CONTROL Y PROGRAMACIÓN DE ROBOTS

ROBÓTICA MÓVIL

ÁLVARO CALVO MATOS
FEDERICO VAZ FERNÁNDEZ
RAÚL ZAHÍNOS MARÍN

GIERM

27 DE DICIEMBRE DE 2019

Índice

1 INTRODUCCIÓN.

El objetivo que se persigue con la realización de este trabajo es el de poner en práctica los conocimientos adquiridos en esta parte de la asignatura.

Para lograr controlar robots móviles se requieren solventar algunos problemas técnicos, comenzando por conocer la constitución del propio robot a controlar. Tipos de ruedas, disposición, grados de libertad, restricciones, actuadores…

El primer paso será el de elaborar un modelo cinemático que describa el movimiento del robot en función de los actuadores. Con él podemos además obtener el modelo cinemático inverso, y así conocer las referencias necesarias para alcanzar el estado o trayectoria que se desee.

Antes de controlar debemos tener en cuenta las dinámicas del robot móvil, aunque en el simulador que se construirá para el control solo se tendrán en cuenta las dinámicas de los actuadores. Es importante no perder de vista que cualquier control diseñado y probado en un simulador basado en modelos simplificados, se comportará, por lo general, peor en la realidad.

Para controlar necesitaremos primero una trayectoria que seguir. Generar una trayectoria supone conocer el terreno por el que nos vamos a mover. Para este ejercicio, supondremos que no existen obstáculos, y que disponemos de todo el plano XY libre para el movimiento de nuestro robot. Solo importará pues la propia trayectoria a seguir, la cual dependerá del movimiento que queramos conseguir.

Por último, necesitaremos sensores con los que poder medir y así cerrar el lazo de control. La naturaleza de estos sensores será crucial para el control del robot móvil, ya que de ellos depende la localización y cálculo de la orientación del robot bajo el sistema de referencia escogido.

El conocimiento de la posición de un robot a partir de los sensores que incorpore es uno de los grandes problemas del control de robots móviles. Aquí se supondrán sensores perfectos, sin ruidos ni errores, de forma que conocemos totalmente el estado actual del robot.

Los controles que se diseñarán serán principalmente controles proporcionales al error presente entre la posición actual del robot y la posición deseada, teniendo en cuenta además el ángulo entre el robot y la posición deseada.

2 ANÁLISIS CINEMÁTICO.

El robot con el que se trabajará presenta una configuración diferencial, cuyas ruedas izquierda y derecha, paralelas entre si y con ejes fijos, están actuadas y tienen la capacidad de girar en ambos sentidos.

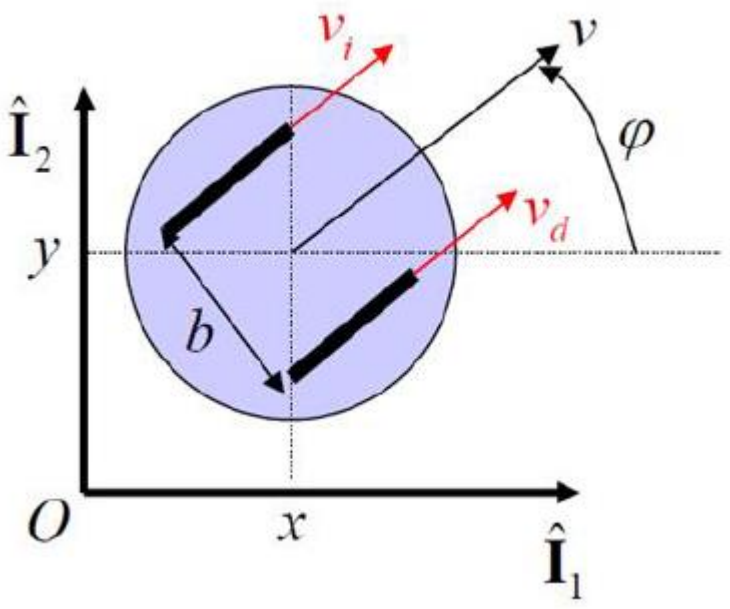

Figura 1. Configuración Diferencial.

Se trata de un robot subactuado, ya que dispone de dos únicos actuadores para el movimiento y la orientación en el plano XY.

Sus parámetros característicos son: b = 0.8 m y R = 0.4 m.

Posteriormente se decidirán el resto de características del robot en consecuencia a estos parámetros.

2.1 MODELO CINEMÁTICO EN SU FORMA JACOBIANA.

El modelo cinemático del robot diferencial en forma jacobiana es el siguiente:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{pmatrix} = \begin{pmatrix} {}^{R}/_{2}\cos(\varphi) & {}^{R}/_{2}\cos(\varphi) \\ {}^{R}/_{2}\sin(\varphi) & {}^{R}/_{2}\sin(\varphi) \\ -{}^{R}/_{b} & {}^{R}/_{b} \end{pmatrix} \begin{pmatrix} \omega_i \\ \omega_d \end{pmatrix}$$

De las cuales:

- Variables generalizadas:

$$q = \begin{pmatrix} x \\ y \\ \varphi \end{pmatrix}$$

- Variables de actuación:

$$p = \begin{pmatrix} \omega_i \\ \omega_d \end{pmatrix}$$

Estas ecuaciones forman el modelo cinemático directo del robot diferencial, y se han escrito dentro de una función de Matlab para disponer de un simulador de la cinemática de nuestro robot.

