a parede frontal" de DIST, "velocidade da roda esquerda" de VE e "velocidade da roda direita" de VD.

- **Regra 1:** Se DIST dos sensores 3, 4 e 5 é perto, então VE é zero.

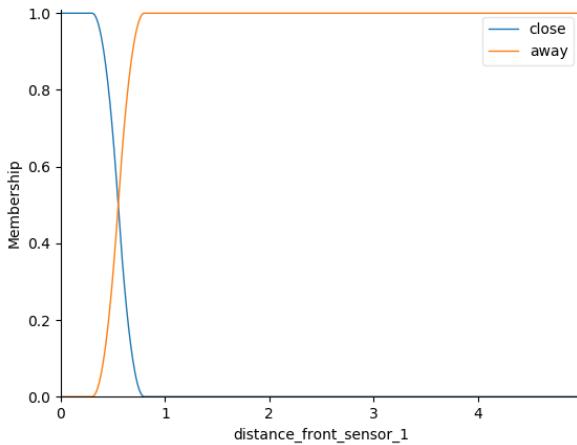


Figura 5. Função de pertinência, de formato S e Z, das distâncias dadas pelo sensores à sua frente.

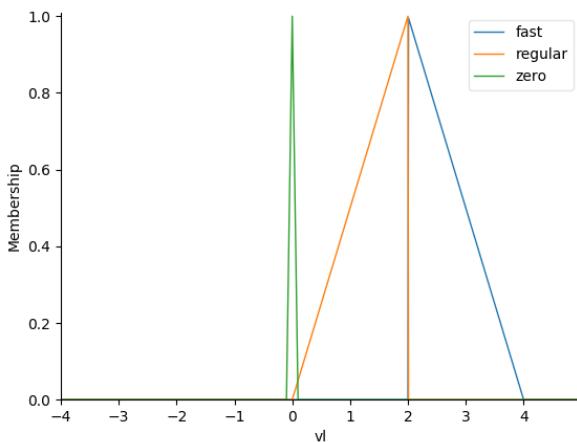


Figura 6. Função de pertinência da velocidade da roda do robô.

- **Regra 2:** Se DIST dos sensores 3, 4 e 5 é perto, então VD é **rápida**.
- **Regra 3:** Se DIF é positiva, e DIST dos sensores 3 e 4 é longe, então VE é **rápida**.
- **Regra 4:** Se DIF é positiva, então VD é **zero**.
- **Regra 5:** Se DIF é zero, e DIST dos sensores 3 e 4 é longe, então VE é **regular**.
- **Regra 6:** Se DIF é zero, e DIST dos sensores 3 e 4 é longe, então VD é **regular**.
- **Regra 7:** Se DIF é negativa, e DIST dos sensores 3 e 4 é longe, então VE é **regular**.
- **Regra 8:** Se DIF é negativa, e DIST dos sensores 3 e 4 é longe, então VD é **regular**.

As regras 1 e 2 fazem com que o robô vire à esquerda, as regras 3 e 4 fazem com que o robô vire à direita e as demais regras são necessárias para que o robô siga em frente sempre próximo à parede direita.

A biblioteca Scikit-Fuzzy implementa o modelo de inferência de Mamdani (max-min) para computar os valores das variáveis de saída. Entretanto, como cada variável de saída é um conjunto *fuzzy*, o método do centróide é aplicado para obter um valor *crisp* para cada uma delas.

B. Andar até o objetivo

1) Variáveis de entrada e saída: Para que o robô possa andar até seu objetivo, o sistema *fuzzy* leva em consideração as variáveis:

- A distância retornada por cada sensor, de forma que esses valores são avaliados separadamente, como será explicado a seguir
- A direção que o robô deve ser alinhado, levando em consideração a direção que está seu objetivo e a orientação em que o robô se encontra.

As distâncias lidas pelos sensores é responsável por evitar que o robô bata em algum obstáculo, seja ele uma parede ou um móvel. Esse trecho foi baseado no código fornecido pelo monitor, com modificações para fazer com que o robô não desvie totalmente de seu objetivo.

Já a direção é calculada através da soma do ângulo em que se encontra o objetivo em relação ao robô e o ângulo que o robô está rotacionado em relação ao sistema de eixos original.

A função de pertinência utilizada para as distâncias foi trapezoidal, apresentada na figura 7. Já a função utilizada para as direções foi uma função triangular representada na figura 8.

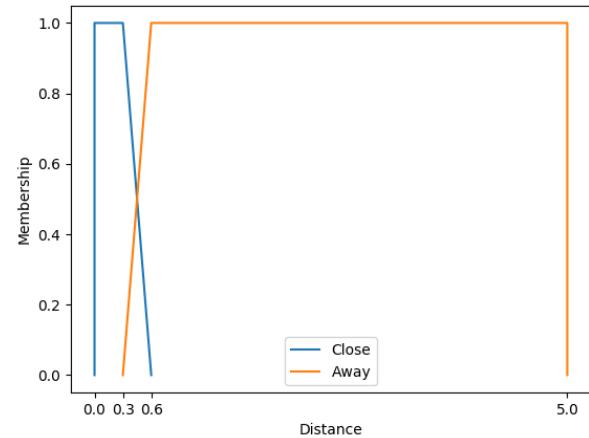


Figura 7. Função de pertinência da distância dos objetos.

Como variável de saída, temos variáveis que fazem com que o robô rotacione e ande em direção ao objetivo. Isso quer dizer:

- Velocidade, em rad/s, da roda esquerda do robô.
- Velocidade, em rad/s, da roda direita do robô.

