FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

TA154 - Robótica Móvil

Práctica 3 -Localización EKF y Mapeo

1C2025

Dominguez Lautaro Tadeo 109645 ltdominguez@fi.uba.ar

Índice

1. Filtro de Kalman Extendido

En este apartado del informe se implementará un Filtro de Kalman Extendido (EKF) utilizando una estructura de código provista por la cátedra. Dicha estructura contiene los elementos básicos del algoritmo, pero requiere la implementación de las etapas de predicción y corrección, tarea que se desarrollará a continuación.

1.1. Paso de predicción EKF

Para iniciar con la etapa de predicción, consideramos un robot diferencial operando en el plano, cuyo estado está definido por el vector (x, y, θ). El modelo de movimiento utilizado es el modelo de odometría visto en clase, el cual se detalla a continuación.

$$g(\bar{x}_{t-1}, \bar{u}_t) = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} = \begin{pmatrix} x_{t-1} + \delta_{trans} \cdot \cos(\theta_{t-1} + \delta_{rot1}) \\ y_{t-1} + \delta_{trans} \cdot \sin(\theta_{t-1} + \delta_{rot1}) \\ \theta_{t-1} + \delta_{rot1} + \delta_{rot2} \end{pmatrix} \tag{1}$$

Donde $\bar{x}_{t-1}$ es la pose del robot en un instante anterior y $\bar{u}_t$ es la acción que realiza el robot. A partir de esta definición de g, se calculó G que corresponde al jacobiano de dicha función.

$$G = \begin{bmatrix} 1 & 0 & -\delta_{trans} \cdot \sin(\theta_{t-1} + \delta_{rot1}) \\ 0 & 1 & \delta_{trans} \cdot \cos(\theta_{t-1} + \delta_{rot1}) \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Una vez calculado el jacobiano, se partió a la implementación del paso de predicción en software *MATLAB*, como se muestra a continuación.

```
function [mu, sigma] = prediction_step(mu, sigma, u)
    % Updates the belief, i. e., mu and sigma, according to the motion model
    %
    % u: odometry reading (r1, t, r2)
    % mu: 3 x 1 vector representing the mean (x, y, theta) of the normal distribution
    % sigma: 3 x 3 covariance matrix of the normal distribution

    % Compute the noise-free motion. This corresponds to the function g, evaluated
    % at the state mu.

    theta_tmenos1 = mu(3);

    mu = [mu(1)+ u.t*cos(mu(3) + u.r1);
          mu(2) + u.t*sin(mu(3) + u.r1);
          mu(3) + u.r1 + u.r2];

    % Compute the Jacobian of g with respect to the state
    G = [1, 0, -u.t*sin(theta_tmenos1 + u.r1);
         0, 1, u.t*cos(theta_tmenos1 + u.r1);
         0, 0, 1];

    % Motion noise
    Q = [0.2, 0, 0;
         0, 0.2, 0;
         0, 0, 0.02];

    sigma = G*sigma*G' + Q;
end
```

Listing 1: Prediction step

1.2. Paso de corrección EKF

El paso de corrección tiene como objetivo incorporar las mediciones de los sensores para reducir la incertidumbre generada durante el movimiento. En este caso, se cuenta con sensores que únicamente miden distancias (*range*) del robot a ciertos puntos conocidos del entorno, denominados *landmarks*, cuyo comportamiento se modela mediante la siguiente ecuación.

$$h(\bar{x}_{t-1}) = \begin{pmatrix} z_{1,t} \\ \vdots \\ z_{n,t} \end{pmatrix} = \begin{pmatrix} \sqrt{(x_{t-1} - l_{1,x})^2 + (y_{t-1} - l_{1,y})^2} \\ \vdots \\ \sqrt{(x_{t-1} - l_{n,x})^2 + (y_{t-1} - l_{n,y})^2} \end{pmatrix} \tag{3}$$

Donde n es la cantidad de landmarks y el resultado de la función los que se espera que los sensores midan en el instante t. A partir de esto de cálculo H, jacobiano correspondiente a la función $h(\bar{x}_{t-1})$.