El código finalmente utilizado sería:

```
function [out] = Modelo_Cinematico(in)
% Variables de actuación - Velocidad angular de las ruedas
w_i=in(1);
w_d=in(2);

% Variables generalizadas
x=in(3);
y=in(4);
phi=in(5);

% Datos geométricos
b = 0.8 ; % Distancia entre las ruedas
R = 0.4 ; % Radio de las ruedas

% Modelo cinemático directo - Forma Jacobiana
J_q=[R/2*cos(phi) R/2*cos(phi);
    R/2*sin(phi) R/2*sin(phi);
    -R/b        R/b        ];

 p_d = [w_i w_d]';

 q_d =J_q * p_d ;

out = q_d;  %[x_d y_d phi_d];
end
```

2.2 TRAYECTORIAS DE LAZO ABIERTO.

En el siguiente apartado se nos pide alimentar al robot con unas actuaciones en velocidad sobre las ruedas constante y una actuación oscilatoria senoidal sobre la variable de dirección, para el caso del robot diferencial esto se traduce en una velocidad extra, senoidal y desfasada para cada rueda.

$$\omega_i = 1.1 + sin(2\pi t)$$

$$\omega_d = 0.2 + sin(2\pi t + \pi)$$

Como resultado de la simulación se obtienen las siguientes graficas:

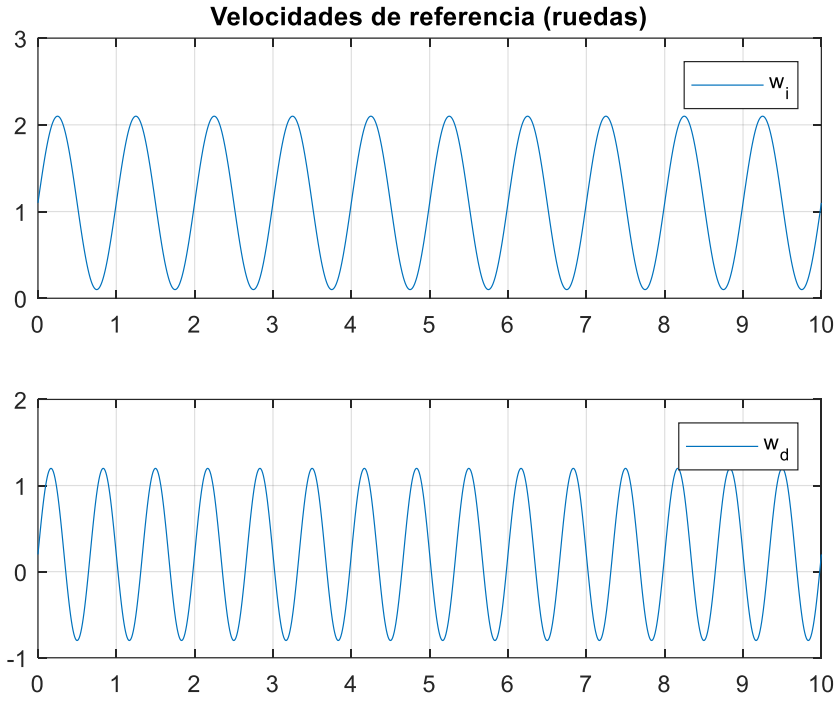

Figura 2. Velocidades de actuación en lazo abierto.

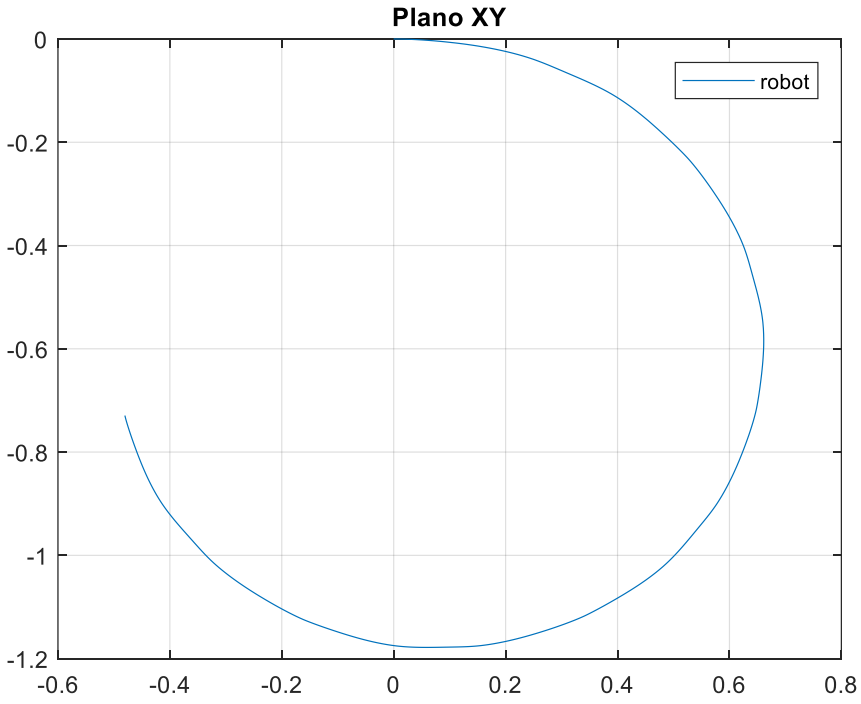

Figura 3. Trayectoria resultante sobre el plano XY de la actuación en lazo abierto.

2.3 TRAYECTORIA PARABÓLICA GENÉRICA.

En este apartado se pide calcular las referencias que deberían de pasarse al robot diferencial para que siga una trayectoria parabólica.

Para ello se nos exige el uso de la jacobiana inversa como modelo de cinemática inversa. Este se ha calculado y expresado en la siguiente función de Matlab:

```
function [out] = Modelo_Cinematico_Inverso(in)
v = in(1);
phi_d = in(2);

% Datos geométricos
b = 0.8 ; % Distancia entre las ruedas
R = 0.4 ; % Radio de las ruedas

w = [1/R -b/(2*R); 1/R b/(2*R)]*[v phi_d]';

w_i = w(1);
w_d = w(2);

out = [w_i w_d];
end
```

El modelo cinemático inverso calcula las velocidades angulares de cada rueda en función del vector velocidad v y de la derivada del ángulo φ. Para calcularlos se ha procedido a parametrizar la ecuación de una parábola genérica respecto al tiempo. Una vez hecho, se han derivado x e y, obteniendo las componentes horizontal y vertical del vector velocidad v. Derivando de forma similar se obtiene $\dot{\varphi}$.

