Localização Probabilística de um Robô usando Pontos de Referência Redundantes

Departamento de Engenharia Mecânica
Mestrado em Robótica e Sistemas Inteligentes
Robótica Móvel - Vitor Santos e Nuno Lau
Universidade de Aveiro

Filipe André Seabra Gonçalves - 98083

Abril de 2024

1 Introdução

Este relatório tem por objetivo explicar a abordagem e o código desenvolvido na criação de uma função em *Matlab* que faz uma previsão da posição de um robô usando **Extended Kalman Filter** e diversos pontos de referência, *landmarks*, dados pela função **BeaconDetection**.

Is dois robôs usados foram o *Differencial Drive*, **DD**, e o *Tricycle*, **TRI**.

Se não forem dados argumentos de entrada à função, são usados os valores defeito da função, vistos na Tabela 1

Número de Landmarks	N	4
Intervalo de Tempo	Dt	1
Raio das Rodas	r	0.15
Distância entre as Rodas	L	1
Incerteza na Velocidade Linear	Vn	0.1
Incerteza na Velocidade Angular	Wn	0.1
Velocidade Média Desejável	V	5

Table 1: Valores defeitos dos argumentos de entrada da função

2 Trajetória

Para calcular a trajetória a ser feita pelo robô móvel é primeiramente preciso identificar os **landmarks** existentes. Para tal é usada a função **BeaconDetetion**, criada pelo professor, para encontrar os N **landmarks**. Depois é calculado o número de pontos que deve existir entre cada dois **landmarks**, contando com a posição inicial como sendo o primeiro **landmark**, de acordo com a velocidade média desejável do robô e o intervalo de tempo entre cada ponto. Por fim são calculados as corodenadas em X da trajetória entre os dois **landmarks**.

Como as posições estão a ser calculadas, o robõ iria fazer um movimento linear entre todos os pontos de referência. No entanto, não é essa a trajetória que o robô deveria fazer. Assim sendo, é usada a função **pchip** do *Matlab* que, dadas as coordenadas de X e as coordenadas de Y dos **landmarks** e os intervalos de tempo antes calculados, recebemos as posições em Y esperadas da trajetória.

Podemos ver o resultado da não utilização da função **pchip** e a sua utilização, nas Figuras 1 e 2 respetivamente.

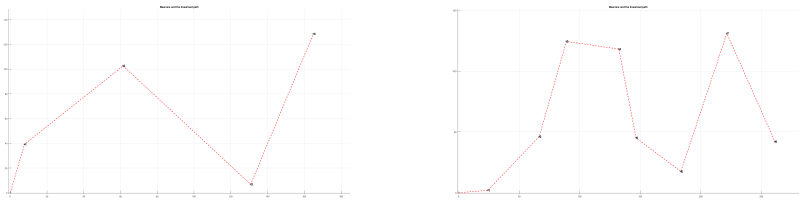

Imagem 1: Trajetória Linear - 4 e 8 Landmarks

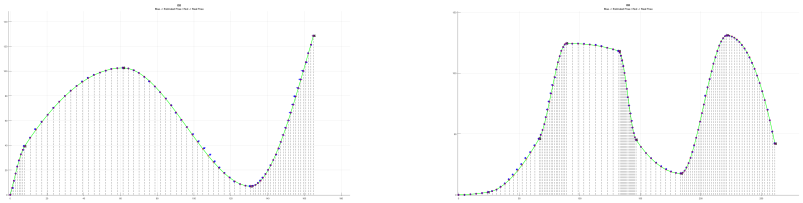

Imagem 2: Trajetória Pchip - 4 e 8 Landmarks

3 Posicionamento do Robô

Com os pontos da trajetória calculados, para obter as posições e orientações do robô em cada ponto do tempo foi necessário fazer um ciclo que começa no tempo inicial e acaba no tempo exatamente antes do final.

Para cada segmento de tempo calculou-se primeiro a orientação que o robô deve ter de acordo com o ponto da trajetória onde se encontra e o seguinte, em segundo lugar usou-se a cinemática inversa de cada tipo de robô, usando o raio e a distância entre rodas, bem como a diferença de coordenadas entre o ponto atual e o seguinte, e o intervalo de tempo entre movimentos, e por fim, antes de adicionar à lista de controlo, calculou-se a velocidade em X e a velocidade em Y, bem como a sua velocidade angular.

