Actividad 6.2 - Lazo abierto

Ana Itzel Hernández García

Trayectoria con un robot tipo diferencial aplicando las técnicas: “**Lazo abierto**”

Limpieza de pantalla

```
clear
close all
clc
```

Puntos deseados

```
x_vec = [-9, -7, -2, 3, 5, -9];
y_vec = [10, 1, -10, 1, 10, 10];
```

Contron Lazo Abierto

Parámetros del movimiento

```
ts = 0.01;       % Tiempo de muestreo
v = 10;         % Velocidad lineal
w = 10;         % Velocidad angular
```

Inicialización del robot

```
x = x_vec(1); y = y_vec(1); theta = 0; % Posición y orientación inicial
x1 = []; y1 = []; phi = []; % Trayectoria simulada
hx = []; hy = [];          % Historia del centro del robot

for i = 1:(length(x_vec)-1)
    x_goal = x_vec(i+1);
    y_goal = y_vec(i+1);

    % ROTACIÓN EN EL LUGAR
    % Calcular el ángulo al objetivo
    angle_to_goal = atan2(y_goal - y, x_goal - x);

    % Calcular la diferencia de ángulos más corta
    angle_diff = angle_to_goal - theta;

    % Normalizar la diferencia de ángulos para estar en el rango [-pi, pi]
    angle_diff = mod(angle_diff + pi, 2*pi) - pi;

    % Calcular el tiempo necesario para rotar
    t_rot = abs(angle_diff) / w;  % Tiempo en segundos para llegar al ángulo

    % Actualizar la orientación del robot
    theta = theta + angle_diff;

    % Registrar la trayectoria
```

```
    x1(end+1) = x;
    y1(end+1) = y;
    phi(end+1) = theta;
    hx(end+1) = x;
    hy(end+1) = y;

    % Calcular la distancia al objetivo
    dist = sqrt((x_goal - x)^2 + (y_goal - y)^2);

    % Calcular el tiempo necesario para moverse hasta el objetivo
    t_trans = dist / v;  % Tiempo en segundos para cubrir la distancia

   % Movimiento lineal paso a paso
    n_steps = ceil(t_trans / ts); % Número de pasos
    for j = 1:n_steps
        x = x + v * cos(theta) * ts;
        y = y + v * sin(theta) * ts;

        % Registrar trayectoria paso a paso
        x1(end+1) = x;
        y1(end+1) = y;
        phi(end+1) = theta;
        hx(end+1) = x;
        hy(end+1) = y;
    end
end

k_global = length(x1);
```

SIMULACIÓN VIRTUAL

Configuración de escena

```
scene = figure;
set(scene,'Color','white');
set(gca,'FontWeight','bold');
sizeScreen = get(0,'ScreenSize');
set(scene,'position',sizeScreen);
camlight('headlight');
axis equal;
grid on;
box on;
xlabel('x(m)'); ylabel('y(m)'); zlabel('z(m)');

% Calcular rangos automáticamente basados en la trayectoria real
minX = min(x1) - 1;
maxX = max(x1) + 1;
minY = min(y1) - 1;
maxY = max(y1) + 1;

view(2); % Vista 2D
```

```
axis([minX maxX minY maxY 0 1]); % Rango ajustado a la trayectoria real
```

