Projeto 3 - MC906A

Marcelo Biagi Martins - 183303
Erick Seiji Furukawa - 170600

1 Resumo

A lógica fuzzy é uma forma de lógica em que as variáveis lógicas podem assumir qualquer valor real entre 0 e 1, onde 0 corresponde ao valor totalmente falso e 1 ao valor totalmente verdadeiro. Este tipo de lógica permite que estados intermediários de verdade existam, tornando possível realizar a análise de conceitos não quantificáveis como uma distância, que pode ser perto, médio ou longe, que não necessariamente possui um valor numérico atrelado a ela. O projeto 3 de MC906 apresentado neste relatório envolveu o desenvolvimento de dois comportamentos para o robô P3-DX do simulador V-REP utilizando lógica fuzzy: Segue parede e desvia de obstáculos. Foram realizados diversos testes ao realizar alterações nos diferentes parâmetros do robô para analisar a mudança em seu comportamento.

2 Introdução

A lógica Fuzzy, opostamente à booleana, utiliza um conceito de "partial truth" que não é polarizada apenas entre dois valores como a última, mas sim dividida na quantidade desejada de valores entre 0 e 1 para representar as "meias verdades". Assim, é um tipo de lógica que parece mais com as noções humanas de averiguação de resultado, pois realiza um balanço mais ponderado das opções [1].

A lógica fuzzy busca descrever o problema através de membership functions, que podem assumir diversas formas de representação dos dados, como triângulos e trapézios; realizar os cálculos necessários e devolver o resultado "desfuzzificado", de forma "crisp". As variáveis podem se sobrepor, como exemplo, uma variável fuzzy "baixo" pode ter valores (1,4) e uma "alta" pode ter valores (3,5), sendo que elas possuem um intervalo em comum, que é a forma de por na prática o conceito de meia verdade. O número 3, nesse caso, não é totalmente alto nem totalmente baixo, sendo uma mistura dos dois.

Neste projeto de MC906, foi desenvolvido dois comportamentos para um robô Pioneer P3-DX: seguir parede e evitar obstáculos. Cada um desse com suas variáveis fuzzy que implementam a lógica acima mencionada.

3 O modelo

Para desenvolver o script dos robôs com lógica fuzzy, foi utilizado a biblioteca de funções Scikit-Fuzzy para python, sendo que os mesmo também foram feitos nessa linguagem.

Apesar do script de seguir parede já desviar de obstáculos, foi criado um outro com lógica distinta para melhorar a precisão dos testes. Além disso, parâmetros como o cenário e o range das variáveis fuzzy de cada comportamento foram variados. O modelo de inferência escolhido para eles foi o de Mamdani, pois todas as regras de ambos os comportamentos são da forma "se x_1 é A_1 e x_2 é A_2 então u é B", em que x1 e x2 são antecedentes e B é consequente. O primeiro script desenvolvido foi o segueparede.py, que comanda o P3-DX a seguir uma parede pela direita sempre, analisando os sensores da direita e da frente. A lógica básica que envolve o código é simples e com quatro instruções [2], conforme a tabela:

Tabela 1: **Regras básicas de como seguir parede**

Frente	Direita	Comando
Perto	Perto	Chegou em uma quina, curva abrupta à esquerda
Longe	Perto	Está seguindo a parede, continuar em frente
Perto	Longe	Obstáculo a frente, virar abrubtamente à esquerda para desviar e se manter paralelo à parede
Longe	Longe	Longe da parede, virar suavemente à direita para chegar mais perto

No entanto, a lógica utilizada não era tão enxuta, pois as variáveis fuzzy abrangiam mais casos do que os da tabela. Assim, no segueparede.py, dividiu-se as variáveis em duas antecedents, uma para a distância da frente e outra para a da direita, e dois consequents, para a velocidade e a direção que o robô deve adotar após as medições. As distâncias foram divididas em quatro: muito perto, perto, médio e longe, enquanto que a direção foi dividida em oeste, noroeste, norte e nordeste e a velocidade, por sua vez, em nula, devagar e rápido.