Cada linha da lista de controlo representa um movimento do robô, quer este seja apenas rotação, como apenas movimento linear, ou ambos. No caso deste projeto, foi feita a separação entre movimento angular e movimento linear para melhor compreensão.

Para a cinemática inversa dos robôs foram usados as seguintes fórmulas:

- DD (*Differencial Drive*):
  No caso de estar na posição inicial ([0, 0]):

$$\theta = \arctan(Yn, Xn)$$
$$W = \theta / t$$
$$VR = W * L/2 = -VL$$

  No caso de não estar:

$$VL = VR = \|Y\vec{X}\|;$$

- TRI (*Tricycle*):

$$w1 = V * \cos(\theta)/r - \sin(\theta)$$
$$aw2 = \theta / L$$

4 Filtro de Kalman Extendido - EKF

O Filtro de Kalman Estendido (**EKF**) é uma ferramenta crucial na robótica móvel, especialmente no contexto de localização e navegação de robôs. O **EKF** é uma extensão do Filtro de Kalman, desenhado para lidar com sistemas não lineares, uma característica comum nos modelos de robôs móveis.

O **EKF** proporciona uma maneira de prever o estado de um sistema dinâmico a partir de medições sensoriais, que normalmente contém ruído. No caso da robótica móvel e deste projeto, o estado é composto pela posição do robô em duas dimensões e a sua orientação.

O processo de funcionamento do EKF pode ser dividido em duas fases:

- A predição: é usado o modelo de movimento do robô para prever o próximo estado, baseado no estado atual e nas ações de controle aplicadas, que neste caso é a velocidade linear e angular do robô, antes calculadas de acordo com a velocidade méda desejada e com o período de tempo entre medições. Esta etapa irá projetar o estado futuro que será corrigido com novas medições.

- A atualização: o EKF ajusta o estado previsto usando as novas medições dos sensores. A atualização é feita de modo a minimizar a diferença entre as previsões e as medições reais, de acordo com as incertezas aplicadas aos sensores usados.

No nosso caso, a predição é feita para cada momento da trajetória, usando o modelo de movimento do robô com as velocidades linear e angular calculadas na secção 3.

Em relação á atualização, para cada posição calculada, e usando a função **BeaconDetection**, encontramos os dois beacons mais próximos à posição estimada do robô, para fazer a triangulação, e atualizamos o *pose* (posição em duas

dimensões e orientação) previsto com as medidas do modelo de sensores. Por fim, aplicamos o **EKF** na posição estimada com todos os valores calculados.

Deste modo, o movimento do robô **DD**, pode ser visto na Figura 3:

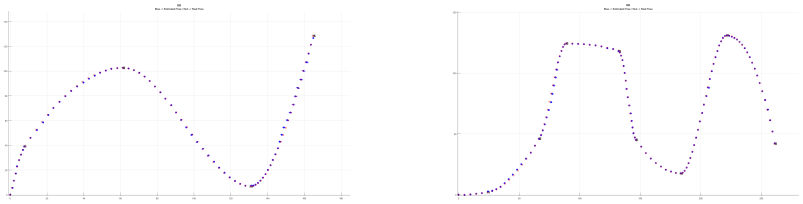

Imagem 3: Movimento Final - Robô DD - 4 e 8 Landmarks

E o movimento do robô **TRI**, pode ser visto na Figura 4:

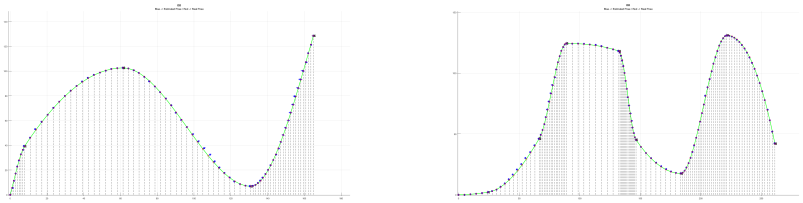

Imagem 4: Movimento Final - Robô TRI - 4 e 8 Landmarks

5 Conclusão

Concluindo, foram cumpridos todos os requisitos do projeto, desde o cálculo da trajetória com o **pchip**, até à criação dos três ficheiros de texto com as as poses, para o ficheiro *loc_98083.txt*, e as velocidades linear e angular de cada ponto da trajetória para os dois tipos diferentes de robôs, *DD_98083.txt* e *TRI_98083.txt*.

Qualquer problema que tenha com a submissão de código, tem o link para o github aqui: `https://github.com/FlipGoncalves/RobotProbabilisticLocalization`.