MO810 - Trabalho Final

LUÍSA MADEIRA CARDOSO *

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

***Resumo* – O objetivo deste trabalho é a implementação de um sistema de localização para um robô móvel utilizando o cálculo de distância com uma base. O filtro Kalman extendido foi empregado para combinar a estimativa provida pela odometria com a observação da localização da base. Os resultados obtidos apenas com a odometria são comparados aos resultados que utilizam o filtro. Além disso, também foram realizados experimentos com mais de uma base. O filtro de Kalman provou-se uma técnica eficiente para computar a localização do robô, especialmente quando mais de uma base é utilizada.**

***Palavras-chave* – V-REP Pioneer Localização KF EKF**

I. INTRODUÇÃO

Este projeto se propõe a realizar a localização de um robô diferencial utilizando uma base, odometria e o filtro de Kalman. Parte-se do pressuposto que o cálculo de distância com a base é dado.

Todas as simulações expostas foram realizadas no simulador *V-REP* utilizando o *Pioneer P3-DX*. A implementação foi feita em Python 3.5, com a utilização de algumas bibliotecas como Numpy 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:

- Odometria
- Localizando a Base
- Filtro de Kalman

II. ODOMETRIA

O cálculo da odometria é realizado com base na estimativa de velocidade das 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.

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.

A pose do robô no momento t depende da pose anterior, em $t-1$, e pode ser calculada através das equações:

$$x_t = x_{t-1} + (\Delta s * cos(\theta_{t-1} + \frac{\Delta\theta}{2}))$$

$$y_t = y_{t-1} + (\Delta s * sin(\theta_{t-1} + \frac{\Delta\theta}{2}))$$

$$\theta = \theta_{t-1} + \Delta\theta$$

A. Implementação

A implementação do cálculo da odomoetria é feita pela classe *OdometryPoseUpdater*. Para fins práticos a pose inicial do robô é obtida com a leitura do *Ground Truth*. O cálculo da velocidade da roda encontra-se em uma classes distinta chamada *Wheel*. A orientação do giro é obtida utilizando a hipótese que a diferença angular deve ser sempre menor do que π.

III. LOCALIZANDO A BASE

A ideia inicial deste projeto era permitir que a localização do robô fosse obtida a partir da comunicação com uma base. O princípio seria semelhante a tecnologia utilizada no StarGazer (HagiSonic): o robô envia um sinal e a base o reflete. Deste modo é possível determinar a distância entre os dois objetos.

A. Teoria

Através da distância de um único ponto, é impossível determinar sua localização precisa. Considerando o sistema local de coordenadas do robô, pode-se ver na figura 1 que a base poderia estar em qualquer ponto do círculo determinado pela distância calculada entre o sensor e a base. Portanto, são necessários mais sensores no robô para determinar a localização da base.

Com três sensores é possível determinar a posição da base através da resolução de um sistema linear de equações. Na figura 2 podemos ver que o círculo que parte de cada sensor, se intersecta em um único ponto.

A equação de cada uma das circunferências pode ser descrita da seguinte forma:

$$r_1^2 = (x - x_1)^2 + (y - y_1)^2 \quad (1)$$

$$r_2^2 = (x - x_2)^2 + (y - y_2)^2 \quad (2)$$

$$r_3^2 = (x - x_3)^2 + (y - y_3)^2 \quad (3)$$

Para encontrar o ponto de intersecção entre as três circunferências, é necessário encontrar os valores de x e y combinando as três equações quadráticas em um sistema de duas equações lineares. Subtraindo (2) de (1) e (3) de (1):

$$2x(x_2-x_1)^2+2y(y_2-y_1)^2+(x_1^2-x_2^2)+(y_1^2-y_2^2)-(r_1^2-r_2^2) = 0$$

$$2x(x_3-x_1)^2+2y(y_3-y_1)^2+(x_1^2-x_3^2)+(y_1^2-y_3^2)-(r_1^2-r_3^2) = 0$$

A solução do sistema é a localização da base considerando o sistema de coordenadas local do robô.