A princípio, foi utilizada a função ctrl.automf() para determinar as funções de membership automaticamentes com triângulos como na figura 2 para a distância frontal. Após alguns testes, observou-se que o robô batia na parede com frequência, então foi decidido criar uma função membership mais complexa, com a variação nas distâncias consideradas perto, médio e longe (figura 1).

O script era simples: os sensores da direita(7 e 8) e o da frente(4) eram lidos, e foram criadas regras para determinar a direção que o robô deve virar, e a velocidade com que deve prosseguir.

Tabela 2: **Algumas regras do robô que segue parede**

Frente	Direita	Comando
Muito perto	-	O robô está prestes a bater: parar o robô e virar a esquerda
Perto	-	Virar a esquerda com velocidade lenta
Medio	-	Virar a diagonal a esquerda (noroeste) com velocidade devagar
Longe	-	Prosseguir com velocidade rápida
Longe	Longe	Está longe de uma parede: Virar a diagonal a direita (nordeste)
Longe	Medio	Seguir em frente
Longe	Perto	Virar a esquerda com velocidade devagar
Longe	Muito Perto	Virar a esquerda com velocidade nula

A ideia do robô era utilizar os dados de distância da frente e da direita para calibrar sua velocidade e a direção em que seguir. Para utilizar como input e realizar o cálculo da direção e velocidade, foram utilizados o sensor da frente (4) como input da distância frontal, e o valor mínimo dos sensores da direita (7 e 8) como input da distância à direita. Foram utilizados dois sensores para a distância à direita, pois caso existam vãos na parede, o robô pode prosseguir com um comportamento inadequado, ao assumir que não há obstáculos. Utilizando dois sensores, a possibilidade de ocorrer este problema diminui, já que um dos sensores é perpendicular ao eixo do robô e o outro está na diagonal superior direita, fornecendo um espaço amplo de análise. O output gerado (variáveis consequent) é alimentado no robô através da função "robot.set_velocity()"do arquivo "robot.py".

Foi observado que, apesar do robô andar perto das paredes como o planejado, ele estava zigueza-gueando, o que foi considerado como uma movimentação indesejada e, portanto, uma outra versão do script foi desenvolvida para tentar calcular o ângulo relativo entre o robô e a parede e, assim,

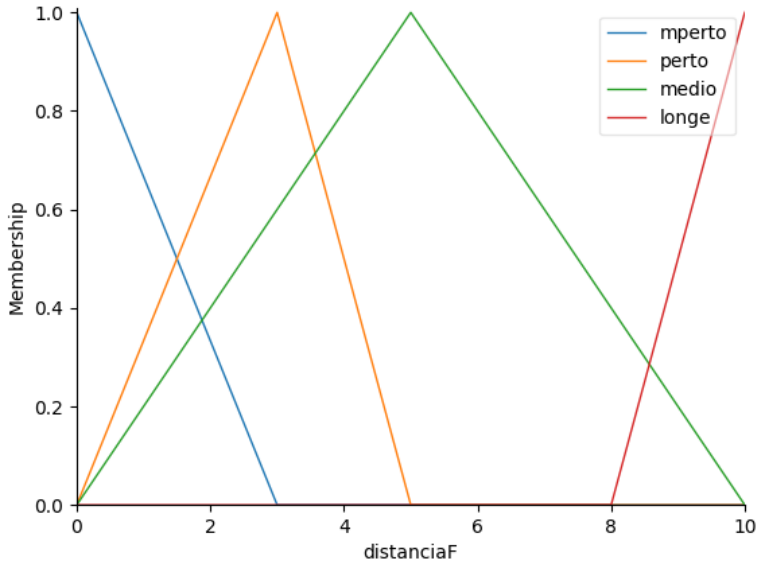

Figura 1: Membership function da variavel de distância frontal para o robô que segue parede