A função de pertinência dessas saídas é a mesma entre as duas e é apresentada na figura 9.

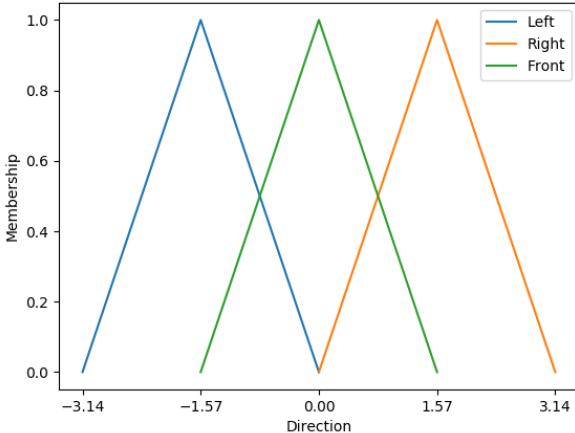


Figura 8. Função de pertinência da direção do objetivo.

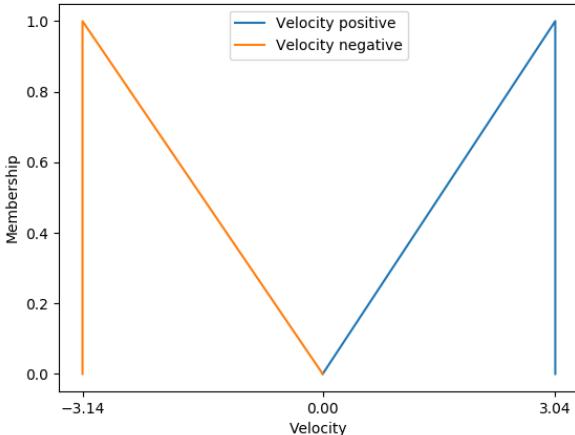


Figura 9. Função de pertinência da velocidade da roda do robô.

2) *Sistema de regras e modelo de inferência:* As regras implementadas para o funcionamento do sistema e determinação das velocidades das rodas é baseadas nos sensores e direção. Em relação ao sensores, foi considerado que:

- Os sensores **0, 1, 2 e 3** são sensores a esquerda.
 - Os sensores **4, 5, 6 e 7** são sensores a direita.
 - Os sensores **2, 3, 4 e 5** são sensores a frente.
- Dado essas convenções, as regras estipuladas foram:
- Para a roda esquerda:
 - **Regra 1:** Se há obstáculo a esquerda, então a velocidade é **positiva**.
 - **Regra 2:** Se há obstáculo a direita e a frente, então a velocidade é **negativa**.
 - **Regra 3:** Se há obstáculo a direita e a frente é livre, então a velocidade é **positiva**.
 - **Regra 4:** Se não há obstáculos perto e o objetivo está a direita ou em frente, então a velocidade é **positiva**.
 - **Regra 5:** Se não há obstáculos perto e o objetivo

está a esquerda, então a velocidade é **negativa**.

- Para a roda direita:

- **Regra 6:** Se há obstáculo a direita, então a velocidade é **positiva**.
- **Regra 7:** Se há obstáculo a esquerda e a frente, então a velocidade é **negativa**.
- **Regra 8:** Se há obstáculo a esquerda e a frente é livre, então a velocidade é **positiva**.
- **Regra 9:** Se não há obstáculos perto e o objetivo está a esquerda ou em frente, então a velocidade é **positiva**.
- **Regra 10:** Se não há obstáculos perto e o objetivo está a direita, então a velocidade é **negativa**.

As regras 1, 2, 3, 6, 7 e 8 evitam com que o robô bata em obstáculos, ao mesmo tempo em que faz com que a direção seja levemente controlada. Já as regras restantes tratam da direção que o robô deve seguir para chegar ao objetivo.

Como especificado anteriormente, a biblioteca Scikit-Fuzzy implementa o modelo de inferência de Mamdani (max-min) para computar os valores das variáveis de saída. Porém, como cada variável de saída é um conjunto *fuzzy*, o método de defuzzificação aplicado para obter um valor *crisp* foi variado e será apresentado na seção seguinte.

V. RESULTADOS E DISCUSSÃO

Primeiramente, iremos apresentar os resultados obtidos com a configuração final dos parâmetros (funções de pertinência, método de inferência, e sistema de regras) dos controladores. Em seguida, analisaremos as diferentes variações desses parâmetros e seus desdobramentos, que foram observados até atingir a configuração final com melhores resultados para cada comportamento do robô.

A. Segue parede

Na Figura 10 é possível visualizar um exemplo do robô seguindo a parede direita, de forma que realiza curvas à esquerda e à direita quando necessário, exatamente no ângulo preciso para deixá-lo paralelamente à parede.

Um dos primeiros desafios encontrados para conseguir um resultado como esse foi como fazer com que o robô conseguisse virar à esquerda com sucesso, sem que colidisse com a parede.

A solução para esse desafio foi utilizar o valor dado pelo sensor 5, além dos sensores 3 e 4, vide Figura 12. Nas primeiras configurações testadas, as regras 1 e 2 não incluíam a distância do sensor 5, o que fazia com que o robô colidisse, como pode ser visto na Figura 11. O uso do sensor 5, portanto, garante que o robô se alinhe paralelamente à parede direita após virar à esquerda.