B. Transformação de coordenadas

Para implementação do filtro de Kalman utilizado neste trabalho é necessário identificar a posição da base no sistema de coordenadas globais. A transformação do sistema de coordenadas locais do robô para o sistema global é dado pela rotação (4) seguida da translação(5) do ponto(x, y), no qual dx, dy e α são dados pela pose do robô.

$$\begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (4)$$

$$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (5)$$

C. Implementação

O robô possui três transceptores em seu topo que estão dispostos como mostrado na figura 2. A base se encontra nas coordenadas $(0, 0)$ do sistema global de referência. A distância entre os tranceptores e a base é calculada utilizando o módulo de cálculo de distâncias provido pelo simulador V-REP. É importante ressaltar que este cálculo de distâncias em uma simulação mais verossímil precisaria ser implementado.

A implementação do cálculo de intersecção das três circunferências é o método *calculatePoint* no módulo *AngleUniverse*. Ele é utilizado pela classe *BaseDetector* para calcular a posição da base dadas as distâncias obtidas dos transceptores. A base é representada pela classe *DetectedBase* que possui o método *getAbsolutePosition* que realiza a transformação das coordenadas locais para as coordenadas globais dada uma determinada pose.

IV. Filtro de Kalman

O filtro de Kalman (KF) é uma implementação de filtros *Bayesianos* que realiza remoção de ruídos e predição de valores num sistema de estados contínuos. O interessante desta técnica é sua capacidade de combinar diferentes estimativas e suas respectivas covariâncias, computando uma distribuição Gaussiana baseada apenas em estados anteriores.

Por definição, o KF trabalha com probabilidades lineares. Sua versão extendida (EFK) trabalha com a hipótese de as funções que modelam as probabilidades não são lineares.

