

# MO810 - Trabalho 2

MATEUS CORADINI SANTOS \*

\*Prof (a): Profa. Dra. Esther Luna Colombini

Aluno especial - Mestrado

E-mail: mateuscoradini@gmail.com

**Resumo** – O objetivo inicial deste trabalho são os testes das teorias aplicadas em sala de aula, colocando em prática a implementação do sistema de controle diferencial para um robô móvel executado em um ambiente simulado, no caso, usamos o software V-REP. Foram implementados dois comportamentos de controle *Avoid Obstacle* e *Wall Follow* utilizando sistemas Fuzzy em Java e estimativa de Pose baseada em odometria. As classes de controle para os algoritmos em Fuzzy, foram implementadas a modo de fazer somente testes básicos de implementação e não houve um trabalho de modelagem específica para obter melhores resultados, pois o objetivo do trabalho era entender e aprender a utilizar a lógica em função das tarefas executadas anteriormente como malha aberta. A odometria não está correta, e ainda necessita de tratamento, pois não está limpando os erros acumulados. Os algoritmos de fuzzy para *WallFollow* e *AvoidObstacle* se encontram sem otimização no momento.

**Palavras-chave** – V-REP Pioneer AvoidObstacleCollision Wall-Follow Fuzzy

## I. INTRODUÇÃO

Este projeto consiste no desenvolvimento de um sistema com inteligência necessária para o controle do modelo de robô *Pioneer P3-DX* no simulador *V-REP*. A implementação foi realizada em Java versão 1.7, com a utilização de algumas bibliotecas como *JFuzzyLite*. Os ciclos de leitura e atuação dos sensores são controlados por threads na qual utilizamos um tempo de 300ms para atuação. O código fonte pode ser obtido em <https://github.com/mateuscoradini/VRepSimulator/V2>. As instruções de instalação se encontram na página principal, mapeados para MAC-OS e Windows.

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

- Odometria - Estimativa de Pose
- Controle: Avoid Obstacle (Evitar obstáculo)
- Controle: Wall Follow (Seguir parede)

## II. ODOMETRIA

A estimativa de pose, foi comparada inicialmente ao GroundTruth (posição real do robô), para que medisse as falhas em acumulo, ocorridas durante o tempo de execução do trajeto do robô. Para isso, a implementação foi realizada na classe *PositionLocator*. Conforme velocidade das rodas, calculados pela fórmula citada em sala de aula, estima-se a localização aproximada no vetor x,y,z, a localização do robô.

### A. Componente: Rodas

Cada roda, possui um encoder, que nos informa o valor da sua posição angular. Com as informações desses encoders, é possível determinar a velocidade do robô através da 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  $R$  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 estão sendo determinados os ângulos.

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

### B. Velocidade do Robô diferencial

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

$$V = \frac{V_r + V_l}{2} \quad \omega = \frac{V_r - V_l}{2L}$$

Figura 1. Fórmula: Velocidade Linear (V), (W) Velocidade Rotacional ao longo do eixo

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

### C. Pose

A pose do robô é então calculada

```
float x = (float) (robotLastPosition.getX()
+ (deltaSpace * Math.cos(addDelta(robotLastPosition.getOrientation(), deltaTheta / 2))));
float y = (float) (robotLastPosition.getY()
+ (deltaSpace * Math.sin(addDelta(robotLastPosition.getOrientation(), deltaTheta / 2))));
float orientation = addDelta(robotLastPosition.getOrientation(), deltaTheta);
```

Figura 2. Classe PositionLocator: odometry

#### D. Resultados dos testes

Não foi possível gerar o gráfico de coordenadas das linhas de comparação, então determinei uma saída de console comparativa, somente para verificar a implementação e efetuar teste de que se havia erro acumulado. A saída abaixo foi gerada ao usar o algoritmo para controle de Wall Follow, após alguns algum tempo de execução:

```
GroundTruth {0.49356467 X, -1.8204131 Y, -1.986935 O}
Odometry {0.46356467 X, -1.7204131 Y, -1.986935 O}
GroundTruth {0.48098224 X, -1.7404131 Y, -2.0268958 O}
Odometry {0.45356467 X, -1.7404131 Y, -2.0268958 O}
GroundTruth {0.46728283 X, -1.7625132 Y, -2.0673726 O}
Odometry {0.43356467 X, -1.7104131 Y, -2.0673726 O}
```

Figura 3. Saída do console em vetor x,y, orientação, da PoseLocator

Não consegui concluir com uma dimensão exata o acumulo erro crítico. Gostaria de gerar esses dados a partir de um gráfico e conseguir tabelar o acumulo de erro. Para a próxima melhoria, irei implementar alguma alternativa para ter visualmente o caminho de cada posição percorrida e estimada do robô. Logo se nota uma diferença após a execução por um minuto a minuto de execução, após a rotação do robô, o acumulo de diferença da odometria e do GroundTruth da posição real do robo. Para a implementação ser finalizada, necessita de uma correção de orientação usando-se algum outro encoder anexado ao robô Pioneer.