Quando a função gaussiana de pertinência da diferença das distâncias entre os sensores à direita é utilizada, percebe-se uma mudança considerável no caminho feito pelo robô ao seguir a parede direita. Ao analisar as Figuras 13 e 14, percebe-se que a cada vez que o robô realiza uma curva sua distância da parede direita aumenta. Como não faria sentido que as curvas de negativo e positivo da função se sobreponessem e como a

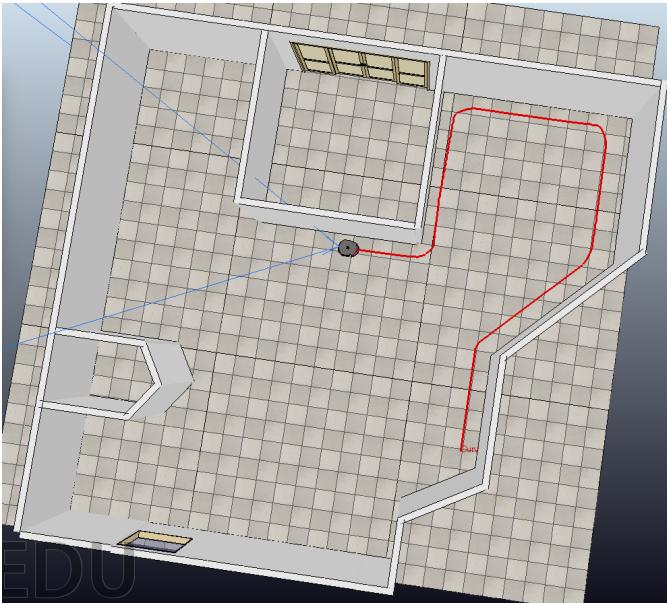


Figura 10. Exemplo do robô seguindo a parede direita no cenário *custom2*.

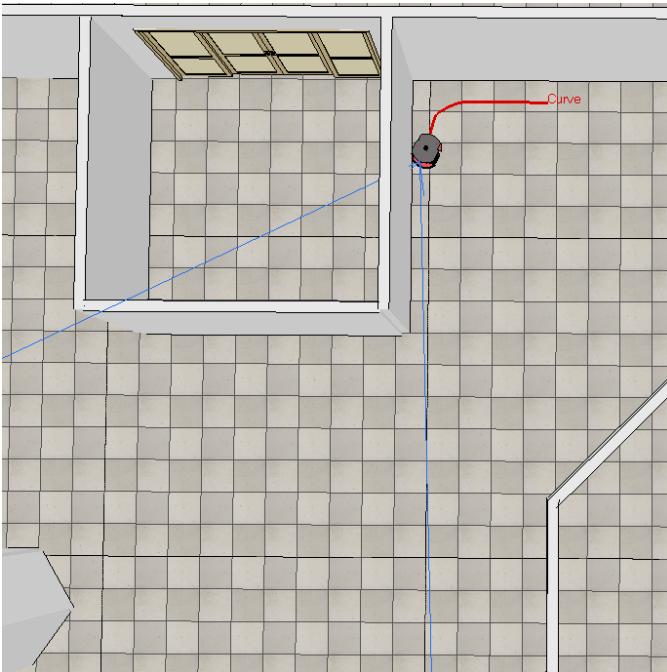


Figura 11. Exemplo do robô batendo na parede ao tentar virar à esquerda no cenário *custom2*. Nesse exemplo não se utiliza o valor da distância dada pelo sensor 5.

função tem universo contínuo, a extensão da curva de zero foi levemente aumentada. Isso faz com que a velocidade da roda permaneça muito perto de zero na curva por menos tempo, como pode ser observado na Figura 15, e é o bastante para que o robô se afaste consideravelmente da parede. Se, por exemplo, a parede à esquerda estivesse mais perto do robô nesse caso, ele iria colidir, já que os valores dos sensores da esquerda não são utilizados, e portanto, a parede da esquerda

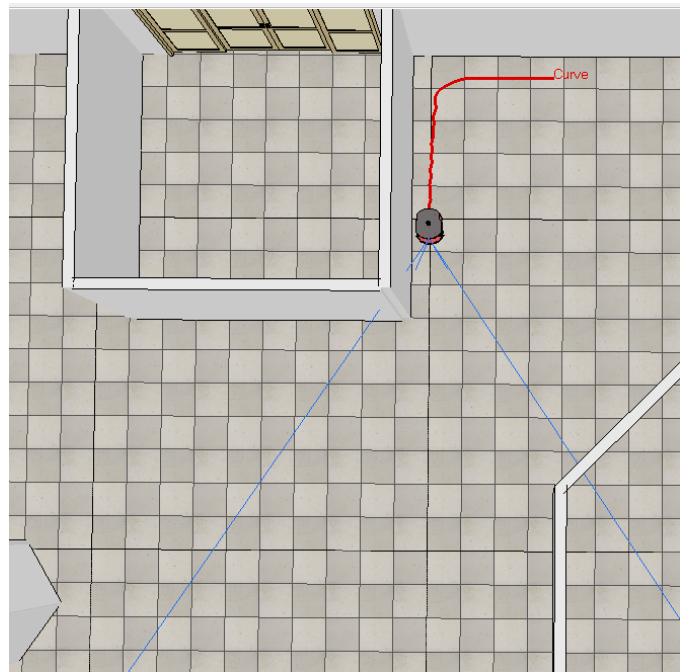


Figura 12. Exemplo do robô seguindo a parede direita com sucesso no cenário *custom2*.

não é perceptível para ele.