Estos cálculos se han introducido dentro de una función única entrada es el tiempo.

```
function [out] = Generador_de_trayectorias(in)
t = in(1);

x_d = 0.25;
A = 2;
x = x_d * t;
y_d = (-2*x + A)*x_d;



v = sqrt(x_d^2 + y_d^2);
phi_d = -(2*x_d)/((A - 2*x)^2 + 1);

w = Modelo_Cinematico_Inverso([v phi_d]);

w_i = w(1);
w_d = w(2);

out = [w_i w_d];
end
```

El resultado de la simulación resulta bastante bueno como podemos comprobar en la siguiente figura:

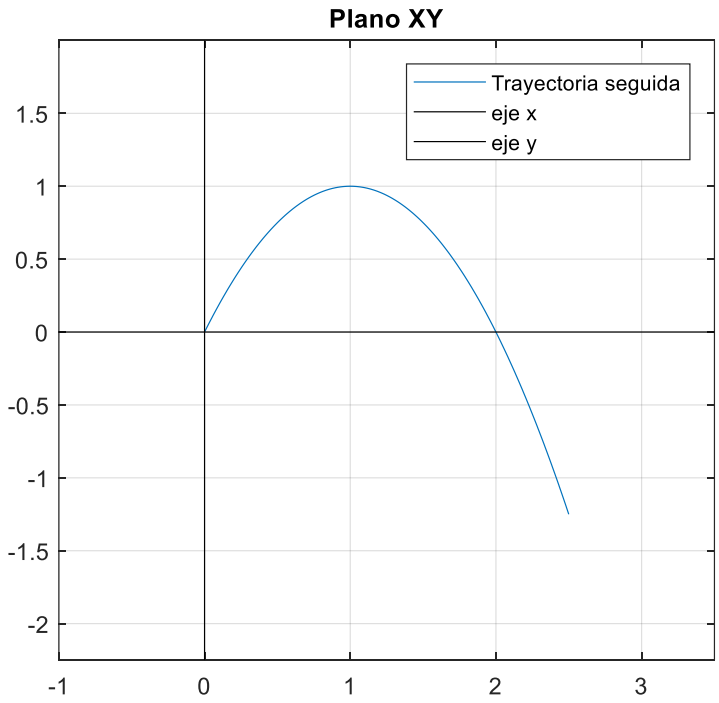

Figura 4. Trayectoria parabólica en lazo abierto sobre el plano XY.

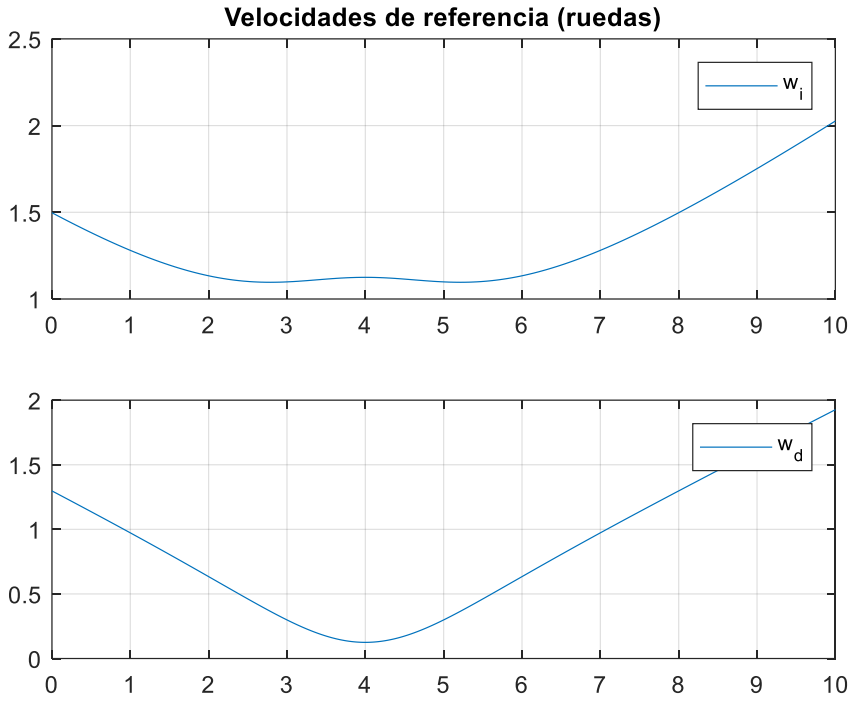

Figura 5. Referencia en velocidad obtenidas para el seguimiento de la trayectoria parabólica.

3 CONTROL DEL ROBOT.

Probado el funcionamiento del simulador cinemático del robot es hora de comenzar a diseñar y probar controles. Se agregarán bloques al modelo de Simulink creado hasta ahora con el objetivo de mejorar un poco más el modelo y de implementar los controles necesarios para el cumplimiento de las diferentes especificaciones.

3.1 DINÁMICA BÁSICA DE LOS ACTUADORES.

Con el objetivo de ser un poco más fieles a la realidad, se agregará la dinámica de los actuadores al simulador, de forma que se vean diferencias entre las referencias que se marcan y las velocidades que toma realmente el robot.

Se empleará para ello una función de transferencia de segundo orden con dos polos en el mismo lugar.

$$G(s) = \frac{1}{(0.1s + 1)^2}$$

Se agregan además saturaciones en velocidad y aceleración para simular otros límites físicos de los actuadores. Se obtiene el siguiente esquema de Simulink que se añadirá a partir de ahora al simulador:

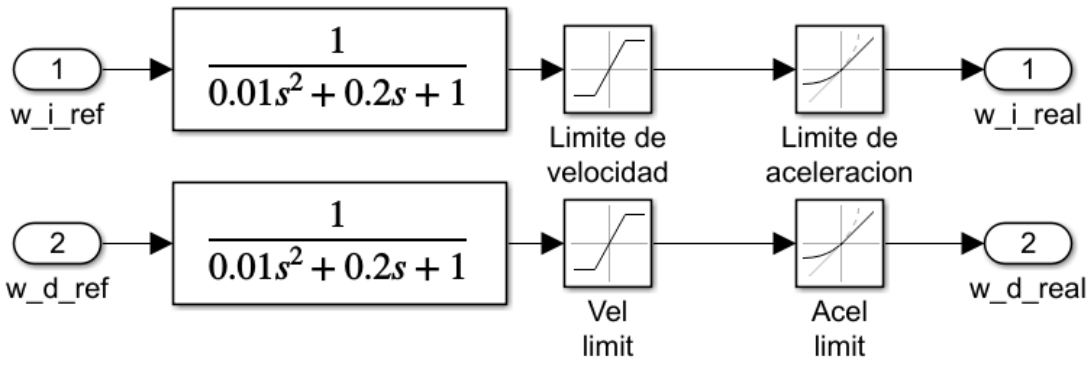

Figura 6. Simulación de la dinámica básica de los actuadores.