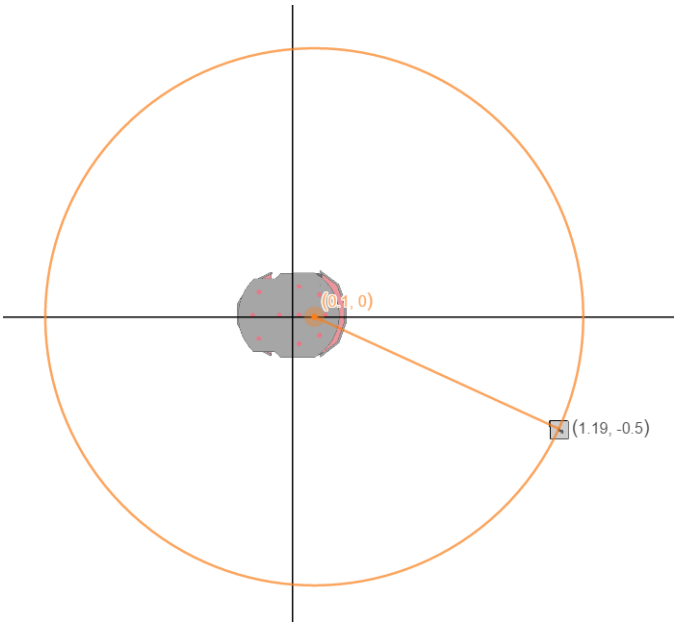

Figura 1. Robô com apenas um sensor de distância da base

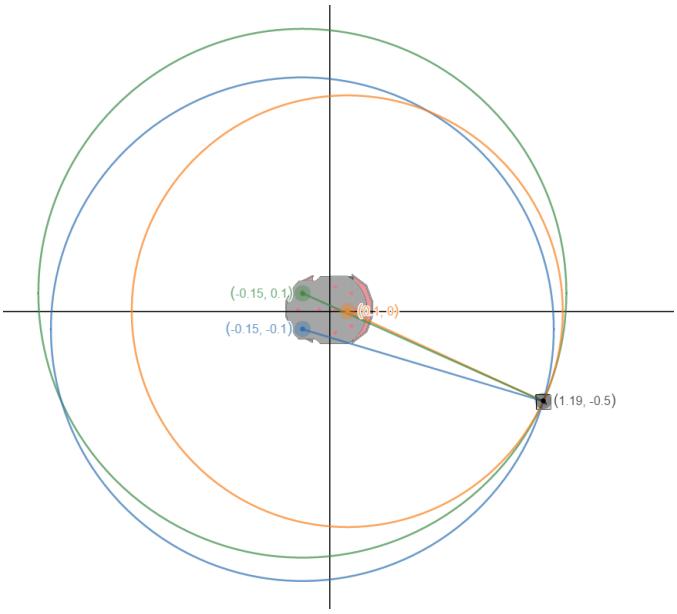

Figura 2. Robô três sensores de distância da base

Como um robô tipicamente pode realizar uma trajetória circular, o modelo mais indicado é o EKF.

A lógica do EKF está descrita pelo algoritmo 1, onde $\bar{\mu_t}$ pode ser entendido como a estimativa do estado no tempo t, μ_{t-1} o cálculo do estado no tempo $t-1$, Σ_{t-1} a covariância calculada em $t-1$ e z_t são as observações no tempo t.

A. Localização com EKF

Esta sessão é dedicada a explicar como o filtro de Kalman Extendido pode ser utilizado para realizar a localização do

Algorithm 1 Extended Kalman filter $\bar{\mu}_t, \mu_{t-1}, \Sigma_{t-1}, \Sigma_{\Delta t}, z_t$

$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-}1$
$\mu_t = \bar{\mu}_t + K_t(\bar{z}_t - z_t)$
$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
return μ_t, Σ_t

robô com o auxílio da detecção de *landmarks*. As fórmulas utilizadas foram primariamente retiradas da tabela 7.2 do livro *Probabilistic Robotics*[1]. Como suporte também foi utilizada uma apresentação realizada em 2014[2].

O objetivo da utilização do filtro é a melhoria na cálculo da pose do robô. Portanto, na primeira linha do algoritmo 1, μ_t representa a pose do robô no instante de tempo t . A estimativa de μ_t, $\bar{\mu}_t$, é dada pelo computação da odometria.

A primeira linha do algoritmo pode ser entendida como o modelo de erro da odometria. Portanto, foi acrescentado mais um fator em sua composição. Sua forma final é dada pela equação:

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + V_t \Sigma_{\Delta t} V_t^T + R_t$$

Os valores de G_t, $Sigma_{\Delta t}$, V_t e R_t estão descritos abaixo:

$$\beta_t = \theta_{t-1} + \frac{\Delta\theta_t}{2}$$

$$G_t = \begin{bmatrix} 1 & 0 & -\Delta s * sin(\beta_t) \\ 0 & 1 & \Delta s * cos(\beta_t) \\ 0 & 0 & 1 \end{bmatrix}$$

$$\Sigma_{\Delta t} = \begin{bmatrix} K_s|\Delta s_t| & 0 \\ 0 & K_t|\Delta\theta_t| \end{bmatrix}$$

$$D = wheelsDistance$$

$$V_t = \begin{bmatrix} \frac{1}{2}cos(\beta_t) - \frac{\Delta s}{2D}sin(\beta_t) & \frac{1}{2}cos(\beta_t) + \frac{\Delta s}{2D}sin(\beta_t) \\ \frac{1}{2}sin(\beta_t) + \frac{\Delta s}{2D}cos(\beta_t) & \frac{1}{2}sin(\beta_t) - \frac{\Delta s}{2D}cos(\beta_t) \\ \frac{1}{D} & \frac{1}{D} \end{bmatrix}$$

$$R_t = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix}$$

A segunda linha do algoritmo calcula a matriz K_t, conhecida como o ganho de Kalman. Essa matriz pode ser entendida como o mecanismo que indica se deve-se confiar no valor estimado (odometria) ou na observação dos sensores (z). No cenário proposto, os sensores são capazes de estimar a posição de *landmarks* cuja posição real é conhecida anteriormente. A matriz Q representa a distribuição dos erros das observações. A matriz H_t é a matriz Jacobiana do cálculo de z. Portanto, para cada *landmark* que encontra-se em (L_x, L_y), as matrizes H e Q recebem duas linhas:

$$x_t = \bar{\mu}_t[x], y_t = \bar{\mu}_t[y]$$

$$q = (L_x - x_t)^2 + (L_y - y_t)^2$$

$$H_t = \begin{bmatrix} -\frac{L_x - x_t}{\sqrt{q}} & -\frac{L_y - y_t}{\sqrt{q}} & 0 \\ \frac{L_y - y_t}{q} & -\frac{L_x - x_t}{q} & -1 \end{bmatrix}$$

$$Q_t = \begin{bmatrix} \sigma_{Id}^2 & 0 \\ 0 & \sigma_{I\theta}^2 \end{bmatrix}$$