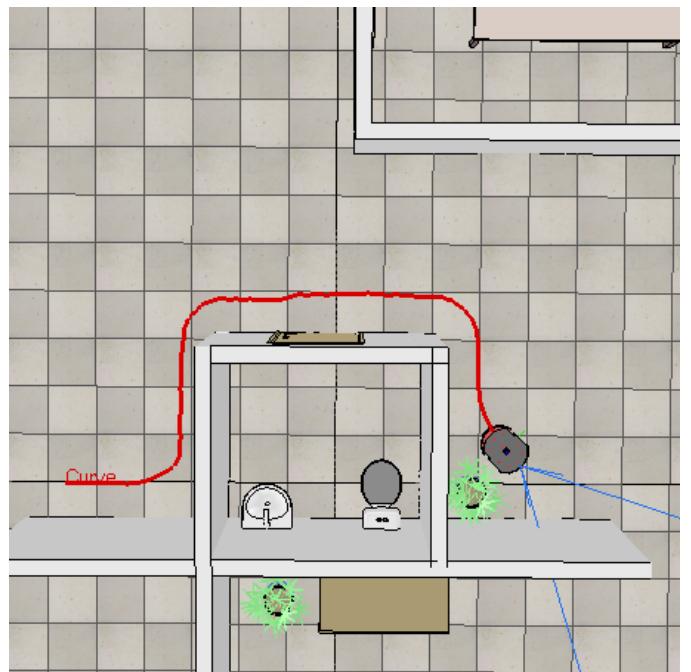


Figura 13. Exemplo do robô seguindo a parede direita no cenário *pd3dx* com a função triangular de pertinência vista na Figura 2.

Já quando os formatos S e Z são utilizados na função de pertinência das distâncias dadas pelos sensores 3, 4 e 5, a diferença nos resultados não é tão distoante. As Figuras 4 e 5 mostram o caminho que o robô percorre no cenário *custom1* quando a função de pertinência em questão tem formato

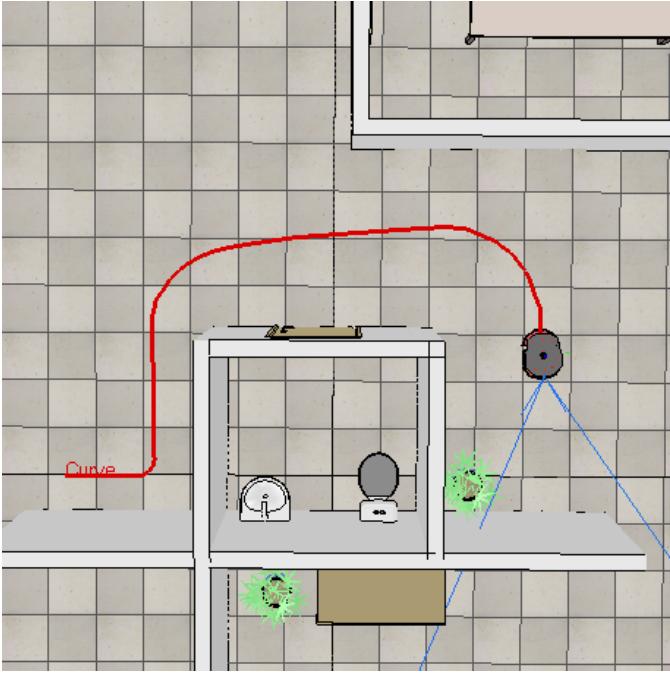


Figura 14. Exemplo do robô seguindo a parede direita no cenário *p3dx* com a função gaussiana de pertinência vista na Figura 2.

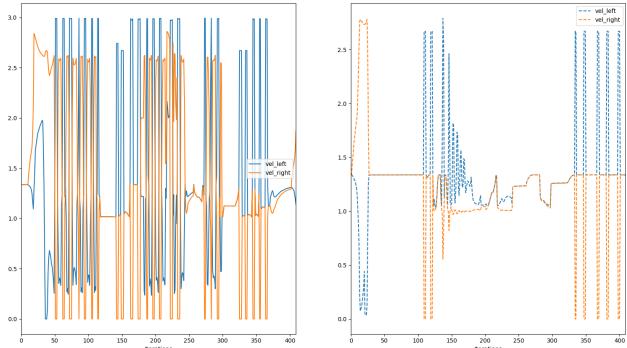


Figura 15. Velocidades da roda esquerda e direita em cada iteração. À esquerda, as velocidades com o uso da função triangular de pertinência (vide Figuras 2 e 13). À direita, as velocidades com o uso da função gaussiana de pertinência (vide Figuras 3 e 14).

trapezoidal e S/Z, respectivamente. A olho nu, é possível perceber que o formato S e Z favorece que após uma curva à esquerda, o robô se alinhe melhor paralelamente à parede. Analisando os gráficos das velocidades na Figura 18, percebe-se que o formato S e Z varia mais as velocidades, no geral, em relação ao formato trapezoidal.

O resultado mostrado na figura 19 ilustra como o robô se comporta ao iniciar em uma posição longe de qualquer parede. Como esperado, o robô segue em frente até encontrar uma parede à sua frente, e então começa a segui-la, de forma que se alinha paralelamente a ela.



Figura 16. Exemplo do robô seguindo a parede direita no cenário *custom1* com a função trapezoidal de pertinência vista na Figura 4.