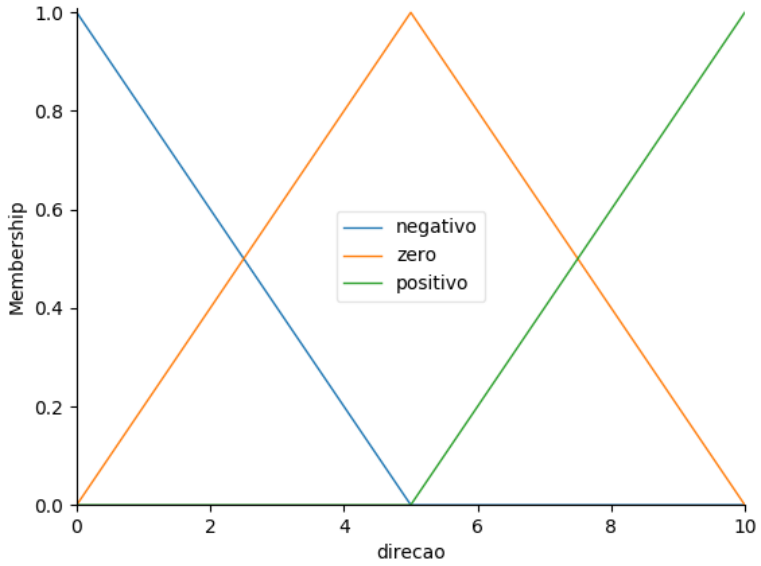

Figura 2: Membership function da variavel de direção para o robô que segue parede

determinar a direção em que ele deve seguir.

A segunda e final versão do segueparede.py é o "followWall2.py", que faz as leituras dos sensores da direita (7 e 8) e utiliza-os para determinar o ângulo entre o robô e a parede, dividindo a diferença da leitura dos sensores pela distância entre eles. Utilizando esse valor como input, juntamente com a distância da frente e a da direita, o robô foi capaz de produzir um movimento mais estável, com curvas apenas nos lugares realmente necessários. A membership function da distância frontal também foi mudada para trapézios, pois concluiu-se que os triângulos anteriores estavam fazendo com que a

distância frontal necessária para começar a virar ficasse muito grande.

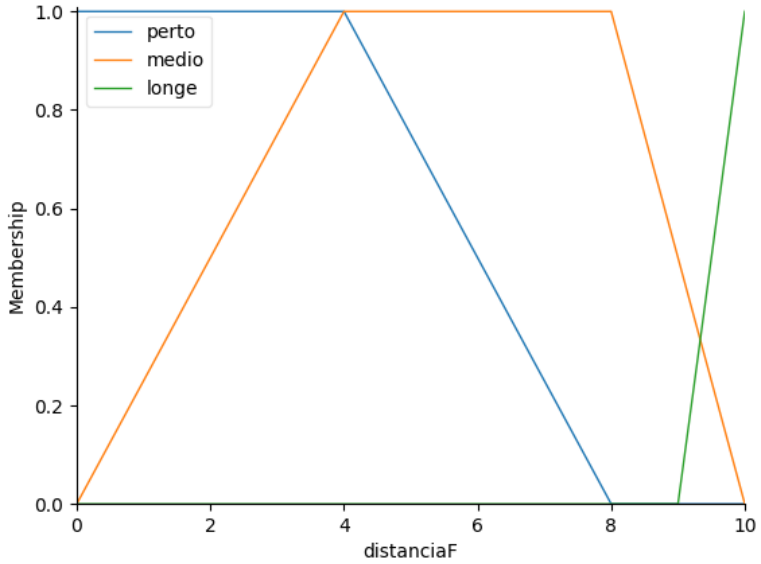

Figura 3: Membership function da variavel de distância frontal para o followWall2.py

O followWall2, por sua vez, precisou de mais regras para funcionar, sendo estas dispostas na tabela 3.