Límites automáticos

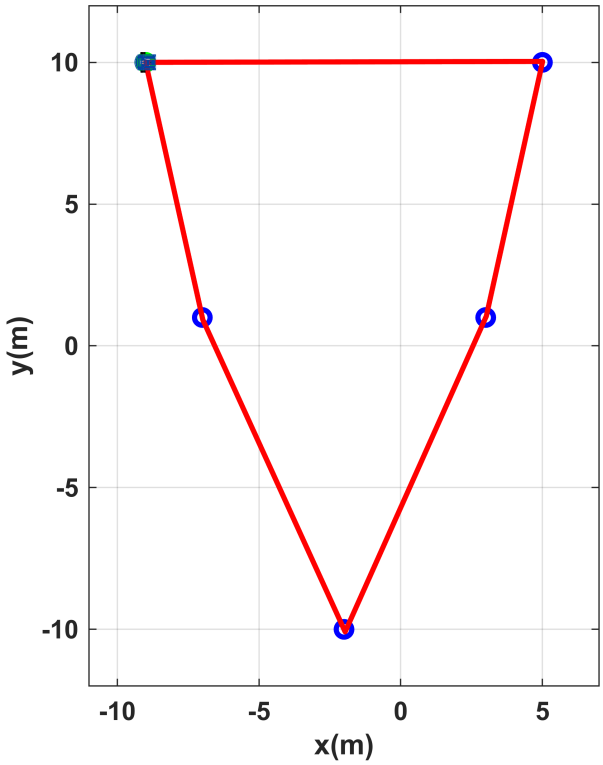
```
xmin_lim = min(x_vec) - 2;
xmax_lim = max(x_vec) + 2;
ymin_lim = min(y_vec) - 2;
ymax_lim = max(y_vec) + 2;
axis([xmin_lim xmax_lim ymin_lim ymax_lim 0 1]);
```

Gráfica inicial

```
scale = 4;
MobileRobot_5;
H1 = MobilePlot_4(x1(1), y1(1), phi(1), scale); hold on;
H2 = plot3(hx(1), hy(1), 0, 'r', 'lineWidth', 2);
H3 = plot3(x_vec, y_vec, 0 * ones(size(x_vec)), 'bo', 'lineWidth', 2);
H4 = plot3(hx(1), hy(1), 0, 'go', 'lineWidth', 2);
```

Animación

```
step = 1;
for k = 1:step:k_global
    delete(H1);
    delete(H2);
    H1 = MobilePlot_4(x1(k), y1(k), phi(k), scale);
    H2 = plot3(hx(1:k), hy(1:k), zeros(1,k), 'r', 'lineWidth', 2);
    pause(ts);
end
```



Ventajas:

- La simplicidad ya que la implementación es directa
- Cuenta con una eficiencia computacional al no necesitar estimación de errores, es decir, consume menos recursos.
- Funciona bien cuando el entorno es predecible y no hay perturbaciones.

Desventajas:

- No hay corrección de errores y cualquier desviación por ruido, errores en la estimación de posición o dinámicas no modeladas no será corregida, por lo tanto no se adapta a obstáculos o cambios
- Tiene una menor precisión, los errores de posición acumulados pueden alejar al robot de su trayectoria deseada.

Actividad 6.2 - Lazo cerrado (Control de Posición)

Ana Itzel Hernández García

Trayectoria con un robot tipo diferencial aplicando las técnicas: “**Lazo cerrado con posiciones deseadas**”

Limpieza de pantalla

```
clear all
close all
clc
```

Puntos deseados

```
x_vec = [-9, -7, -2, 3, 5, -9];
y_vec = [10, 1, -10, 1, 10, 10];
```

Contron Lazo Cerrado

Tiempo de simulación y muestreo

```
tf = 10*length(x_vec);  % Tiempo total (s) de acuerdo a laos puntos
ts = 0.01;          % Tiempo de muestreo (s)
t = 0:ts:tf;
N = length(t);
```

Condiciones iniciales

```
x1(1) = x_vec(1);
y1(1) = y_vec(1);
phi(1) = 0;
```

Punto de control

```
hx(1) = x1(1);
hy(1) = y1(1);
```

Inicializamos vectores de velocidad

```
v = zeros(1, N);
w = zeros(1, N);
Error = zeros(1, N);

k_global = 1; % índice global para recorrer todo el vector de tiempo

% Bucle principal para recorrer todos los puntos de x_vec e y_vec
for i = 1:length(x_vec)

    hxd = x_vec(i);
    hyd = y_vec(i);

    while k_global < N
```

```
        % Errores de posición
        hxe(k_global) = hxd - hx(k_global);
        hye(k_global) = hyd - hy(k_global);
        Error(k_global) = sqrt(hxe(k_global)^2 + hye(k_global)^2);

        % Condición de llegada al punto
        if Error(k_global) < 0.17
            break;
        end

        % Jacobiano
        J = [cos(phi(k_global)) -sin(phi(k_global));
             sin(phi(k_global))  cos(phi(k_global))];

        % Ganancias
        K = [1 0;
             0 1];

        % Ley de control
        he = [hxe(k_global); hye(k_global)];
        qpRef = pinv(J) * K * he;
        v(k_global) = qpRef(1);
        w(k_global) = qpRef(2);

        % Actualización del estado
        phi(k_global + 1) = phi(k_global) + w(k_global) * ts;
        xp1 = v(k_global) * cos(phi(k_global));
        yp1 = v(k_global) * sin(phi(k_global));
        x1(k_global + 1) = x1(k_global) + xp1 * ts;
        y1(k_global + 1) = y1(k_global) + yp1 * ts;
        hx(k_global + 1) = x1(k_global + 1);
        hy(k_global + 1) = y1(k_global + 1);

        k_global = k_global + 1;
    end

end
```

