MO810 - Trabalho 2

LUÍSA MADEIRA CARDOSO *

*Aluno especial - Mestrado
E-mail: lu.madeira2@gmail.com

***Resumo* – O objetivo deste trabalho é a implementação de um sistema de controle de um robô diferencial executado no simulador V-REP. Foram implementados dois comportamentos de controle *Avoid Obstacle* e *Wall Follow* utilizando sistemas Fuzzy em Python. Além disso a pose do robô foi estimada através da odometria.**

Os controles implementados em Fuzzy mostram-se promissores, mas ainda apresentam falhas. O cálculo da odometria provou-se extremamente vulnerável a erros acumulados.

***Palavras-chave* – V-REP Pioneer AvoidCollision WallFollow Fuzzy**

I. INTRODUÇÃO

Este projeto consiste no desenvolvimento de um sistema de controle para o *Pioneer P3-DX* no simulador *V-REP*. A implementação foi realizada em Python 3.5, com a utilização de algumas bibliotecas como o Scikit-fuzzy e Matplotlib. Os ciclos de atualização de leitura do sensores acontecem por padrão a cada 200ms. O código fonte pode ser obtido em https://github.com/luwood/MO810-vrep-python. As instruções de instalação se encontram no README do projeto.

Este artigo está dividido em três sessões principais, cada uma com sua apresentação e discussão dos resultados.

- Odometria
- Controle: evitar obstáculo
- Controle: seguir parede

II. ODOMETRIA

O cálculo da odometria foi realizado com base na estimativa de velocidade das rodas. A implementação deste componente está na classe *OdometryPoseUpdater*.

A. Rodas

Cada roda possui um *encoder* que provê sua posição angular. Através da coleta temporal desta informação é possível determinar sua velocidade utilizando a seguinte fórmula:

$$V = \frac{\Delta\theta}{\Delta time} R$$

Em que $\Delta\theta$ representa a diferença angular entre posições do *encoder* durante um intervalo de tempo $\Delta time$ e R é o raio da roda. É importante destacar que o cálculo da diferença angular deve levar em conta a orientação do giro e o universo em que os ângulos estão.

A implementação do cálculo de velocidade da roda encontra-se na classe *Wheel*. A orientação do giro é obtida utilizando a hipótese que a diferença angular deve ser sempre menor do que π.

B. Velocidade do Robô diferencial

Dada a velocidade de cada roda, pode-se calcular a velocidade linear e angular do robô através da fórmula:

$$V = \frac{V_r + V_l}{2}$$

$$\omega = \frac{V_r - V_l}{D}$$

Em que V_r é a velocidade da roda direita, V_l é a roda esquerda, D é a distância entre as rodas, V é a velocidade linear e ω é a velocidade angular.

C. Pose

A pose do robô é então calculada

```
x = lastPose.x + (deltaSpace * cos( \
    addDelta(lastPose.orientation, deltaTheta/2))
y = lastPose.y + (deltaSpace * sin( \
    addDelta(lastPose.orientation, deltaTheta/2))
theta = addDelta(lastPose.orientation, \
                 deltaTheta)
```

D. Resultados

O gráfico comparando a posição real do robô e a posição calculada através da odometria pose ser visto na figura 1. É possível observar que o trajeto em linha reta obtido pela odometria é preciso. Porém assim que a primeira curva é realizada a diferença entre as posições começa a divergir. A diferença torna-se maior a cada iteração devido aos erros acumulados. A tabela I mostra a evolução do erro no cálculo da orientação durante um determinado período de teste.

Pode-se concluir que a odometria é um método de estimativa extremamente suscetível a erros acumulados. Para tornar este método viável seria necessário necessário realizar correções no cálculo da orientação. A utilização de uma bússola, por exemplo, poderia auxiliar nesta computação.

III. CONTROLE: EVITAR OBSTÁCULOS

A classe que implementa o comportamento *AvoidCollision* é *FuzzyAvoidObstacle*. A entrada do sistemas são os oito sensores ultrassônicos frontais do *Pioneer*. Todos os sensores são modelados pelas mesmas funções de precedência descritas na figura 2 [1]

As saídas do sistema são a velocidade angular e linear do robô. A figura 3 mostra a modelagem da primeira, na qual a função de defuzificação é dada pela média dos máximos

Tabela I
DIFERENÇA θ EM GRAUS - A CADA 200MS

Real	Odometria	Erro
-0.424	-0.406	0.018
-0.515	-0.488	0.027
-0.783	-0.718	0.065
-1.182	-1.065	0.117
-2.033	-1.818	0.215
-3.048	-2.724	0.324
-4.355	-3.861	0.494
-6.122	-5.416	0.706
-7.843	-6.953	0.89
-9.851	-8.747	1.104
-12.613	-11.227	1.386
-14.864	-13.288	1.576
-18.003	-16.129	1.874
-22.479	-20.084	2.395