Tabela 3: **Regras adicionais do robô que segue parede**

Ângulo relativo	Frente	Comando
Positivo	-	O robô está indo de encontro com a parede: Virar a esquerda
Zero	-	O robô está em paralelo com a parede: Seguir reto
Negativo	Longe	O robo está se distanciando, mas está longe de uma quina: Virar à direita
Negativo	Medio ou Perto	O robo está se distanciando, mas está próximo de uma quina: Virar à esquerda

O outro comportamento para o P3-DX que foi implementado é o "avoid obstacle". Este comportamento faz com que o robô tente prosseguir em uma direção global definida, enquanto desvia de obstáculos que encontrar no caminho. Por questões de variação, este robô altera sua direção mudando a velocidade das rodas com o uso das funções "robot.set_left_velocity()"e "robot.set_right_velocity()". Utilizando os sensores diagonais 2 e 7 para determinar as distâncias laterais, os sensores frontais 4 e 5 para determinar a distância frontal, e a função "robot.get_current_orientation()"para determinar a orientação global do robô através do ângulo γ de Euler,a velocidade das rodas esquerda e direita é determinada. Suas regras estão dispostas na tabela 4 e são um aprimoramento de regras mais simples para desviar de obstáculos [3].

Tabela 4: **Algumas regras do robô que desvia de obstáculos**

Ângulo global	Frente	Direita	Esquerda	Comando
Negativo	-	-	Longe	O robo está indo para a direita do objetivo: virar a esquerda
Positivo	-	Longe	-	O robo está indo para a esquerda do objetivo: virar a direita
Zero	-	-	-	Direção correta: seguir em frente
-	Longe	-	-	Seguir em frente
-	-	Medio ou Perto	-	Virar a esquerda
-	-	-	Medio ou Perto	Virar a direita

4 Resultados e Discussão

Conforme os valores dos gráficos 1, 2 e 3, os testes iniciais foram realizados. Lembrando que a forma padrão da biblioteca scikit-fuzzy de desfuzzificar os dados é por centroides, a figura 4 representa o comportamento do robô no cenário fornecido pelo site da disciplina:

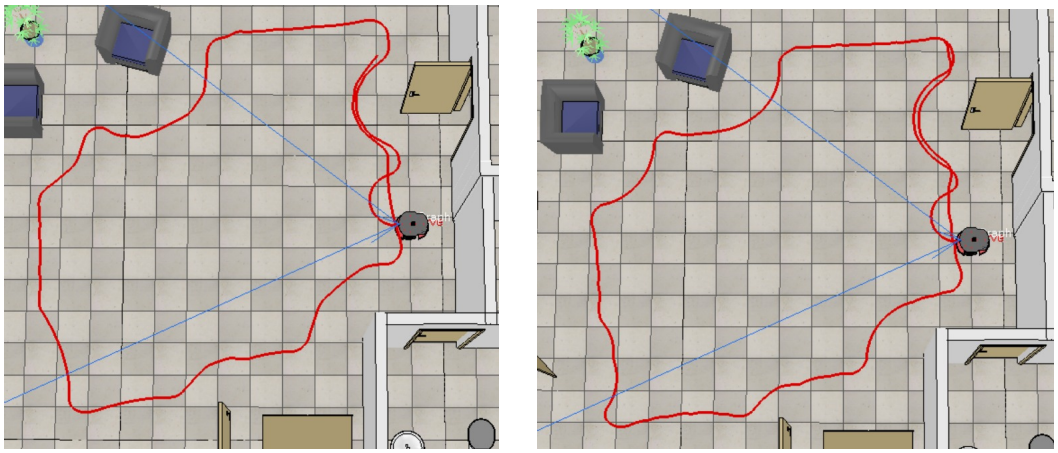

Figura 4: Segue parede em execução, com (a) desfuzzyficação=centroide e (b) desfuzzificação=bisector

Conforme a figura 4, pode-se concluir que, se forem levadas em consideração as variações de cálculo durante o trajeto, o mesmo resultado. Como o modo bisector procura dividir a área sob o gráfico fuzzy na metade, enquanto que o centroide busca o centro de massa, algumas pequenas variações são detectáveis. Porém, para a forma de trapézio escolhida, o resultado dos dois modos será praticamente o mesmo na grande maioria das vezes, acarretando em caminhos similares. Outro resultado que foi analisado é quando a posição inicial do P3-DX é trocada. Esse resultado está na figura 5. Como pode-se notar, o follow wall continua executando da mesma forma independente do lugar de início, contanto que seja uma posição com uma distância razoável para permitir um giro sem bater a lateral.