$$H = \begin{bmatrix} \frac{-(x_{t-1}-l_{1,x})}{\sqrt{(x_{t-1}-l_{1,x})^2+(y_{t-1}-l_{1,y})^2}} & \frac{-(y_{t-1}-l_{1,y})}{\sqrt{(x_{t-1}-l_{1,x})^2+(y_{t-1}-l_{1,y})^2}} & 0 \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ \frac{-(x_{t-1}-l_{1,x})}{\sqrt{(x_{t-1}-l_{1,x})^2+(y_{t-1}-l_{1,y})^2}} & \frac{-(y_{t-1}-l_{n,y})}{\sqrt{(x_{t-1}-l_{n,x})^2+(y_{t-1}-l_{n,y})^2}} & 0 \end{bmatrix} \tag{4}$$

Una vez calculado el jacobiano, se partió a la implementación del paso de corrección en software *MATLAB*, como se muestra a continuación.

```
function [mu, sigma] = correction_step(mu, sigma, z, l)
    % Updates the belief, i. e., mu and sigma, according to the sensor model
    %
    % The employed sensor model is range-only.
    %
    % mu: 3 x 1 vector representing the mean (x, y, theta) of the normal distribution
    % sigma: 3 x 3 covariance matrix of the normal distribution
    % z: structure containing the landmark observations, see
    %    read_data for the format
    % l: structure containing the landmark position and ids, see
    %    read_world for the format


    % Compute the expected range measurements.
    % This corresponds to the function h.
    expected_ranges = zeros(size(z, 2), 1);
    for i = 1:size(z, 2)
        landmark_id = z(i).id;
        idx = find([l.id] == landmark_id); % busca el indice del ID correcto

        expected_ranges(i) = sqrt( (l(idx).x - mu(1))^2 + (l(idx).y - mu(2))^2);
    end
    H = zeros(size(z, 2), 3);      % Jacobian of h
    Z = zeros(size(z, 2), 1);      % Measurements in vectorized form)];
    for i = 1:size(z, 2)

        dx = l(z(i).id).x - mu(1);
        dy = l(z(i).id).y - mu(2);
        q = sqrt(dx^2 + dy^2);

        H(i, :) = [-dx / q, -dy / q, 0];
        Z(i) = z(i).range ;
    end

    R = diag(repmat([0.5], size(z, 2), 1));
    K = sigma*H'*inv(H*sigma*H'+R);
    mu = mu + K*(Z- expected_ranges);
    sigma = (eye(rank(sigma))-K*H)*sigma;
end
```

Listing 2: Prediction step

1.3. Resultados

Como producto de la implementación del Filtro de Kalman Extendido, se generó una simulación que permite observar el comportamiento del algoritmo en tiempo de ejecución. En el siguiente video se muestra el resultado obtenido a partir de la ejecución del código, donde pueden visualizarse tanto la evolución del estado estimado del robot como la posicion de los landmarks. El video esta disponible en el siguiente link.

2. Mapeo con poses conocidas

2.1. Desarrollo

En esta sección se implementa un algoritmo de mapeo unidimensional basado en un modelo de sensor de distancia con poses conocidas.El robot permanece estático en c_0 y observa el entorno con un sensor orientado hacia el eje positivo c_n, como se ilustra en la siguiente figura:

Figura 1: Representación de grilla unidimensional

El modelo de sensor utilizado se basa en una asignación probabilística según la distancia medida. Cada celda se actualiza según las siguientes reglas:

- Si la celda esta antes del obstaculo, la celda se considera libre, y se asigna una probabilidad de ocupación baja: $P(\text{ocupado}) = 0{,}3$
- Si la celda esta mas allá de la distancia medida , se considera que puede haber un obstáculo. Se asigna una probabilidad de ocupación mayor: $P(\text{ocupado}) = 0{,}6$
- Las celdas ubicadas a mas de $20cm$ por detrás de la distancia medida no cambian su probabilidad de ocupación

