Actividad 6 (Evaluación Filtro de KALMAN)

Jose Angel Ramirez Ramirez | A01735529


```
function extended_kalman_filter()
```

Se declaran los parametros para el instante 0, donde al concocer exactamente donde esta el robot, sabemos el valor de MuK y que por la tanto no hay covarianza

```
    % Parámetros iniciales
    MuK_ant = [0; 0; 0];
    SigmaK_ant = zeros(3, 3);
```

Se declaran nuestras variables de control que en este caso son la velocidad lineal y la velocidad angular, tambien se declara nuestro tiempo de muestreo, las matrices de ruido tanto para el sensor extereoceptivo como para el propioceptivo y el landmark

```
    % Parámetros de control y ruido
    vk = 1;
    wk = 1;
    dt = 0.1;
    Qk = [0.50 0.01 0.01;
          0.01 0.50 0.01;
          0.01 0.01 0.20];
    Rk = [0.1  0;
          0    0.02];
    landmark_pos = [3; 4];
```

Se declaran las medidad observadas de nuestro robot por el sensor extereoceptivo en cada uno de los 3 instantes

```
    % Medidas observadas
    Zik_array = [4.87, 0.8;
                 4.72, 0.72;
                 4.69, 0.65];
    % Almacenamiento de resultados
    MuK_store = zeros(3, 3);
    SigmaK_store = zeros(3, 3, 3);
```

Declaramos un bucle que nos permita hacer nuestras 3 iteraciones de interes

```
    % Loop para las tres iteraciones
```

```
    for iteracion = 1:3
        Zik = Zik_array(iteracion, :)';
```

Paso 2: Calculamos la posicion estimada de nuestro robot

```
        % Predicción del estado
        MuK_hat = [MuK_ant(1) + dt * vk * cos(MuK_ant(3));
                   MuK_ant(2) + dt * vk * sin(MuK_ant(3));
                   MuK_ant(3) + dt * wk];
```

Paso 3: Calcular el modelo linealizado usando propagación de incertidumbre (Jacobiano)

```
        % Jacobiano de la función de transición de estado
        Hk = [1 0 -dt * vk * sin(MuK_ant(3));
              0 1  dt * vk * cos(MuK_ant(3));
              0 0 1];
```

Paso 4: *Calcular propagación de la incertidumbre*

```
        % Predicción de la covarianza del estado
        SigmaK_hat = Hk * SigmaK_ant * Hk' + Qk;
```

Paso 6: Calcular medida con el Modelo de observación

```
        % Calculada delta_x, delta-y y p
        dx = landmark_pos(1) - MuK_hat(1);
        dy = landmark_pos(2) - MuK_hat(2);
        p = dx^2 + dy^2;

        % Distancia esperada al marcador
        Zik_hat = [sqrt(dx^2 + dy^2);
                   atan2(dy, dx) - MuK_hat(3)];
```

Paso 7: *Linealizar el Modelo de observación*

```
        % Jacobiano de la función de observación
        Gk = [-dx/(sqrt(p))  -dy/(sqrt(p))  0;
               dy/p          -dx/p         -1];
```

Paso 8: *Incertidumbre de la medición*

```
        % Covarianza de la innovación
        Zk = Gk * SigmaK_hat * Gk' + Rk;
```

Paso 9: Obtenere la ganancia de Kalman

```
        % Ganancia de Kalman
        Kk = SigmaK_hat * Gk' / Zk;
```

Paso 10: *Calcular la posición del robot usando la oservación real z_i*

```
        % Corrección del estado
        MuK = MuK_hat + Kk * (Zik - Zik_hat);
```

Paso 11: Calcular la covarianza

```
        % Corrección de la covarianza del estado
        I = eye(3);
        SigmaK = (I - Kk * Gk) * SigmaK_hat;

        % Almacenar resultados
        MuK_store(:, iteracion) = MuK;
        SigmaK_store(:, :, iteracion) = SigmaK;

        % Actualizar estado anterior
        MuK_ant = MuK;
        SigmaK_ant = SigmaK;
    end
```

Se grafican las posiciones obtenidas

```
    % Graficar posiciones y elipses de confianza
    figure;
    hold on;
    for iteracion = 1:3
        MuK = MuK_store(:, iteracion);
        SigmaK = SigmaK_store(:, :, iteracion);

        plot(MuK(1), MuK(2), 'bo');
        plot_covariance_ellipse(MuK, SigmaK);
    end
    xlabel('X position');
    ylabel('Y position');
    title('Extended Kalman Filter - Robot Positions and Confidence
Ellipses');
    legend('Estimated Position');
    grid on;
    hold off;
end

function plot_covariance_ellipse(mu, sigma)
    % Obtener elipses de covarianza
    [eigvec, eigval] = eig(sigma(1:2, 1:2));
    theta = linspace(0, 2*pi, 100);
    circle = [cos(theta); sin(theta)];
    ellipse = eigvec * sqrt(eigval) * circle;
    ellipse = bsxfun(@plus, ellipse, mu(1:2));
    plot(ellipse(1, :), ellipse(2, :), 'r');
end
```