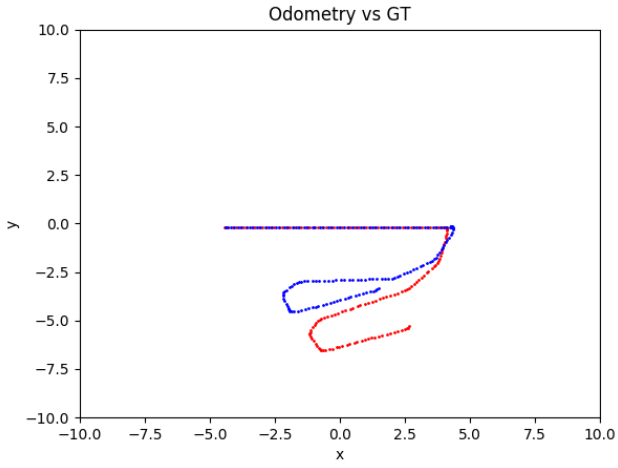

Figura 1. Odometria: Linha vermelha, Ground Thruth: Linha azul - Exemplo com script rodando algoritmo de Braitenberg

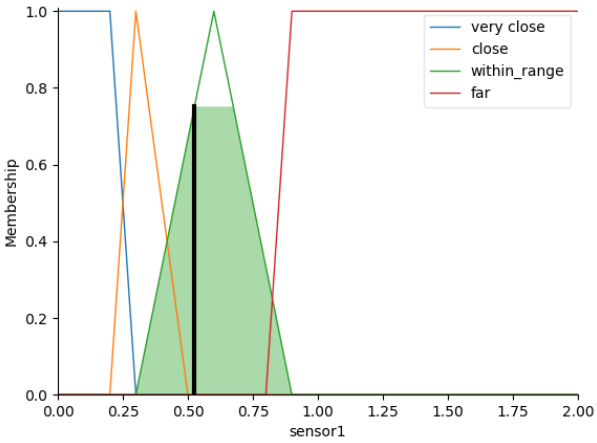

Figura 2. Modelagem sensor de proximidade - Exemplo de ativação de região

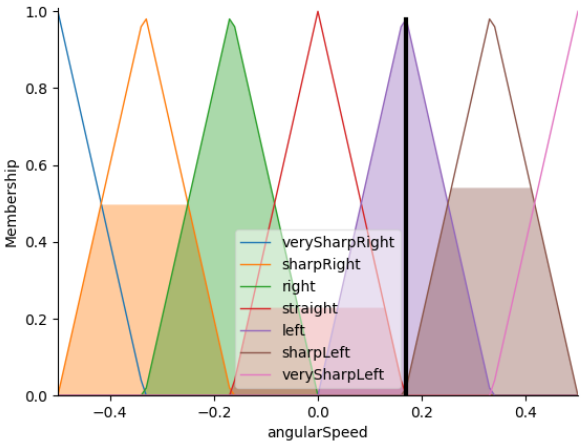

Figura 3. Modelagem velocidade angular - Exemplo de ativação

(*mom*). A velocidade linear é modelada de acordo com a figura 4 e o método *centróide* é utilizado na defuzificação.

O conjunto de regras é composto por 26 declarações, existindo pelo menos 3 por sensor. Os sensores frontais, no entanto, são os fatores mais decisivos para a saída do sistema. Com o intuito de evitar situações em que pode existir o equilíbrio, os sensores frontais sempre levam a ativação da saída "vire a esquerda".

A. Resultados

Um exemplo do resultado obtido pode ser visto na figura 5. O robô nunca colide com nenhum obstáculo e é capaz de passar por ambientes em que existe uma disposição mais complicada dos objetos, por exemplo o canto onde residem duas cadeiras e uma planta. No entanto, pode-se perceber uma forte tendência de curva a esquerda e que outros ambientes nunca são alcançados.

IV. CONTROLE: SEGUIR PAREDE

A classe que implementa o comportamento *WallFollow* é *FuzzyWallFollower*. O controle tem por objetivo manter uma distância de 30cm da parede direita. A entrada do sistemas são apenas dois sensores do *Pionner* modelados de acordo com a figura 6.

As regras deste sistema são extremamente simples e podem ser descritas como:

- Se existe algo na frente, vire a esquerda.
- Se a distância do lado direito é grande, vire a direita.
- Se a distância do lado direito é pequena, vire a esquerda.