SIMULACIÓN VIRTUAL

Configuración de escena

```
scene = figure;
set(scene,'Color','white');
set(gca,'FontWeight','bold');
sizeScreen = get(0,'ScreenSize');
set(scene,'position',sizeScreen);
camlight('headlight');
axis equal;
grid on;
```

```
box on;
xlabel('x(m)'); ylabel('y(m)'); zlabel('z(m)');

% Calcular rangos automáticamente basados en la trayectoria real
minX = min(x1) - 1;
maxX = max(x1) + 1;
minY = min(y1) - 1;
maxY = max(y1) + 1;

view(2); % Vista 2D
axis([minX maxX minY maxY 0 1]); % Rango ajustado a la trayectoria real
```

Límites automáticos

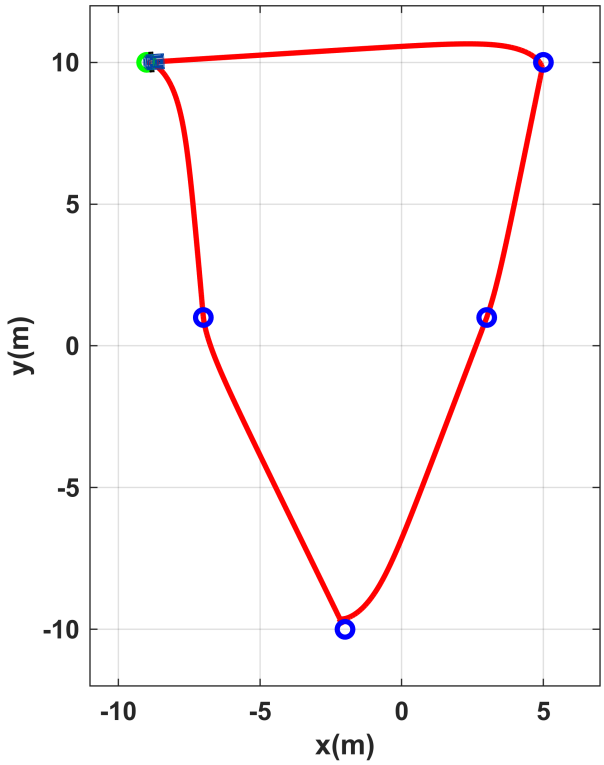
```
xmin_lim = min(x_vec) - 2;
xmax_lim = max(x_vec) + 2;
ymin_lim = min(y_vec) - 2;
ymax_lim = max(y_vec) + 2;
axis([xmin_lim xmax_lim ymin_lim ymax_lim 0 1]);
```

Gráfica inicial

```
scale = 4;
MobileRobot_5;
H1 = MobilePlot_4(x1(1), y1(1), phi(1), scale); hold on;
H2 = plot3(hx(1), hy(1), 0, 'r', 'lineWidth', 2);
H3 = plot3(x_vec, y_vec, 0 * ones(size(x_vec)), 'bo', 'lineWidth', 2);
H4 = plot3(hx(1), hy(1), 0, 'go', 'lineWidth', 2);
```

Animación

```
step = 1;
for k = 1:step:k_global
    delete(H1);
    delete(H2);
    H1 = MobilePlot_4(x1(k), y1(k), phi(k), scale);
    H2 = plot3(hx(1:k), hy(1:k), zeros(1,k), 'r', 'lineWidth', 2);
    pause(ts);
end
```



Ventajas:

- Tiene una mejor precisión, ya que el sistema ajusta continuamente su movimiento.
- Puede corregir desviaciones causadas por ruidos, errores de modelado o entornos no ideales.
- Cunta con adaptabilidad, porque puede llegar a los objetivos aunque no se sigan trayectorias exactas.