#### III. ROBOT FUZZY CONTROL: WALL FOLLOW - SEGUIR PAREDE

A implementação da lógica fuzzy e do comportamento de seguir parede foi implementado na classe *FuzzyWallFollower*. O controle tem por objetivo manter uma distância de 30cm da parede direita. A entrada do sistema é apenas dois sensores do Pioneer modelados de acordo com a figura 4. A figura abaixo representa uma das modelagens usadas no sensor para o controle de wall following. Pode-se ver todas as regras na classe (*FuzzyWallFollower*). As figuras abaixo somente exemplificam a modelagem implementadas nas regras de entrada e saída e os exemplos de cada ativação.

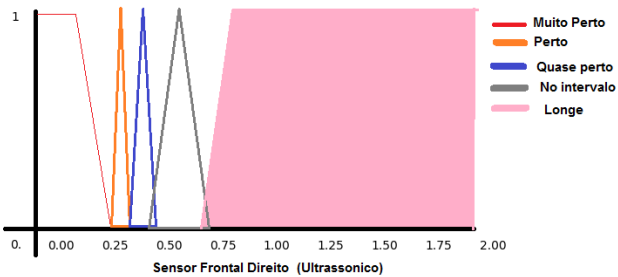


Figura 4. Modelagem sensor frontal

As saídas do sistema são a velocidade angular - figura 5 - e a velocidade linear - figura 6.

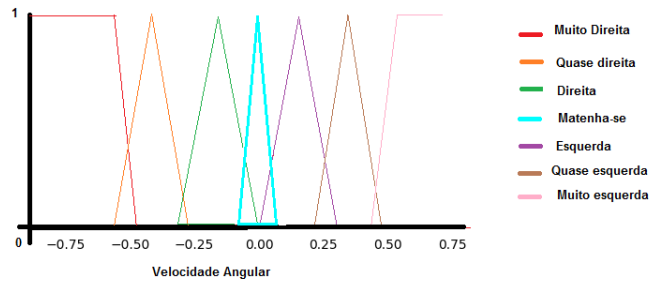


Figura 5. Modelagem Fuzzy da velocidade angular

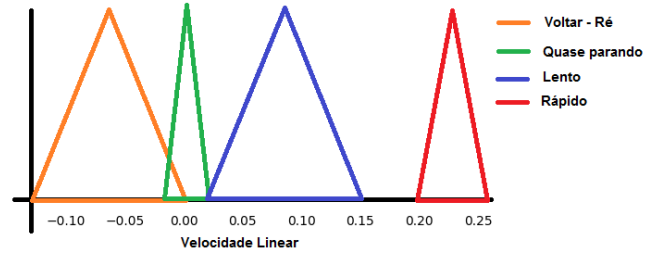


Figura 6. Modelagem velocidade linear

#### A. Resultados

Os resultados se mostraram se promissores quando-se usou o fuzzy com uma modelagem correta, pode-se obter precisão nos comportamentos de execução dos algoritmos mencionados acima. Os resultados descritos nesse trabalho apenas foram exemplificações das implementações sugeridas ao decorrer do trabalho. Para cada implementação, obtive um teste básico para avaliar o aprendizado da orientação passada em sala de aula. Porém ainda há alguns comportamentos a serem implementados, e o ajuste fino dessa modelagem necessita ser realizado caso queira que o comportamento do robô, se adapte-se ao ambiente sugerido. Não consegui evitar alguns problemas como as passagens pelos obstáculos do cenário, porém consegui entender a lógica de implementação dos algoritmos citados e consegui exemplificá-los na linguagem de programação Java. Foram usados alguns artigos como leitura base de conhecimento das implementações realizadas ao longo da última década. Artigos [1], [2].

+-----+

#### REFERÊNCIAS

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