As saídas do sistema são a velocidade angular - figura 7 - e a velocidade linear - figura 8.

A. Resultados

O resultado obtido pode ser visto na figura 9. Note que todas as portas foram fechadas para simplificar o ambiente. O

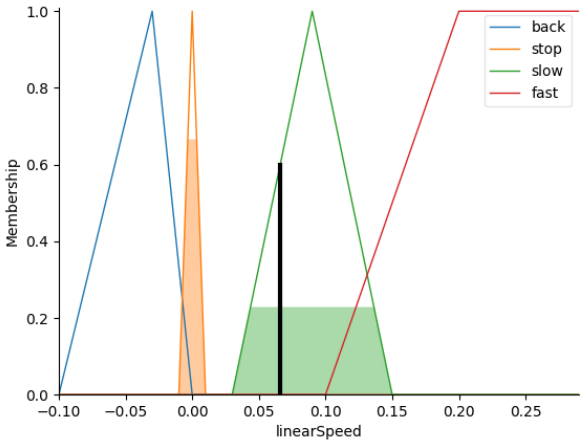

Figura 4. Modelagem velocidade linear - Exemplo de ativação

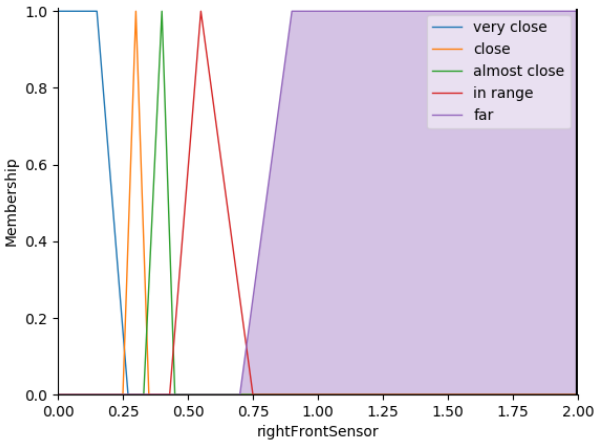

Figura 6. Modelagem sensor frontal - Exemplo de ativação

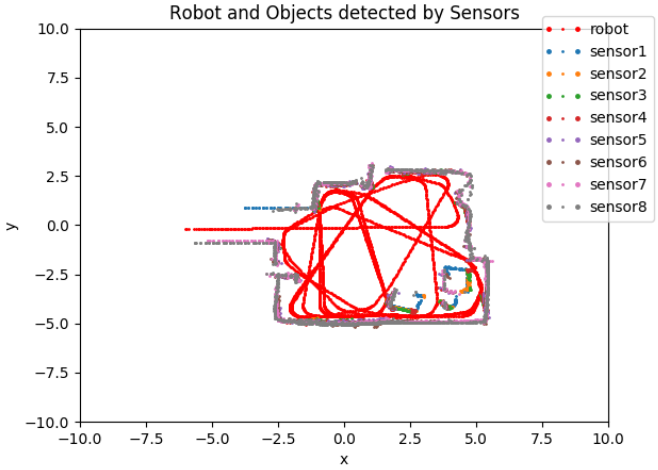

Figura 5. Modelagem velocidade linear - Exemplo de ativação

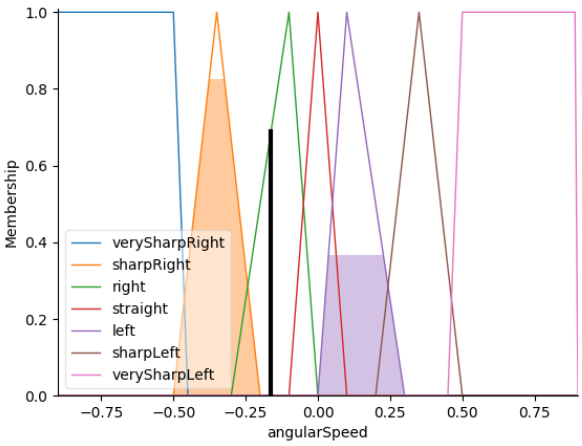

Figura 7. Modelagem velocidade angular - Exemplo de ativação

robô se mantém bem próximo a parede. Entretanto, ele não é capaz de lidar com situações nas quais existe um obstáculo em seu caminho. A região destacada por um circulo azul indica a colisão do robô com um armário. Isso acontece porque apesar de inicialmente o sensor frontal indicar que existe um objeto próximo e que uma curva a esquerda é necessária, assim que o objeto deixa de ser percebido pelo sensor frontal entram em ação as regras relativas a distância da parede. No caso, a distância do armário é capturada na diagonal e é maior do que limite aceitável, levando o robô a tentar se aproximar. No momento em que a curva para a direita é ativada, o robô colide com a quina do móvel.