Teniendo esto en cuenta se partió a la implementación en *MATLAB*, donde se utilizó un enfoque de *log-odds* para actualizar las probabilidades de ocupación de una grilla.

```
function [mu, sigma] = correction_step(mu, sigma, z, l)
    % Updates the belief, i. e., mu and sigma, according to the sensor model
    %
    % The employed sensor model is range-only.
    %
    % mu: 3 x 1 vector representing the mean (x, y, theta) of the normal distribution
    % sigma: 3 x 3 covariance matrix of the normal distribution
    % z: structure containing the landmark observations, see
    %    read_data for the format
    % l: structure containing the landmark position and ids, see
    %    read_world for the format

    % Compute the expected range measurements.
    % This corresponds to the function h.
    expected_ranges = zeros(size(z, 2), 1);
    for i = 1:size(z, 2)
        landmark_id = z(i).id;
        idx = find([l.id] == landmark_id); % busca el indice del ID correcto

        expected_ranges(i) = sqrt( (l(idx).x - mu(1))^2 + (l(idx).y - mu(2))^2);
    end
    H = zeros(size(z, 2), 3);      % Jacobian of h
    Z = zeros(size(z, 2), 1);      % Measurements in vectorized form)];
```

```
24
25      for i = 1:size(z, 2)
26
27          dx = l(z(i).id).x - mu(1);
28          dy = l(z(i).id).y - mu(2);
29          q = sqrt(dx^2 + dy^2);
30
31          H(i, :) = [-dx / q, -dy / q, 0];
32          Z(i) = z(i).range ;
33      end
34
35      R = diag(repmat([0.5], size(z, 2), 1));
36      K = sigma*H'*inv(H*sigma*H'+R);
37      mu = mu + K*(Z- expected_ranges);
38      sigma = (eye(rank(sigma))-K*H)*sigma;
39  end
```

Listing 3: Prediction step

2.2. Resultados

Se realizaron dos experimentos utilizando el mismo algoritmo de mapeo pero con diferentes conjuntos de mediciones. El primero, correspondiente al vector z_1, se obtuvo con un sensor básico que presenta mayor variabilidad en sus lecturas. El segundo conjunto, z_2, fue tomado con un sensor de mayor precisión, con mediciones más consistentes. A continuación, se presentan los mapas resultantes de cada caso.

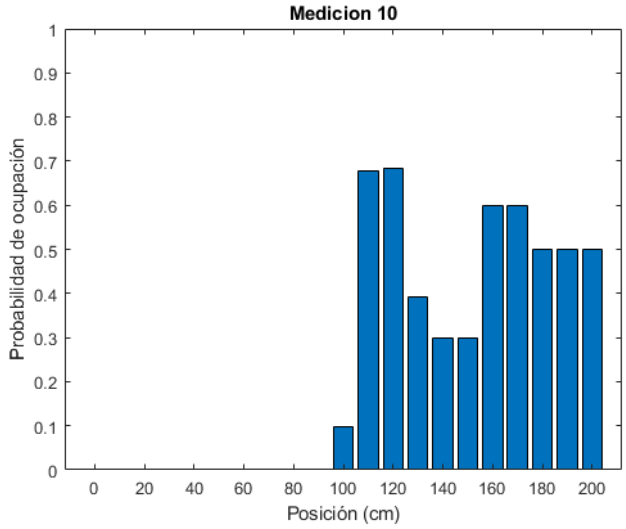

Figura 2: Mapa con sensor de menor calidad z1

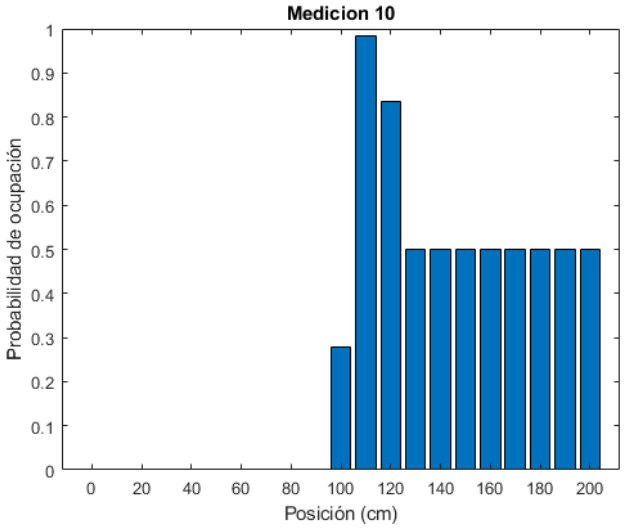

Figura 3: Mapa con sensor de mayor calidad z2

Al comparar los resultados, se puede observar que el mapa generado a partir del sensor de mayor calidad z_2 presenta mayor certeza en la localización de los obstáculos. En cambio, el mapa obtenido con el sensor básico z_1 muestra una mayor variabilidad y ambigüedad en la ubicación de los obstáculos, como resultado del ruido presente en sus mediciones.

A su vez, se generaron videos que muestran paso a paso cómo se actualiza el mapa de ocupación en función de cada medición, tanto para el caso del sensor básico como para el del sensor de mayor calidad. Estas visualizaciones permiten observar de forma dinámica cómo el modelo inverso del sensor va incorporando nueva información y mejorando la estimación del entorno.