Desventajas:

- Requiere cálculos en tiempo real, como las matrices Jacobianas e inversión pseudo-inversa.
- En un entorno físico, se necesita conocer constantemente la posición y orientación actuales del robot por medio de sensores.
- Requiere mayor procesamiento comparado con lazo abierto.

Actividad 6.2 - Lazo cerrado (Control de Trayectoria)

Ana Itzel Hernández García

Trayectoria con un robot tipo diferencial aplicando las técnicas: "**Lazo cerrado con posiciones y velocidades deseadas**"

Limpieza de pantalla

```
clear all
close all
clc
```

Tiempo

```
tf=6.3;              % Tiempo de simulación en segundos (s)
ts=0.02;             % Tiempo de muestreo en segundos (s)
t=0:ts:tf;          % Vector de tiempo
N= length(t);       % Muestras
```

Condiciones iniciales

Damos valores a nuestro punto inicial de posición y orientación

```
x1(1)=-9;  %Posición inicial eje x
y1(1)=10;  %Posición inicial eje y
phi(1)=deg2rad(270); %Orientación inicial del robot
```

Igualamos el punto de control con las proyecciones X1 y Y1 por su coincidencia

```
hx(1)= x1(1);        % Posición del punto de control en el eje (X) metros (m)
hy(1)= y1(1);        % Posición del punto de control en el eje (Y) metros (m)
```

Trayectoria deseada

Puntos deseados

```
x_vec = [-9, -7, -2, 3, 5, -9];
y_vec = [10, 1, -10, 1, 10, 10];

% Número de puntos en la trayectoria
N = length(t);

% Índice de tiempo normalizado de 0 a 1
t_puntos = linspace(0, 1, length(x_vec));
t_interp = linspace(0, 1, N);

% Interpolación lineal
hxd = interp1(t_puntos, x_vec, t_interp);
hyd = interp1(t_puntos, y_vec, t_interp);
```

Velocidades de la trayectoria deseada

```
hxdp=gradient(hxd, ts);
hydp=gradient(hyd, ts);
```

Control, Bucle de simulación

```
for k=1:N

    %a)Errores de control (Aqui la posición deseada ya no es constante,
    % varia con el tiempo)
    hxe(k)=hxd(k)-hx(k);
    hye(k)=hyd(k)-hy(k);

    %Matriz de error
    he= [hxe(k);hye(k)];
    %Magnitud del error de posición
    Error(k)= sqrt(hxe(k)^2 +hye(k)^2);

    %b)Matriz Jacobiana
    J=[cos(phi(k)) -sin(phi(k));... %Matriz de rotación en 2D
       sin(phi(k)) cos(phi(k))];

    %c)Matriz de Ganancias
    K=[2 0;...
       0 2];

    %d)Velocidades deseadas
    hdp=[hxdp(k);hydp(k)];

    %e)Ley de Control:Agregamos las velocidades deseadas
    qpRef= pinv(J)*(hdp + K*he);

    v(k)= qpRef(1);   %Velocidad lineal de entrada al robot
    w(k)= qpRef(2);   %Velocidad angular de entrada al robot
```

Aplicación de control al robot

```
    %Aplico la integral a la velocidad angular para obtener el angulo "phi" de la
orientación
    phi(k+1)=phi(k)+w(k)*ts; % Integral numérica (método de Euler)
```

MODELO CINEMATICO

```
    xp1=v(k)*cos(phi(k));
    yp1=v(k)*sin(phi(k));

    %Aplico la integral a la velocidad lineal para obtener las cordenadas
    %"x1" y "y1" de la posición
    x1(k+1)=x1(k)+ ts*xp1; % Integral numérica (método de Euler)
```

```
    y1(k+1)=y1(k)+ ts*yp1; % Integral numérica (método de Euler)

    % Posicion del robot con respecto al punto de control
    hx(k+1)=x1(k+1);
    hy(k+1)=y1(k+1);


end
```