Por fim, o último teste realizado com o follow wall foi o de mudança do intervalo das variáveis fuzzy "perto", "médio"e "longe"que regulam as distâncias. O valor antigo e o novo estão na figura:

Para realizar esse teste, o robô foi posicionado no mesmo lugar em um cenário sem obstáculos além das paredes, para que seu trajeto pudesse ser analisado com mais precisão. O resultado obtido está na figura 7.:

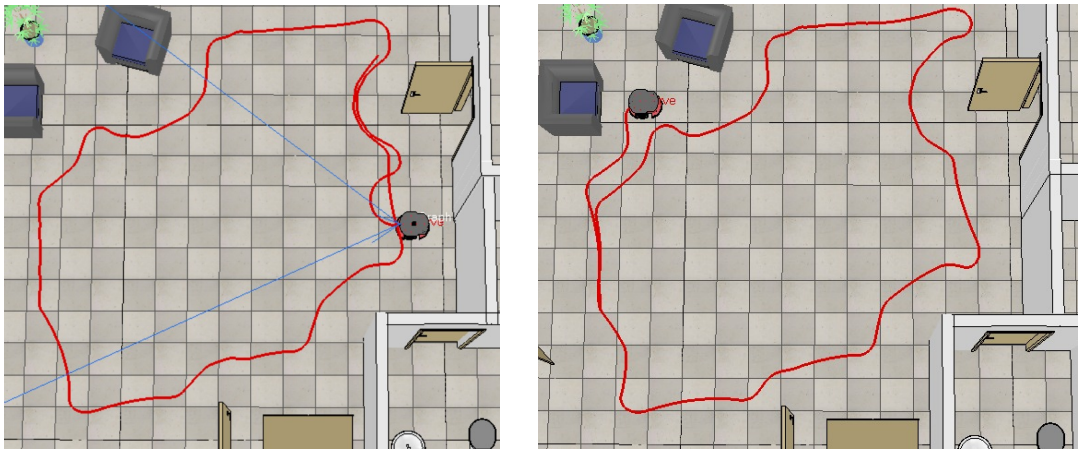

Figura 5: Follow Wall iniciando de dois locais diferentes do mesmo cenário

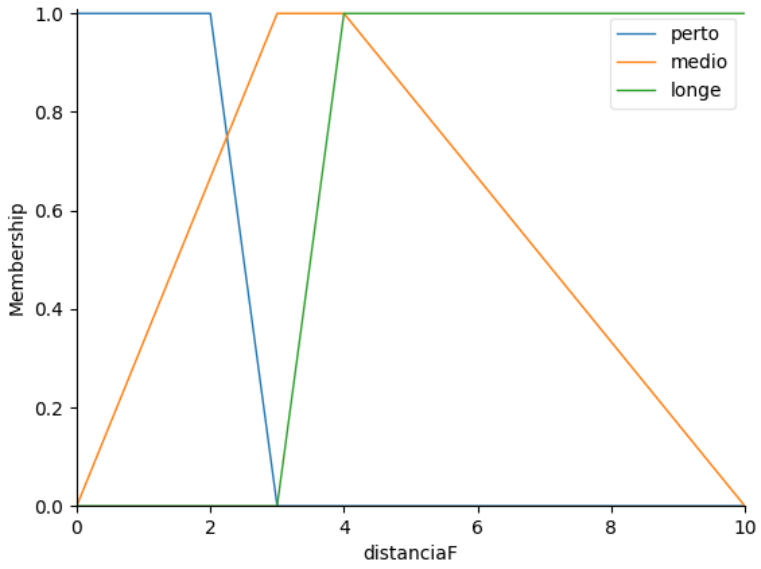

Figura 6: Membership function modificada.

Pode-se perceber que o percurso é praticamente o mesmo, o único fator que realmente foi alterado é o tempo, sendo que o robô com parâmetros modificados demora mais para completar o trajeto. Isso se dá devido à forma como as regras foram definidas, em que o robô anda mais devagar quanto está à distância "média"da parede e, com os parâmetros modificados, isso acontece mais frequentemente.