A figura 10 mostra uma versão mais relaxada das distâncias na modelagem dos sensores. O robô fica então muito mais distante da parede e realiza curvas muito abertas, porém é capaz de evitar obstáculos como os armários. Outras abordagens também foram exploradas, como a adição de uma variável de entrada que corresponde a diferença da distância da parede entre iterações, porém elas não resultaram em melhorias relevantes e portanto não serão abordadas nesta sessão.

Pode-se concluir que apenas os três sensores e a modelagem Fuzzy não foram suficientes para fazer com que o robô tivesse um comportamento de seguir a parede sem nenhum problema. O robô está vulnerável a colidir com qualquer tipo de objeto que não é capturado pelo seu sensor frontal. Além disso, objetos no meio do caminho também podem levar a uma colisão indesejada. Para resolver o segundo problema o modelo poderia ser refinado para realizar curvas rentes aos cantos. O primeiro problema, entretanto, só poderia ser resolvido com a adição de um outro modelo, como o *AvoidCollision* [2].

+————————+

Referências

[1] H. R. Beom and H. S. Cho, "A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning," *IEEE transactions on Systems, Man, and Cybernetics*, vol. 25, no. 3, pp. 464–477, 1995. 1

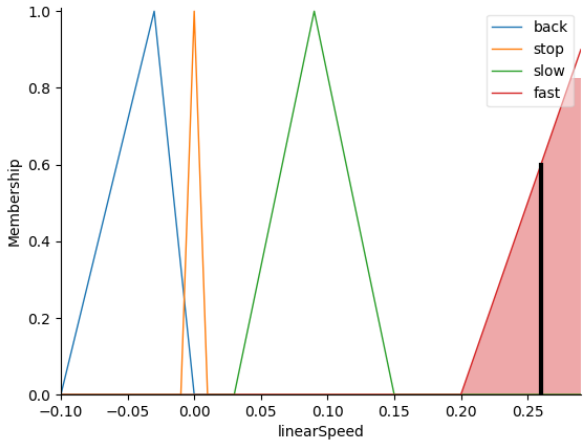

Figura 8. Modelagem velocidade linear - Exemplo de ativação

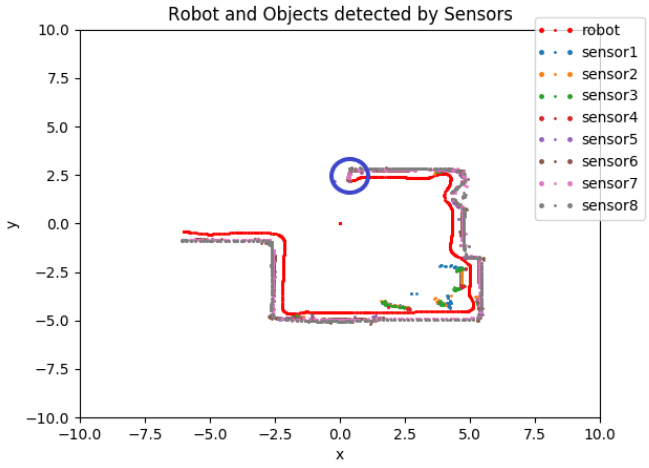

Figura 9. WallFollow - região circulada indica colisão

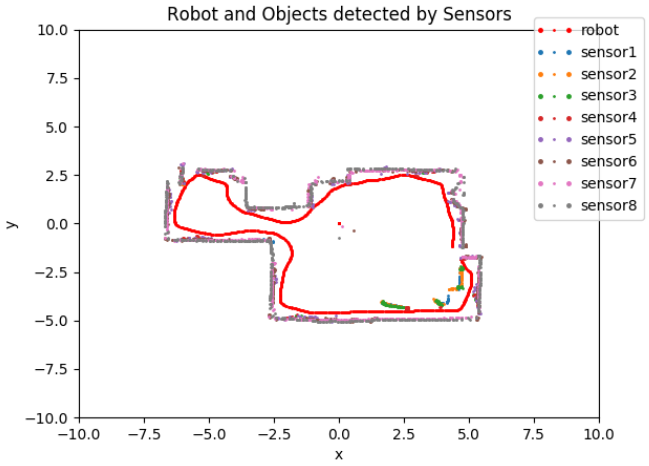

Figura 10. WallFollow - modelagem relaxada

[2] H. Omrane, M. S. Masmoudi, and M. Masmoudi, "Fuzzy logic based control for autonomous mobile robot navigation," *Computational Intelligence and Neuroscience*, vol. 2016, 2016. 3