Destacar que tanto los valores de los polos como los limites para las saturaciones se han colocado a conveniencia, siguiendo un poco también la lógica conocidas las dimensiones físicas del robot.

3.2 CONTROL PUNTO A PUNTO.

Para llevar a cabo el control de punto a punto se miden la distancia desde el punto actual hasta el punto de destino y el ángulo hacia el cual está el punto de destino. Tratamos esta distancia hasta el destino como un error en distancia y a la diferencia del ángulo actual y el que nos dirige hacia el destino lo tratamos como un error en ángulo.

Disponemos así de dos errores a los cuales podemos aplicar las estrategias clásicas de control PID que se han visto en la asignatura.

La velocidad será calculada para ser proporcional a la distancia hasta la meta, o lo que es lo mismo, proporcional al error en distancia.

Para el control en dirección se usará también otro control proporcional.

```
function [out] = Control_punto(in)
x = in(1);
y = in(2);
phi = in(3);

x_ref=3;
y_ref=3;

Kv=0.5;
Kp=4;

v = Kv * sqrt((x_ref - x)^2+(y_ref - y)^2);

phi_ref = atan2( (y_ref - y),(x_ref - x) );
phi_d = Kp * angdiff(phi_ref, phi);

out = Modelo_Cinematico_Inverso([v phi_d]);

end
```

Tras ajustar los valores de las ganancias y ejecutar la simulación correspondiente se obtiene el siguiente resultado:

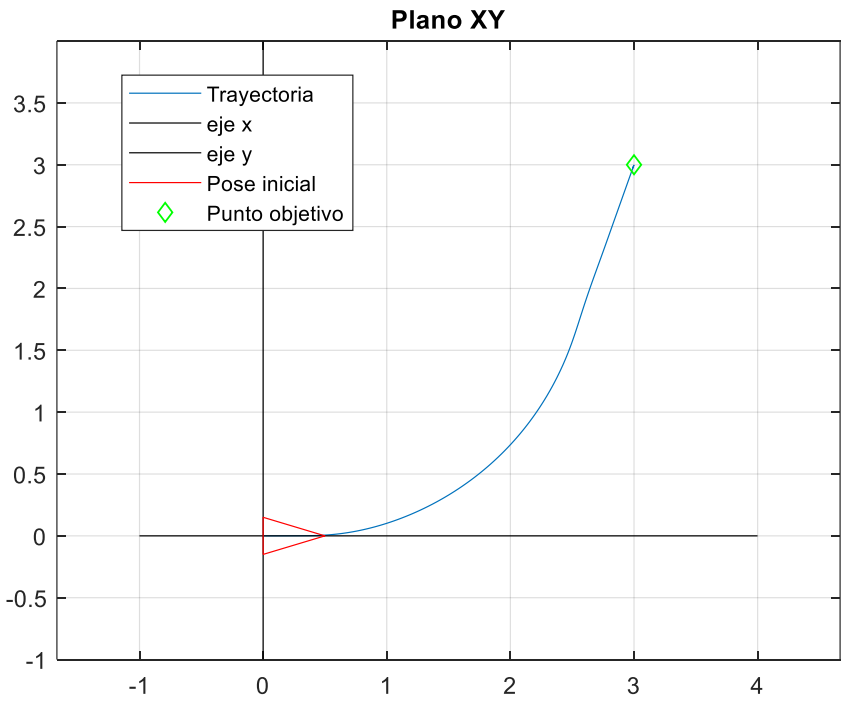

Figura 7. Control punto a punto.

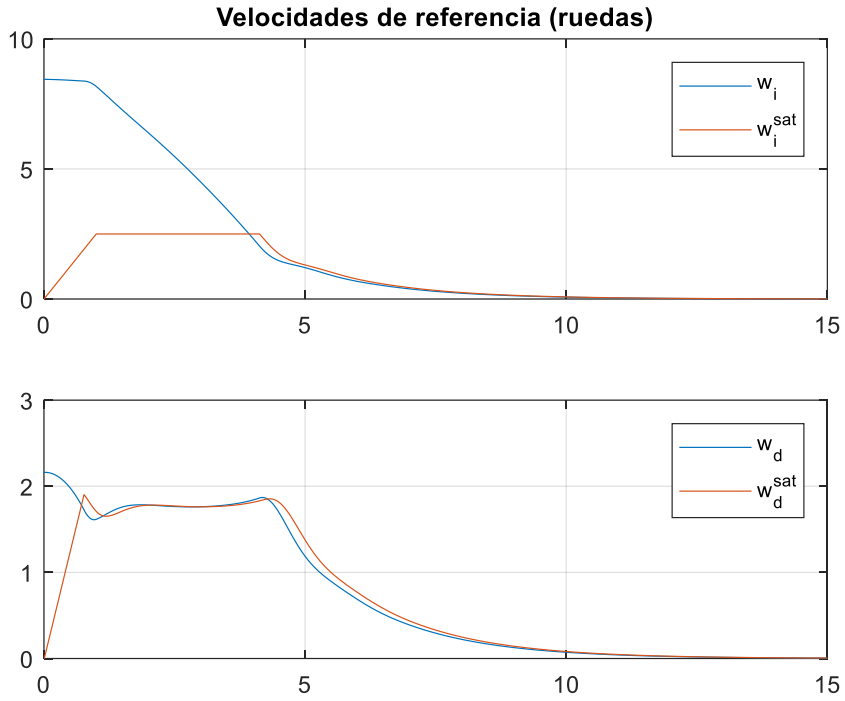

Figura 8. Velocidades de referencia para el control punto a punto.