A terceira linha do algoritmo realiza o cálculo de μ_t. A equação $\bar{z}_t - z_t$ calcula um valor chamado de *inovação* ou resíduo. Em termos práticos ele apenas estima a diferença entre onde o *landmark* deveria estar e onde ele está. Para cada *landmark* utilizado duas linhas são acrescentadas as matrizes:

$$z = \begin{bmatrix} L_{range} \\ L_{bearing} \end{bmatrix}$$

$$\bar{z} = \begin{bmatrix} l_{range} \\ l_{bearing} \end{bmatrix}$$

Onde L representa a posição real do *landmark* e l a posição calculada através dos sensores. As funções de distância e inclinação podem ser calculadas através das equações:

$$p_{range} = \sqrt{(p_x - x_t)^2 + (p_y - y_t)^2}$$

$$p_{bearing} = arctan2((p_y - y_t), (p_x - x_t)) - \theta_t$$

B. Implementação

A implementação do filtro extendido de Kalman encontra-se na classe *KalmanFilterPoseUpdater*. É importante notar que este componente utiliza o cálculo da odometria e sempre realiza a atualização do último valor calculado pela mesma.

Os valores das constantes utilizadas estão descritos na tabela I

Tabela I
CONSTANTES UTILIZADAS

Variável	Valor
K_s	0.1
K_t	0.1
σ_x	1
σ_y	1
σ_θ	1
σ_{Id}	0.5
$\sigma_{I\theta}$	0.1

V. RESULTADOS

Os experimentos realizados com as técnicas de estimativa de pose do robô utilizam o algoritmo de Braitenberg para movimentação do mesmo.

A. Odometria

O gráfico comparando a posição real do robô e a posição calculada através da odometria pose ser visto na figura 3. 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

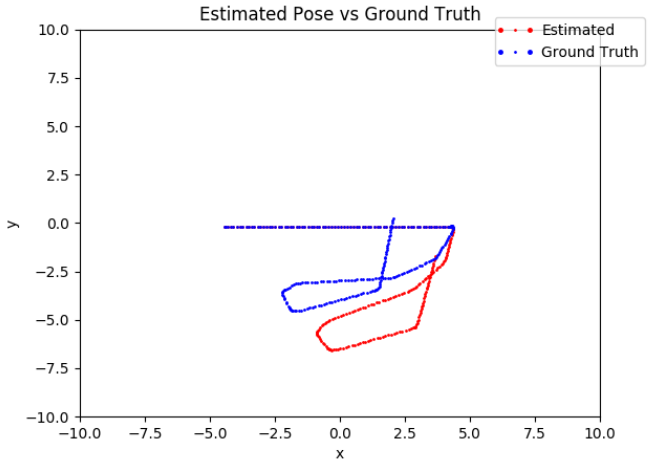

Figura 3. Odometria

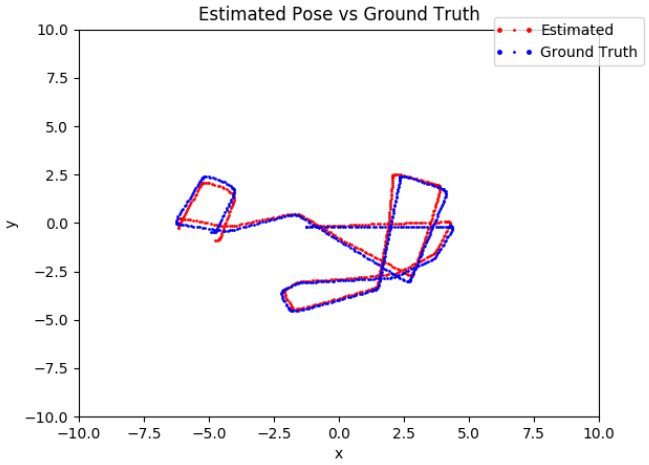

Figura 4. EKF: Utilizando 1 base

Tabela II
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

acumulados. A tabela II 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.