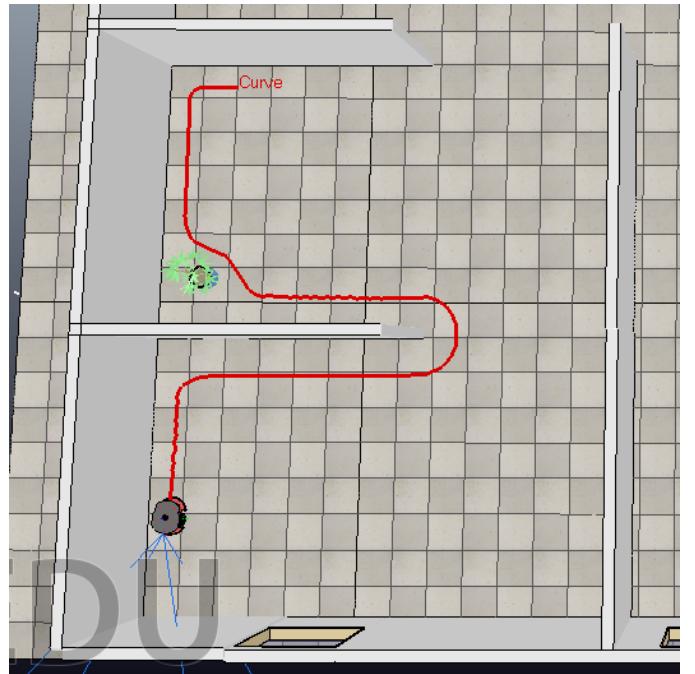


Figura 17. Exemplo do robô seguindo a parede direita no cenário *custom1* com a função de pertinência de formato S e Z vista na Figura 5.

B. Andar até o objetivo

A primeira dificuldade encontrada era a necessidade de juntar dois comportamentos distintos do robô: evitar obstáculos e seguir para o objetivo. Dessa forma, foi necessária a implementação das regras 1, 2, 3, 6, 7 e 8, descritas na seção anterior. Essa solução permitiu com que, ainda que com

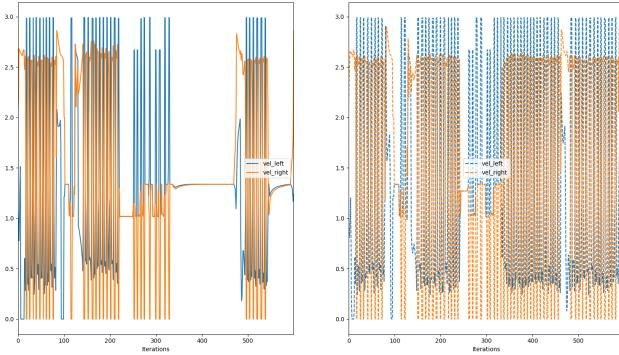


Figura 18. Velocidades da roda esquerda e direita em cada iteração. À esquerda, as velocidades com o uso da função trapezoidal de pertinência (vide Figuras 4 e 16). À direita, as velocidades com o uso da função de pertinência de formato S e Z (vide Figuras 5 e 17).

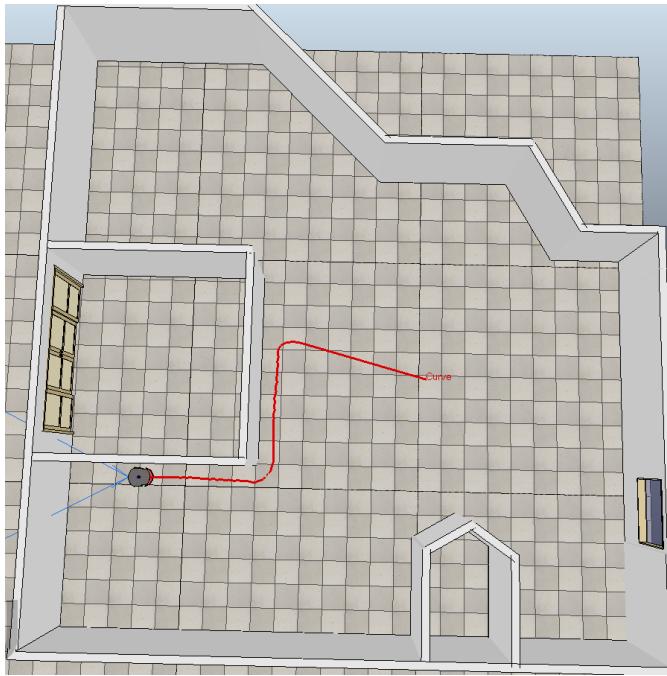


Figura 19. Exemplo do robô seguindo a parede direita partindo de uma posição no meio do cenário *custom2*

desvios não intuitivos, o robô chegasse em seu objetivo, como mostra a figura 20.

Após esse teste, então, o robô foi submetido a outros caminhos e outro cenário de teste para encontrar suas limitações. Esses testes estão apresentados nas figuras 21 e 22.

Um dos testes feitos foi passar de um cômodo para outro. Esse teste não teve sucesso e o robô apenas passava reto pela porta, sem entrar no cômodo objetivo.

Porém, ao procurar uma alternativa para refinar a solução, a implementação de regras diferentes e variações dos conjuntos fuzzy de entrada não pareceu satisfatório. Conseguimos uma solução, então, a partir da troca do método de defuzzificação para obter um valor crisp. Essas soluções são apresentadas nas figuras 23 e 24.