Simulación virtual 3D

Configuracion de escena

```
scene=figure;  % Crear figura (Escena)
set(scene,'Color','white'); % Color del fondo de la escena
set(gca,'FontWeight','bold') ;% Negrilla en los ejes y etiquetas
sizeScreen=get(0,'ScreenSize'); % Retorna el tamaño de la pantalla del computador
set(scene,'position',sizeScreen); % Configurar tamaño de la figura
camlight('headlight'); % Luz para la escena
axis equal; % Establece la relación de aspecto para que las unidades de datos sean
las mismas en todas las direcciones.
grid on; % Mostrar líneas de cuadrícula en los ejes
box on; % Mostrar contorno de ejes
xlabel('x(m)'); ylabel('y(m)'); zlabel('z(m)'); % Etiqueta de los eje

% Calcular rangos automáticamente basados en la trayectoria real
minX = min(x1) - 1;
maxX = max(x1) + 1;
minY = min(y1) - 1;
maxY = max(y1) + 1;

view(2); % Vista 2D
axis([minX maxX minY maxY 0 1]); % Rango ajustado a la trayectoria real
```

Límites automáticos

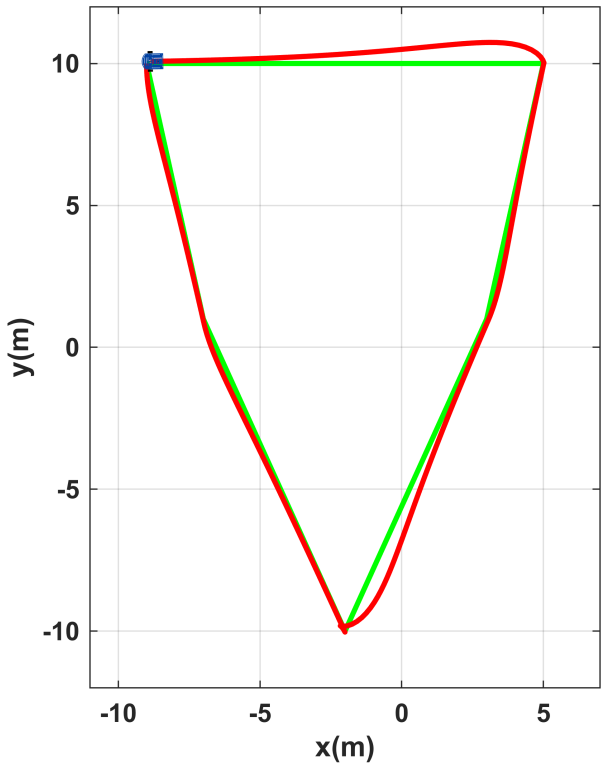
```
xmin_lim = min(x_vec) - 2;
xmax_lim = max(x_vec) + 2;
ymin_lim = min(y_vec) - 2;
ymax_lim = max(y_vec) + 2;
axis([xmin_lim xmax_lim ymin_lim ymax_lim 0 1]);
```

Graficar robots en la posicion inicial

```
scale = 4;
MobileRobot_5;
H1=MobilePlot_4(x1(1),y1(1),phi(1),scale);hold on;
```