Tras todas las simulaciones, se observó que, a la hora de representar las gráficas de velocidad de las ruedas, la velocidad de la rueda derecha e izquierda está cambiada. Esta errata continúa en las gráficas posteriores, pero no afecta en absoluto al control, pues es solo un error en la representación de estas.

3.3 CONTROL EN LÍNEA RECTA.

El seguimiento de una línea recta $(ax + by + c = 0)$ requiere de dos controles para ajustar la dirección. Un controlador dirige el robot de forma que se minimice la distancia normal hacia la recta de acorde a la ecuación:

$$d = \frac{(a, b, c)(x, y, 1)}{\sqrt{a^2 + b^2}}$$

El primer control proporcional gira el robot hacia esta línea. El segundo control ajusta el ángulo de la cabeza del robot de forma que sea paralela a la línea.

$$\theta^* = tan^{-1}\frac{-a}{b}$$

Existirá una ley de control combinada que sea resultado de la suma de los dos controles proporcionales.

```
function [out] = Control_linea(in)
x = in(1);
y = in(2);
phi = in(3);

% Recta de referencia
% ax + by + c = 0
a=0.5;
b=-1;
c=0.5;

% Parámetros del controlador
Kd=0.7;     % Proporcional a la distancia
Kh=1.5;       % Proporcional a la orientación

% Distancia a la recta
d=(a*x +b*y +c)/sqrt(a^2 + b^2);

% Orientación de referencia
phi_ref = atan2(a,-b);

% Señales de control
v = 1;
phi_d = Kd*d + Kh * angdiff(phi_ref, phi);

out = Modelo_Cinematico_Inverso([v phi_d]);
out=[out d];
end
```

Como prueba del funcionamiento de este algoritmo de control se muestran a continuación los resultados de la simulación.

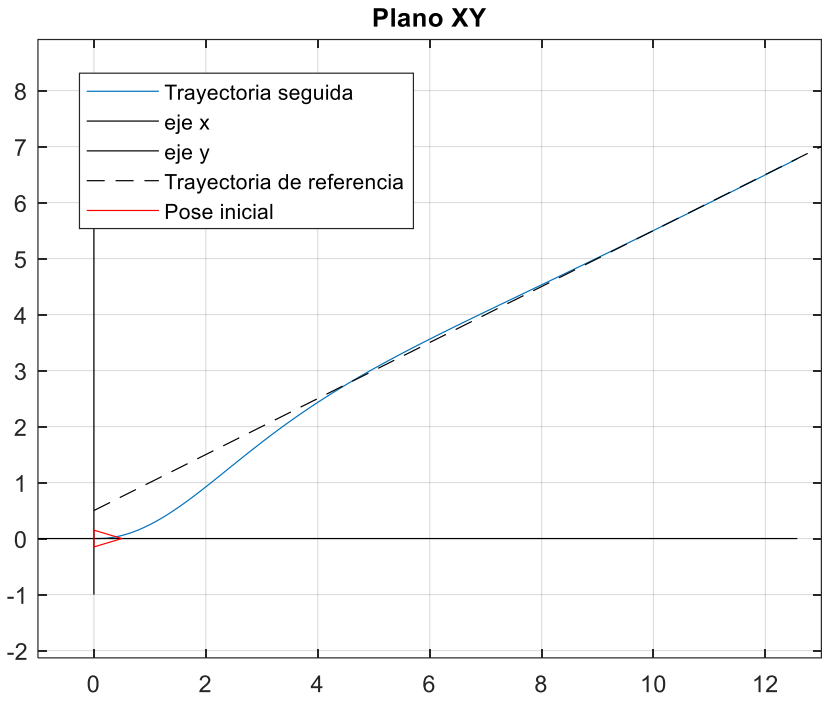

Figura 9. Trayectoria seguida sobre el plano XY durante el control en línea recta.

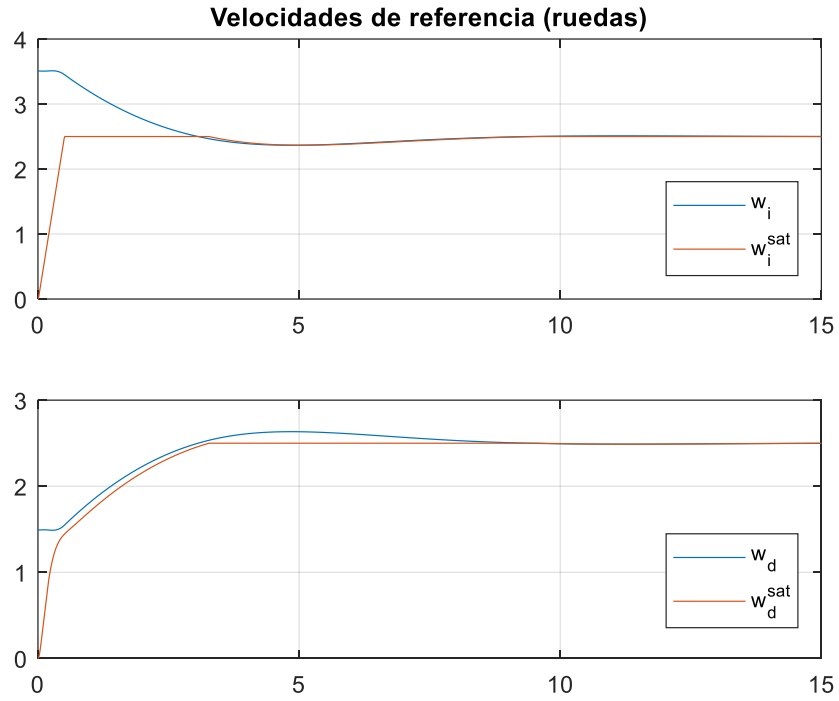

Figura 10. Velocidades necesarias para el seguimiento de la recta.

En la Figura 10 podemos comprobar las dinámicas de los actuadores, que se manifiestan en un tiempo no nulo de respuesta, así como en las notables saturaciones tanto de velocidad como de aceleración.