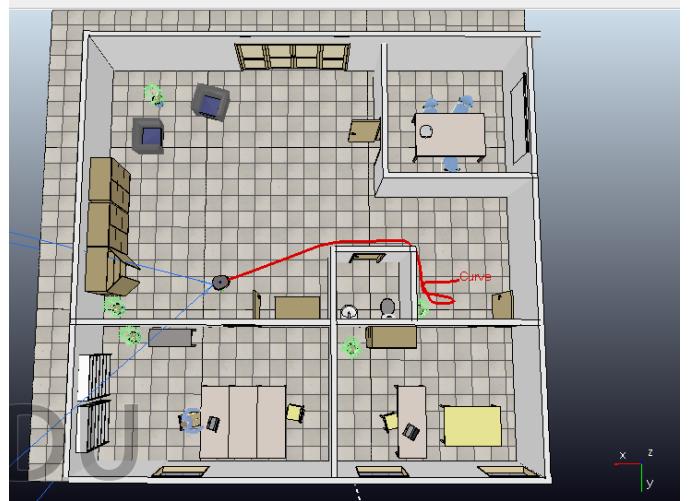


Figura 20. Exemplo do robô indo até seu objetivo do ponto (-5.2, 1.6) até o ponto (2.0, 1.7) com o método de centroid no cenário *pd3dx*

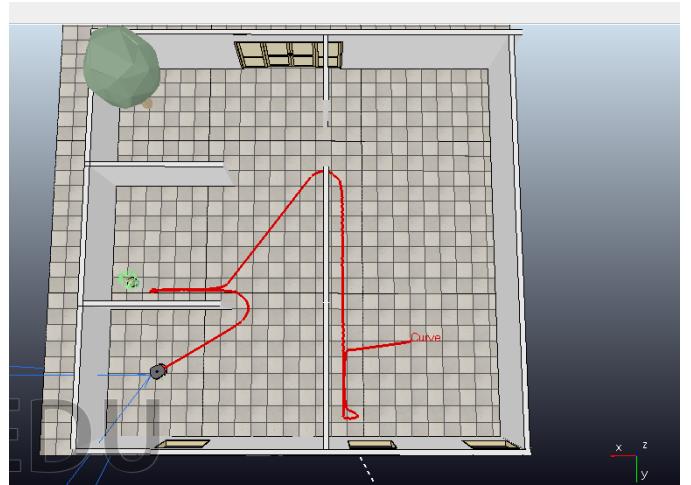


Figura 21. Exemplo do robô indo até seu objetivo do ponto (-4.0, 4.0) até o ponto (4.0, 5.0) com o método de centroid no cenário *Custom1*

Com essa modificação, então, testamos os métodos disponibilizados pela biblioteca:

- Centroid
- Bisector
- MOM (média do máximo)
- SOM (mínimo do máximo)
- LOM (máximo do máximo)

É importante destacar as diferenças encontradas em cada método para as escolhas das velocidades. Para o caso do ponto (-2.0, -3.5) até o ponto (0.8, -4.5), temos as médias do valor absoluto das velocidades negativas e positivas apresentadas na figura 25. Como esperado, os métodos escolhem valores maiores de velocidade da esquerda para direita. As velocidades marcadas como negativas estão invertidas, dado que o gráfico mostra o valor absoluto delas.

Com a seleção de velocidades diferentes, os caminhos adotados pelo robô em cada método também é diferente, bem

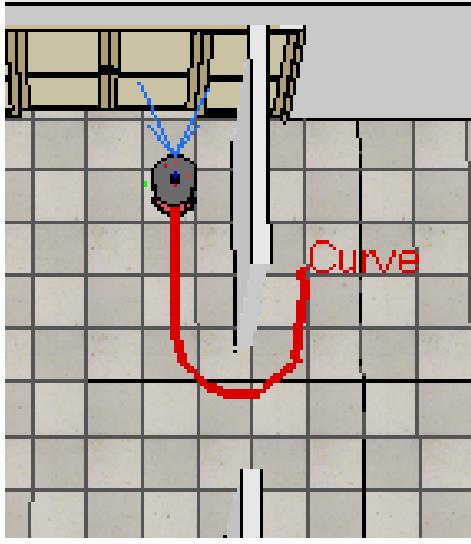


Figura 22. Exemplo do robô indo até seu objetivo do ponto (-2.0, -3.5) até o ponto (0.8, -4.5) com o método de centroid no cenário *Custom1*

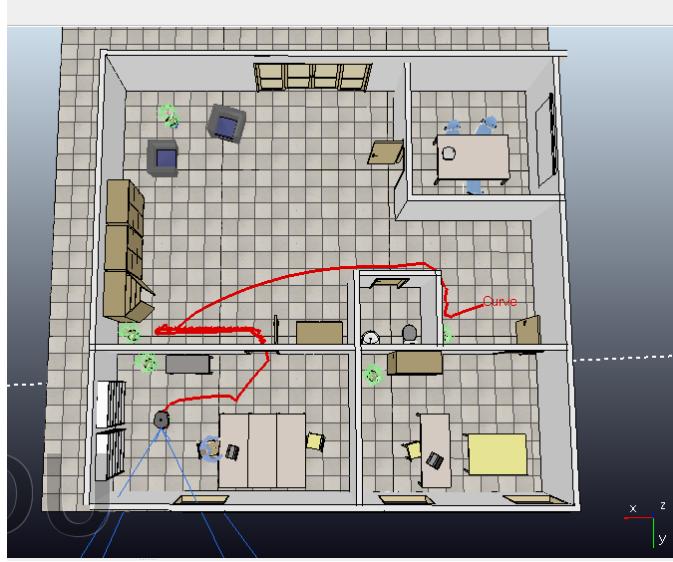


Figura 23. Exemplo do robô indo até seu objetivo do ponto (-5.2, 1.6) até o ponto (4.0, 5.0) com o método *mom* (média do máximo) no cenário *pd3dx*

como o tempo que cada método leva para chegar no resultado. Podemos visualizar esses resultados nas figuras 26 e 27.

VI. CONCLUSÕES

O trabalho contou com dificuldades de confecção. O mais relevante deles foi a manipulação do simulador. Isso inclui a mudança das posições do robô, a confecção de novas cenas, etc.