Quanto ao comportamento de evitar obstáculos, uma mudança foi feita em relação ao Follow Wall: o robô agora gira parado. Antes, as curvas eram feitas ainda em movimento, porém, para evitar obstáculos faz mais sentido girar no lugar para evitar colisões e quinas ao máximo. Alguns exemplos de execução do Avoid Obstacle estão na figura 8.

O resultado mais interessante é o fato de que o Avoid Obstacle se comporta como um Segue Parede quando em situações como a da figura 8b. Além disso, é notável que o script de seguir parede já implementa uma forma de desviar de obstáculos, pois o robô não bate enquanto está percorrendo a parede. No entanto, como o Follow Wall busca sempre ficar à direita do obstáculo, o script Avoid

Figura 7: Segue parede em execução, com (a) variáveis fuzzy originais e (b) modificadas

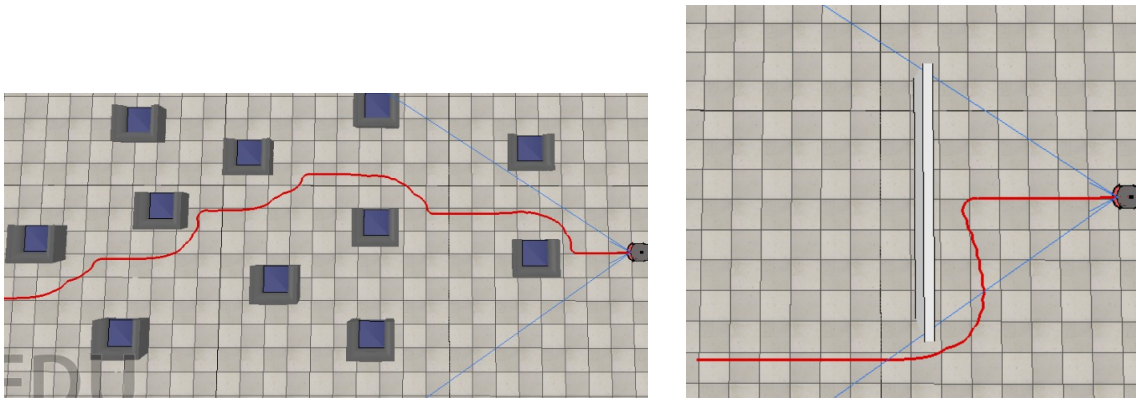

Figura 8: Avoid Obstacle (a) desviando de objetos e (b) contornando uma parede

Obstacle teve de ser criado para suprir essa necessidade e poder realizar curvas para os dois lados livremente.

Além disso, um outro comportamento que poderia ser implementado, mas não era necessário conforme o enunciado, era o GoToGoal. Esse comportamento já está parcialmente implementado no Avoid Obstacle, visto que o P3-DX sempre volta para a mesma direção e, sendo assim, com o uso da função "get_current_position"do arquivo "robot.py"fornecido, a posição (x,y,z) do robô poderia ser adquirida e, com algumas comparações, poderia ser criado. Isso mostra que os três comportamentos estão intrinsecamente ligados, já que para seguir uma parede é necessário evitar obstáculos e para o último, é útil seguir sempre em uma direção. Outro resultado de Avoid Obstacle que mostra a ligação com o segue parede está na figura: 9.

Como já mencionado, em situações como essa o comportamento de evitar obstáculo acaba se assemelhando ao de seguir uma parede. No entanto, como visto na figura 9, o P3-DX consegue seguir a parede pela esquerda, enquanto que o SegueParede busca sempre a direita.