3.4 CONTROL A UNA TRAYECTORIA.

En lugar de una línea recta, podría desearse seguir una trayectoria definida en el plano XY. Esta trayectoria debe generarse como secuencia de coordenadas desde un generador de trayectorias.

El caso es muy similar al control a un punto, solo que en este caso el punto está en movimiento, manteniendo siempre una distancia d_ref hasta el punto a seguir.

Con ello, se obtiene un error tal que:

$$e = \sqrt{(x^* - x)^2 + (y^* - y)\text{^}2} - d^*$$

Lo cual, con una ganancia, resulta el control para la velocidad. Para el control del ángulo, utilizamos:

$$\theta^* = \tan^{-1}\frac{y^* - y}{x^* - x} \qquad \alpha = K_h(\theta^* \ominus \theta)$$

```
function [out] = Control_trayectoria(in)


x = in(1);
y = in(2);
phi = in(3);
x_ref = in(4);
y_ref = in(5);



% Parámetros del controlador
Kh = 5;      % Proporcional a la orientación

% Orientación y distancia de referencia
phi_ref = atan2((y_ref - y), (x_ref - x));
d_ref = 0.2;    % Distancia de persecución

% Error de seguimiento
e = sqrt((x_ref - x)^2 + (y_ref - y)^2) - d_ref;

% Señales de control
phi_d = Kh * angdiff(phi_ref, phi);


out = [e phi_d];
end
```

Con dicho control, se obtiene en simulación:

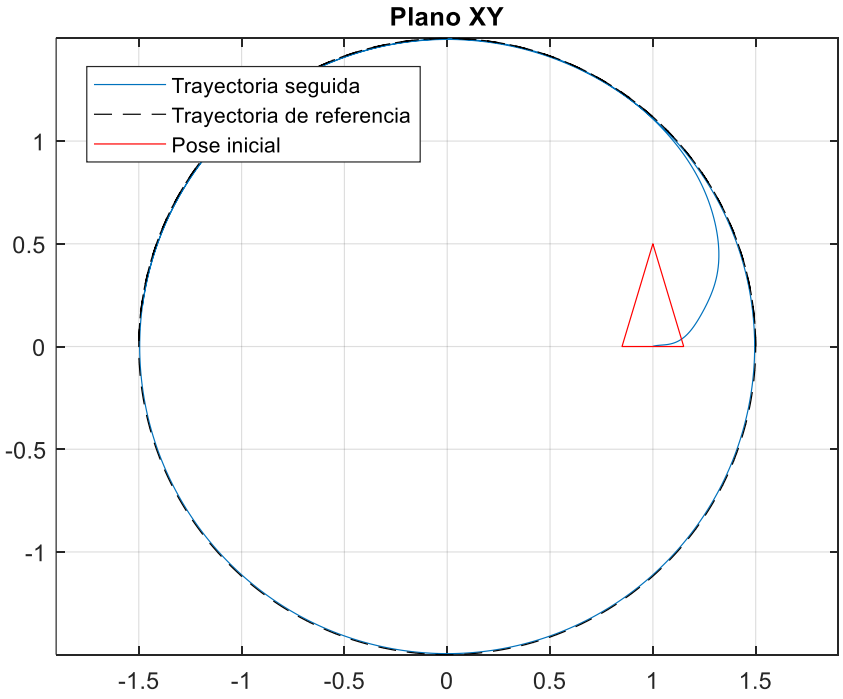

Figura 11. Trayectoria seguida sobre el plano XY durante el control de trayectoria.

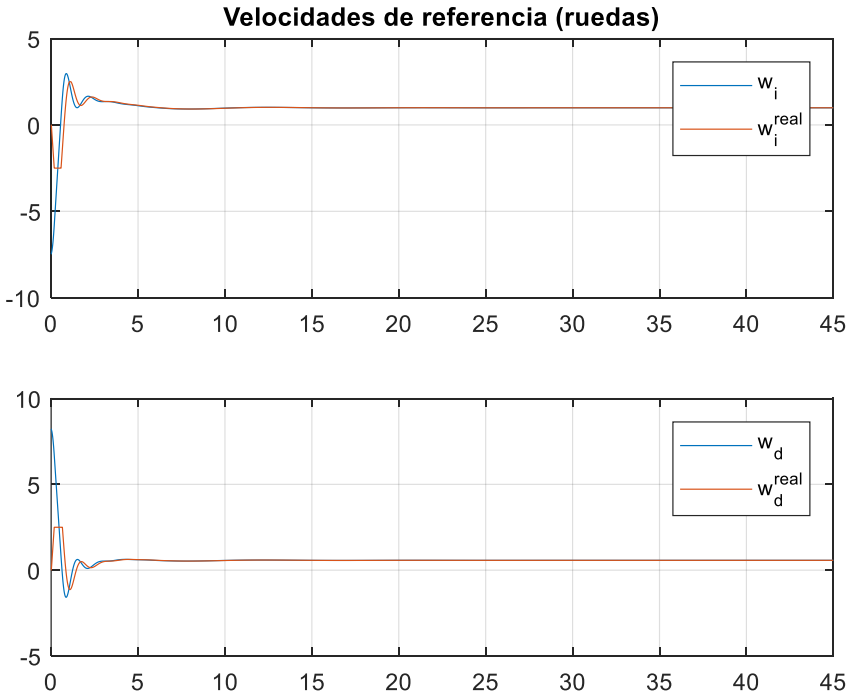

Figura 12. Velocidades necesarias para el seguimiento de la trayectoria.

Se observa al inicio un régimen transitorio abrupto entre que llega a alcanzar la trayectoria marcada.

3.5 CONTROL A UNA POSTURA.

En los controles anteriores, la orientación final no se podía modificar, salvo cambiando la orientación inicial del robot. Este control consiste en añadir dicho aspecto, pudiendo controlarse tanto en posición como en orientación final.