Graficar Trayectorias

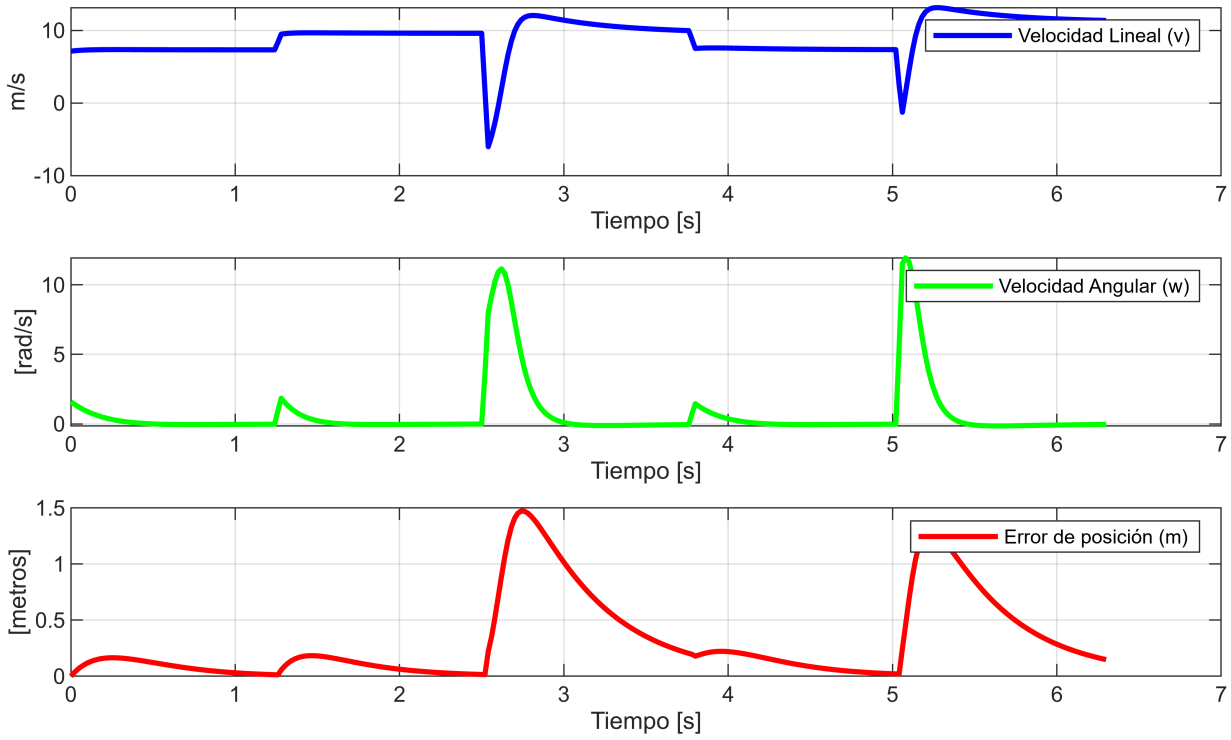
```
H2=plot3(hx(1),hy(1),0,'r','lineWidth',2);
H3=plot3(hxd,hyd,zeros(1,N),'g','lineWidth',2); %Grafico circulo en posición deseada
%H4=plot3(hx(1),hy(1),0,'go','lineWidth',2);%Grafico circulo en posición inicial
% d) Bucle de simulacion de movimiento del robot

step=1; % pasos para simulacion

for k=1:step:N

    delete(H1);
    delete(H2);

    H1=MobilePlot_4(x1(k),y1(k),phi(k),scale);
    H2=plot3(hx(1:k),hy(1:k),zeros(1,k),'r','lineWidth',2);

    pause(ts);

end
```



Graficas

```
graph=figure;  % Crear figura (Escena)
set(graph,'position',sizeScreen); % Congigurar tamaño de la figura
subplot(311)
```

```
plot(t,v,'b','LineWidth',2),grid('on'),xlabel('Tiempo [s]'),ylabel('m/
s'),legend('Velocidad Lineal (v)');
subplot(312)
plot(t,w,'g','LineWidth',2),grid('on'),xlabel('Tiempo [s]'),ylabel('[rad/
s]'),legend('Velocidad Angular (w)');
subplot(313)
plot(t,Error,'r','LineWidth',2),grid('on'),xlabel('Tiempo
[s]'),ylabel('[metros]'),legend('Error de posición (m)');
```



Ventajas:

- Seguimiento continuo y suave de trayectorias, ya que que sigue trayectorias con orientación y velocidad planificada.
- Al considerar velocidades deseadas, se anticipan los movimientos, reduciendo el error de control
- Si la trayectoria cambia con el tiempo, el sistema puede adaptarse en tiempo real.

Desventajas:

- Tiene una mayor complejidad matemática y computacional que involucra derivadas y control vectorial con el Jacobiano.
- Si las velocidades deseadas están mal definidas o derivadas, puede haber inestabilidad o errores amplificados.
- Una mala elección de `ts` puede generar ruido en la estimación de derivadas.
- Una mala elección de k puede causar que el sistema sea muy inestable, tenga una respuesta muy lenta o que reaccione de forma agresiva,

- Antes de comenzar, es necesario conocer toda la trayectoria y sus velocidades asociadas.