A manipulação da biblioteca Scikit-Fuzzy também se mostrou mais árdua, dado a documentação mais simplificada e com poucos exemplos de utilização, sendo necessária uma pesquisa mais abrangente em fontes não oficiais.

Quanto ao que diz respeito ao comportamento de seguir parede, o robô está limitado a seguir a parede à sua direita, o

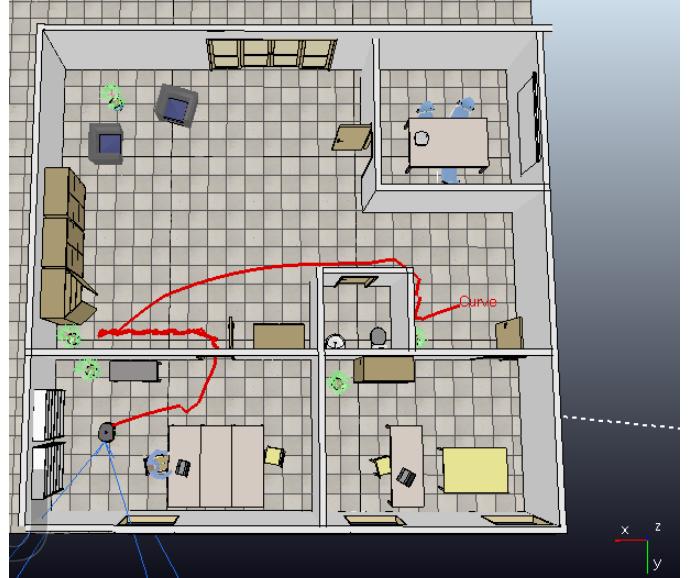


Figura 24. Exemplo do robô indo até seu objetivo do ponto (-5.2, 1.6) até o ponto (4.0, 5.0) com o método *lom* (média do máximo) no cenário *pd3dx*

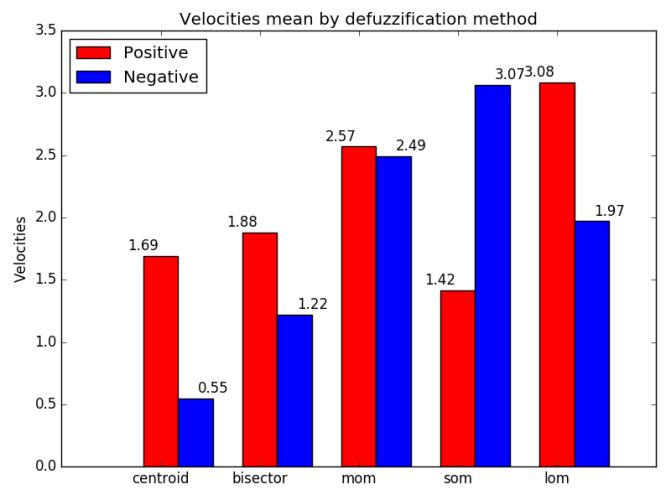
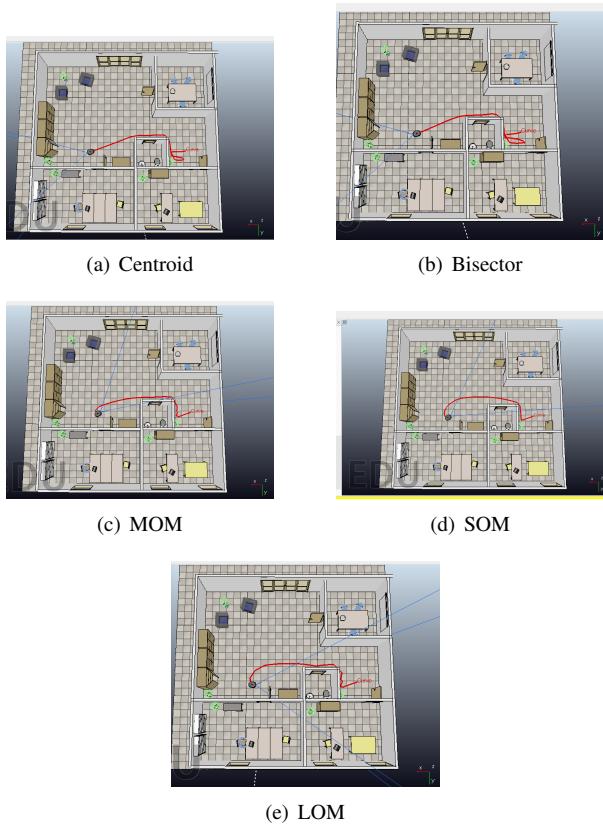


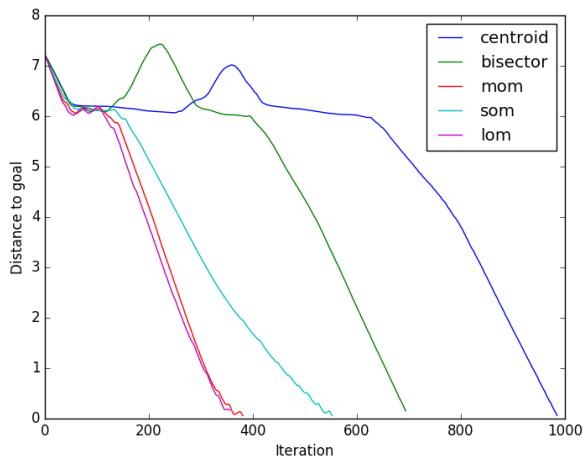
Figura 25. Média das velocidades do robô indo até seu objetivo do ponto (-5.2, 1.6) até o ponto (4.0, 5.0) de acordo com cada método.

que não é ideal para todos os cenários possíveis, como cenários com paredes que não formem polígonos, e até mesmo para os cenários apresentados neste trabalho, caso o robô comece em uma posição muito perto de uma parede à sua frente, por exemplo.