Para ello, se reescriben las ecuaciones, pasando a coordenadas polares:

$$\rho = \sqrt{\Delta_x^2 + \Delta_y^2}$$

$$\alpha = tan^{-1}\frac{\Delta_y}{\Delta_x} - \theta$$

$$\beta = -\theta - \alpha$$

Con lo que la ley de control quedaría tal que:

$$v = k_p \rho$$

$$\gamma = k_\alpha \alpha + k_\beta \beta$$

Donde suponemos que la posición final está frente al vehículo.

k_p y k_α controlan el seguimiento en posición, mientras que k_β controla la orientación, de manera que β tienda a cero cerrando el bucle.

```
function [out] = Control_postura(in)
x = in(1);
y = in(2);
phi = in(3);

% Objetivo:
x_ref = 2;
y_ref = 2;
phi_ref = -pi/3;

% Parámetros del controlador
K_rho=0.5;     % Proporcional a la distancia
K_alpha= 5;       % Proporcional a la orientación
K_beta = -3 ;          %   K_beta < 0 !!!

% Cambio de variables (a polares)
rho = sqrt((x - x_ref)^2 + (y - y_ref)^2 );
alpha = atan2((y_ref - y), (x_ref - x)) - phi;
beta = -phi - alpha;

% Señales de control
v = K_rho*rho;
phi_d = (K_alpha*alpha + K_beta*(beta + phi_ref) );

out = Modelo_Cinematico_Inverso([v phi_d]);
end
```

Los resultados obtenidos con dicho controlador son:

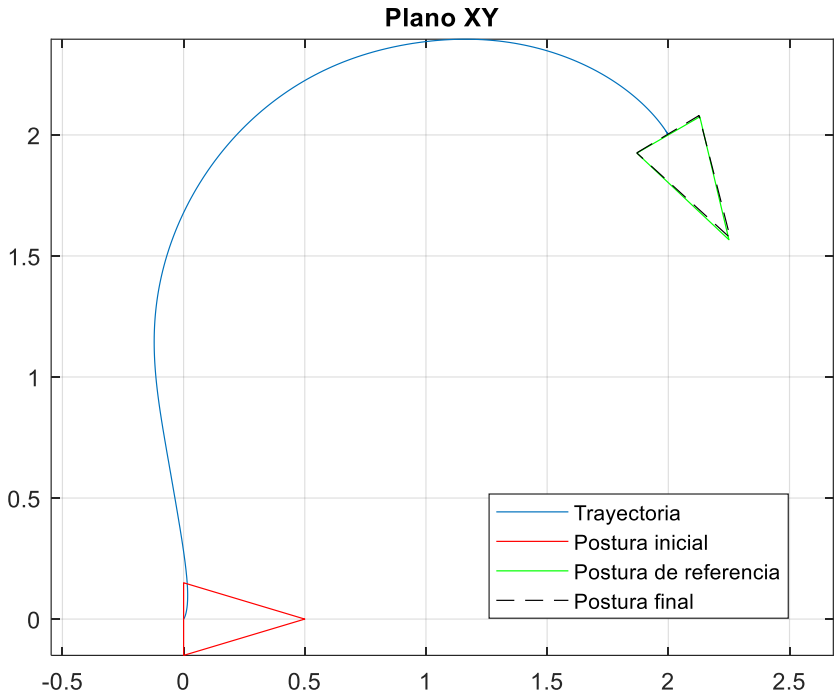

Figura 13. Trayectoria seguida sobre el plano XY durante el control en postura.

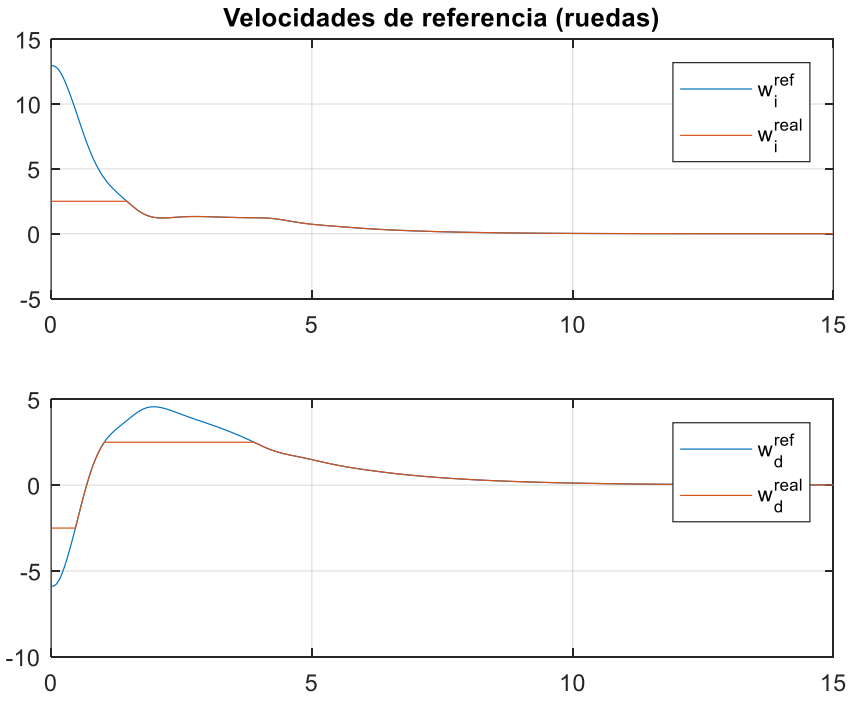

Figura 14. Velocidades necesarias para alcanzar la postura deseada.

En la figura 14 se observa como el robot satura en velocidad para orientarse debidamente al inicio

3.6 CONTROL MEDIANTE ALGORITMO DE PERSECUCIÓN PURA PARA EL SEGUIMIENTO DE UNA TRAYECTORIA SENOIDAL.

El algoritmo de persecución pura resulta muy sencillo y efectivo, en el cual la posición final se va desplazando a lo largo de la trayectoria, a velocidad constante en su forma más sencilla, logrando que el robot siempre apunte hacia la posición final.