Para a análise do Avoid Obstacle, alguns testes foram realizados, nos moldes do Follow Wall. O primeiro deles está disposto na figura 10, que mostra a diferença entre os parâmetros originais e os modificados do gráfico 6. Nessa figura, pode-se perceber que o caminho percorrido é praticamente o mesmo. No entanto, os parâmetros modificados apresentaram linhas mais irregulares. Isso se deve ao mesmo fato do Follow Wall: o robô está em distância "perto"e "médio"mais frequentemente e, por isso,

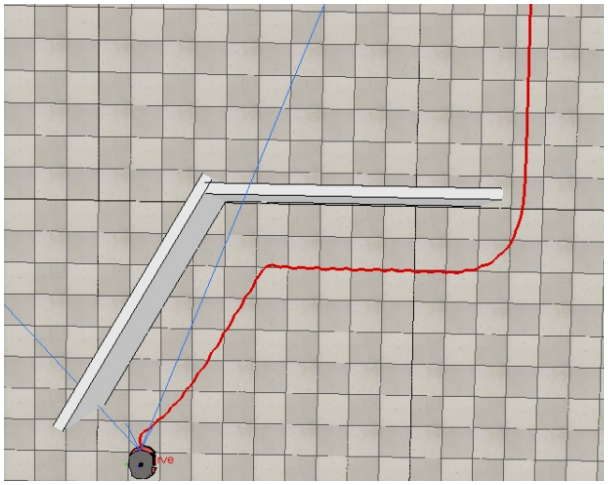

Figura 9: Avoid Obstacle seguindo uma parede

realiza mais curvas, apresentando este comportamento com mais variações. Além disso, o tempo de execução com parâmetros modificados também é maior, já que o P3-DX precisa parar constantemente para realizar as curvas.

Além disso, similarmente ao Follow Wall, a forma de desfuzzyficação foi alterada de centroide para bisector para efeitos de comparação. Os resultados estão na figura 11 e, como pode-se analisar, centroide apresenta linhas mais irregulares. Esse resultado se dá devido à falta de adequação dos parâmetros, já que o mesmo conjunto de variáveis fuzzy não se adequa da mesma maneira nos dois modos. Além disso, como o cenário possuía muitas quinas, e o modo centroide procura o centro dessa área que muda de ângulo várias vezes, faz sentido seu caminho ser menos retilíneo.