Já quanto ao comportamento de seguir parede, o robô apresenta solução em todos os casos testados, mas nem sempre com o caminho ótimo e nem com todos os métodos de defuzzificação. Isso porque os métodos que escolhem velocidades altas podem perder precisão. Esse fator dificulta alguns resultados, como mostrado na figura 27, dado que faz o robô pegar um caminho menos intuitivo, demorando para encontrar o objetivo ou até não encontrando. Ao mesmo tempo, esse mesmo comportamento gera rotações mais bruscas, que pode



Distance to the goal by iteration - Initial: (-5.2, 1.6) - Goal: (2, 1.7)



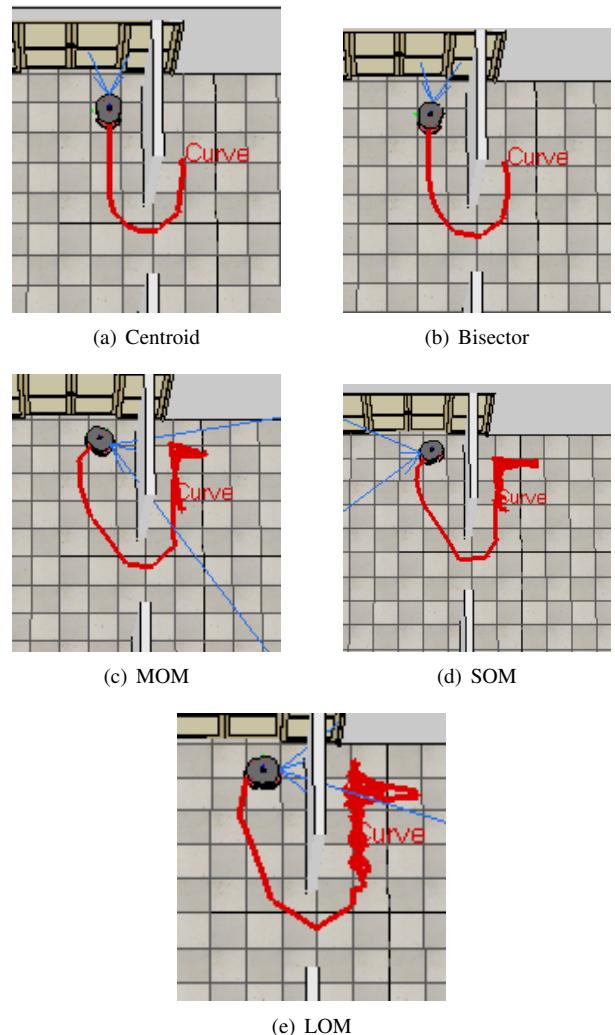
(f) Distância faltante em relação ao número de iterações

Figura 26. Figuras relacionadas ao robô para o caso (-2.0, -3.5) até o ponto (0.8, -4.5).

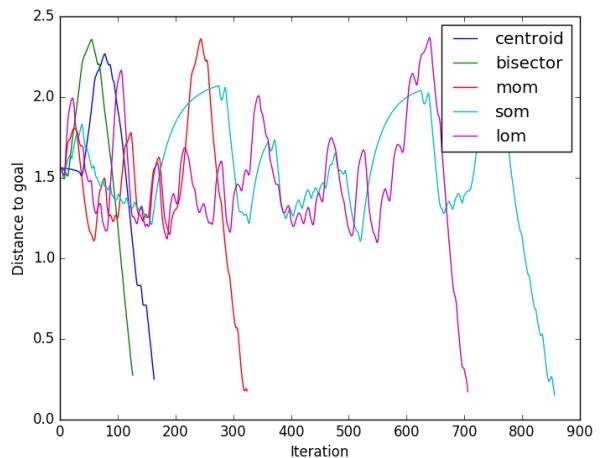
auxiliar no encontro do caminho para o objetivo, como mostra na figura 26.

REFERÊNCIAS

- [1] A. d. S. S. Esther Luna Colombini, "Mobile robotics: V-rep simulator." [Online]. Available: <https://www.ic.unicamp.br/~esther/teaching/2019s1/mc906/vrep.pdf> 1
- [2] E. L. Colombini, "Fuzzy ii." [Online]. Available: <https://www.ic.unicamp.br/~esther/teaching/2019s1/mc906/Aula11.pdf> 1



Distance to the goal by iteration - Initial: (-2.0, -3.5) - Goal: (-0.8, -4.5)



(f) Distância faltante em relação ao número de iterações

Figura 27. Figuras relacionadas ao robô para o caso (-5.2, 1.6) até o ponto (2.0, 1.7)

- [3] ——, "Fuzzy ii." [Online]. Available: <https://www.ic.unicamp.br/~esther/teaching/2019s1/mc906/Aula12.pdf> 1

- [4] ——, “Fuzzy iii.” [Online]. Available: <https://www.ic.unicamp.br/~esther/teaching/2019s1/mc906/Aula13.pdf>
- [5] ——, *Projeto 3*. [Online]. Available: <https://www.ic.unicamp.br/~esther/teaching/2019s1/mc906/P3.pdf>
- [6] “Python 3.” [Online]. Available: <https://www.python.org/>
- [7] “Scikit-fuzzy.” [Online]. Available: <https://pythonhosted.org/scikit-fuzzy/>
- [8] “Numpy.” [Online]. Available: <https://www.numpy.org/>