El algoritmo de control sigue las mismas leyes que el del caso de seguimiento de trayectoria.

```
function [out] = Control_trayectoria_senoidal(in)

x = in(1);
y = in(2);
phi = in(3);
x_ref = in(4);
y_ref = in(5);


% Parámetros del controlador
Kh = 5;     % Proporcional a la orientacion

% Orientación y distancia de referencia
phi_ref = atan2((y_ref - y), (x_ref - x));
d_ref = 0.1;    % Distancia de persecución

% Error de seguimiento
e = sqrt((x_ref - x)^2 + (y_ref - y)^2) - d_ref;


% Señales de control
phi_d = Kh * angdiff(phi_ref, phi);

out = [e phi_d];
end
```

En simulación se obtiene:

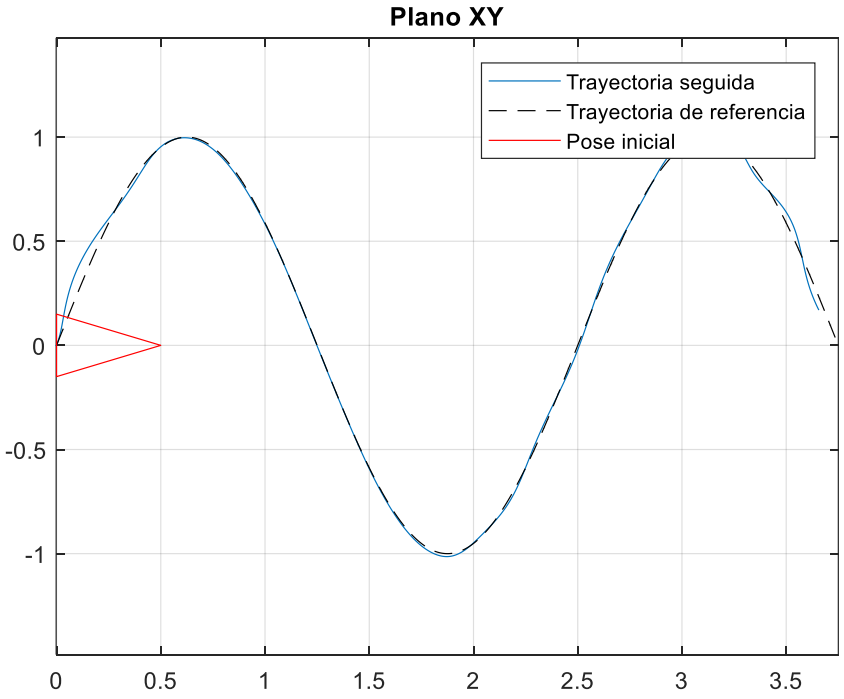

Figura 15. Trayectoria seguida sobre el plano XY durante el control mediante persecución pura.

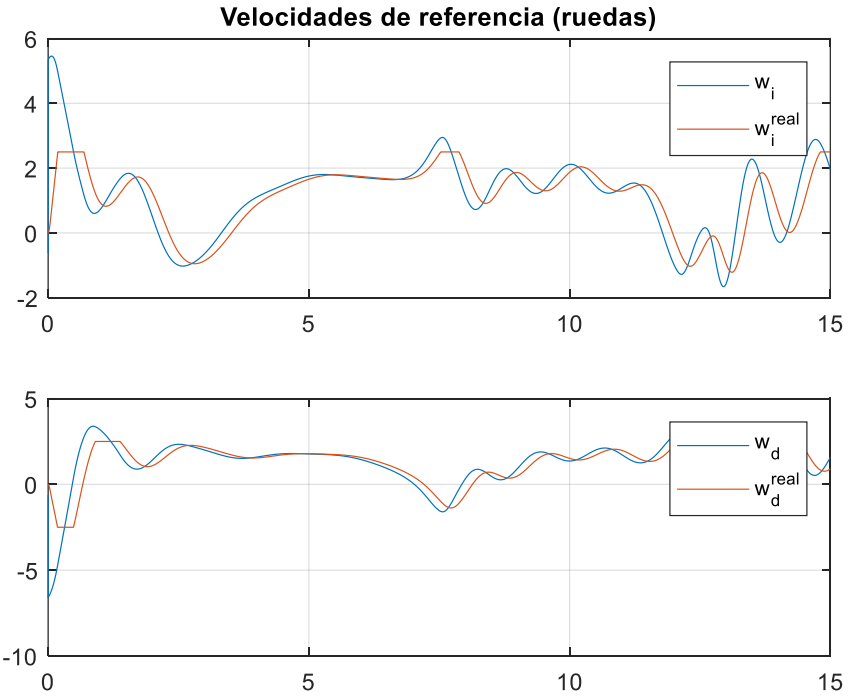

Figura 16. Velocidades necesarias para el seguimiento de la trayectoria senoidal.

Se observa dicha distancia mencionada anteriormente, pues la velocidad de las ruedas no llega exactamente a la referencia.

4 CONCLUSIONES GENERALES.

El simulador es demasiado ideal, por lo que, en un caso real, las prestaciones del control se degradarían hasta una posible inestabilidad en este, por lo que probablemente habría que reajustar las ganancias para un correcto funcionamiento del sistema. Algunas de las razones por las que se llega a esta conclusión son:

- Los radios de las ruedas son exactamente iguales, en un caso real habría cierto margen de fabricación que en simulación no se ha tenido en cuenta.
- Los ejes de las ruedas están perfectamente alineados.
- No se ha tenido en cuenta el deslizamiento de estas con el suelo.
- La dinámica de los actuadores es la misma en los dos, en un caso real habría una mínima diferencia.
- Por consecuencia de todos los factores anteriores, las ruedas giran exactamente lo mismo en simulación, pero en un caso real no sería así.
- Conocimiento exacto de la posición y orientación, sin margen de error.
- Medidas perfectas (sin ruidos).
- Sin derivas al integrar la velocidad.

Según se ha experimentado, puesto que en varios casos el sistema se volvía inestable para alcanzar ciertas posiciones, se determina que el sistema es altamente sensible a la ganancia que le aportemos.

Por último, añadir que el algoritmo de persecución pura es el más sencillo y efectivo de los trabajados, adaptable a cualquier tipo de trayectorias, siempre que dispongamos de un generador que nos la proporcione debidamente.