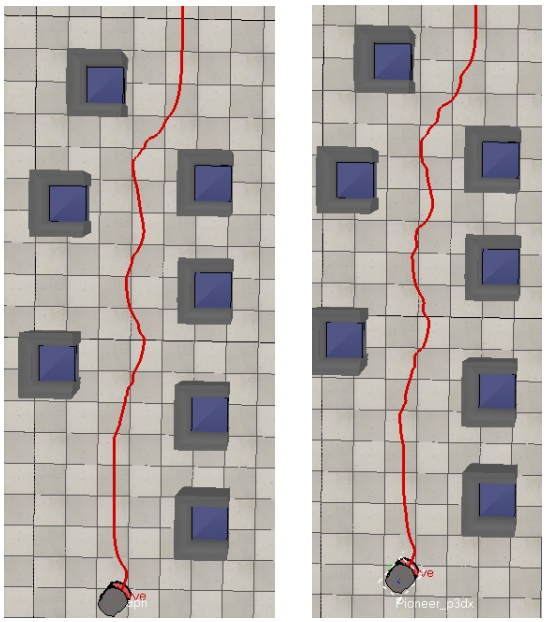

Figura 10: Avoid obstacle com (a) parâmetros originais e (b) modificados

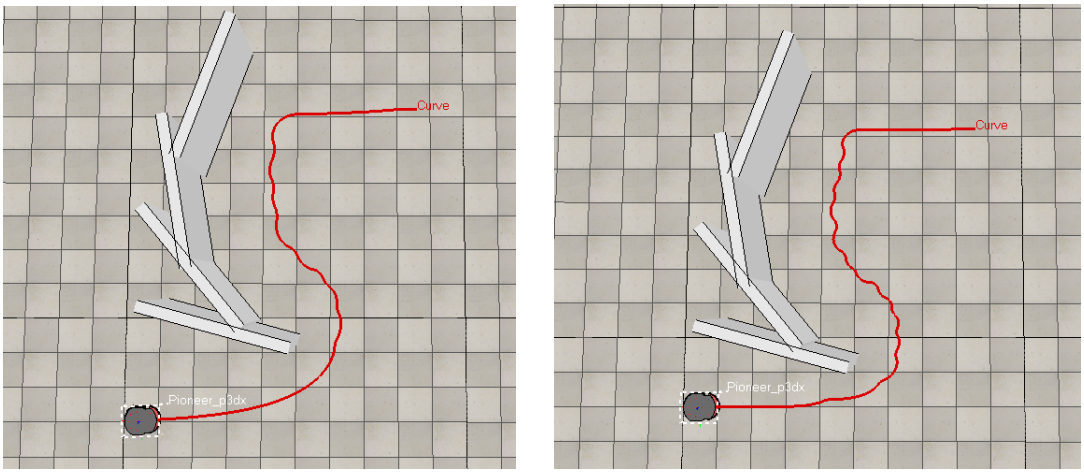

Figura 11: Avoid obstacle com defuzyficação (a)bisector e (b) centroide

Por fim, o último teste iniciou o Avoid Obstacle padrão de duas posições diferentes para analisar seu comportamento. Os resultados estão na figura 12 e pode-se verificar que o P3-DX apresenta o comportamento esperado em ambos os casos.

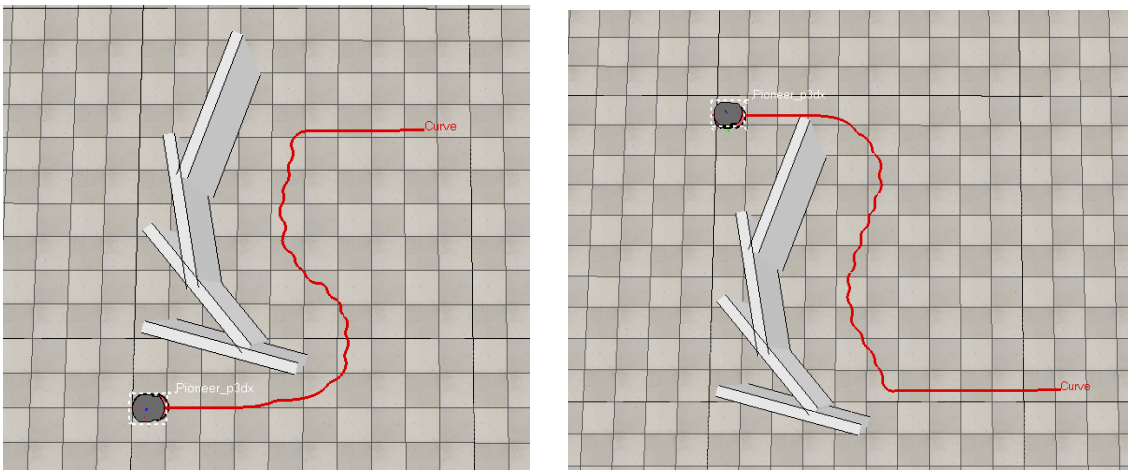

Figura 12: Avoid obstacle com centroide em duas posições distintas

5 Conclusões

Através dos testes realizados com esse projeto, foi possível observar o comportamento do robô sobre diversas situações obtidas através da variação dos métodos de defuzzyficação, variáveis fuzzy de entrada e saída, membership function, regras para as variáveis fuzzy, entre outros.

Foi observado que as membership functions das variáveis fuzzy possuem grande influência no comportamento do robô pois através delas, é possível alterar o comportamento do robô significativamente. Além disso, as regras escolhidas devem ser cuidadosamente pensadas, visto que elas regulam todo o comportamento do robô e correlacionam as variáveis fuzzy de entrada e saída.

Quanto a modificação do método de defuzzyficação, não foram obtidos nenhum resultado importante, já que os dois modos escolhidos coincidem em certos pontos e, por isso, foi a mudança menos significativa se tratando de modificar o comportamento do robô.

Referências

[1] *A lógica fuzzy*, Disponível em https://pt.wikipedia.org/wiki/L%C3%B3gica_difusa 1

[2] *Como construir um robô que segue parede*, Disponível em https://www.allaboutcircuits.com/projects/how-to-build-a-robot-follow-walls/ 1

[3] *Como construir um robô que evita obstáculos*, Disponível em https://www.allaboutcircuits.com/projects/build-your-own-robot-avoiding-obstacles/ 4