B. *Filtro de Kalman Extendido*

A figura 4 mostra a comparação entre a posição estimada pela odometria e o filtro de Kalman e a posição real do robô quando apenas uma base é utilizada. Pode-se observar a evidente melhoria no trajeto calculado em comparação com a figura 3. Porém também fica evidente que em alguns pontos o erro na predição da rota é maior do que em outros.

Na tentativa de melhorar o cálculo da trajetória, foi acrescentada mais uma base, na posição $(3, -3)$. O resultado desta iteração pode ser visto na figura 5. A rota torna-se mais precisa do que quando utilza-se apenas uma base.

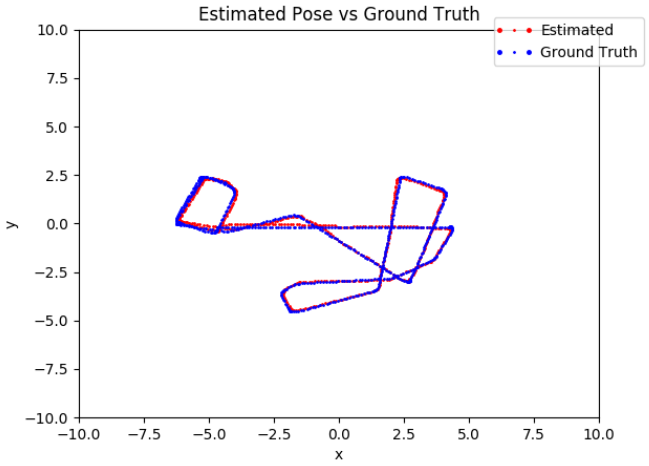

Figura 5. EKF: Utilizando 2 bases

Acrescentando mais uma base no ambiente, na coordenada $(4, 6)$, o cálculo da trajetória fica muito próximo da trajetória real, como pode ser notado na figura 6.

VI. CONCLUSÃO

O filtro extendido de Kalman mostrou-se uma excelente maneira de realizar correções no calculo da odometria. A hipótese de estimar a pose do robô utilizando apenas uma base mostrou-se válida, porém exibe erros consideráveis que podem prejudicar os comportamentos do robô que dependem desta estimativa. Os experimentos deixaram claro que quanto mais *landmarks* são acrescentados mais preciso fica a estimativa da pose.

+————————-+

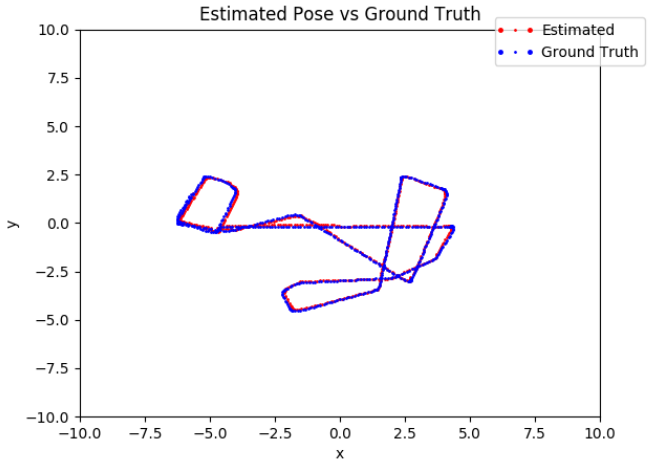

Figura 6. EKF: Utilizando 3 bases

REFERÊNCIAS

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005. 3

[2] P. Pinheiro. (2014, May) Kalman filter for mobile robot localization @ONLINE. [Online]. Available: https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxwYXVsb3BpbmV8Z3g6NWExMGI2MmZhZmViMzg33