Universidad de Alcalá

Práctica 1. Identificación y control neuronal (II)

Sistemas de Control Inteligente

Grado en Ingeniería Computadores
Grado en Ingeniería Informática
Grado en Sistemas de Información
Universidad de Alcalá

Diseño de un control de posición mediante una red neuronal no recursiva.

En esta práctica, se estudiará el diseño de controladores neuronales para el control de posición y seguimiento de trayectorias de robots móviles. Se hará uso del entorno Matlab para el entrenamiento de las redes neuronales y el entorno Simulink para la simulación del robot móvil y los controladores propuestos. Afrontaremos en esta sesión el diseño de un control de posición mediante una red neuronal no recursiva para abordar en la siguiente el diseño de un control para seguimiento de trayectorias mediante redes recurrentes.

1. Objetivo y descripción del sistema.

En este apartado se plantea el control de posición de un robot móvil dentro de un entorno sin obstáculos de dimensiones 10x10 metros y cuyo origen de coordenadas se encuentra en el centro geométrico del entorno.

El objetivo de esta parte de la práctica es diseñar un controlador neuronal que sea capaz de "emular" el comportamiento de un controlador de tipo "caja negra" y cuyo bloque de Simulink es proporcionado junto con la práctica (controlblackbox.slx).

El esquema de control propuesto se describe mediante el siguiente diagrama de bloques en el entorno Matlab/Simulink:

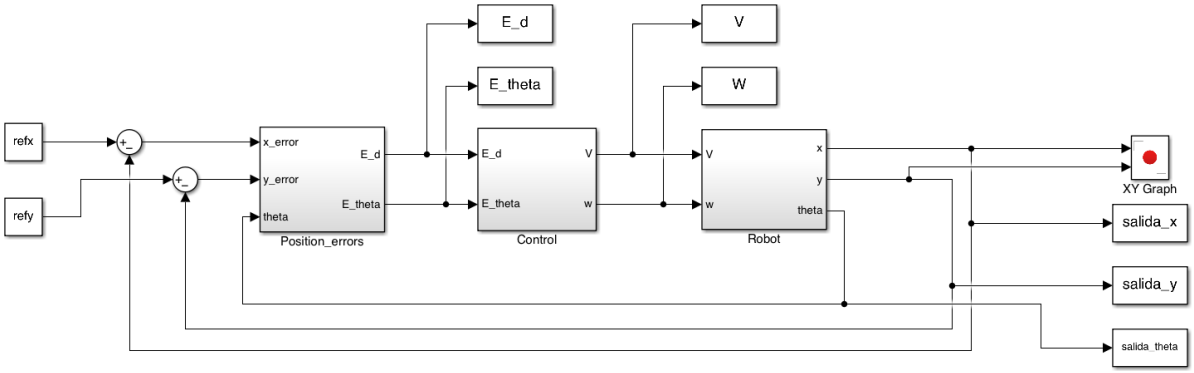


Figura 1. Esquema general de un control de posición.

A continuación, se describen los distintos subsistemas que componen este esquema.

1.1. Subsistema que modela el comportamiento de robot: Robot.

Este subsistema modela el comportamiento del robot móvil y es el diseñado en el segundo apartado de la práctica 0.

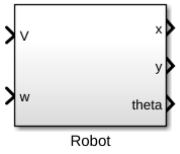


Figura 2. Subsistema Robot diseñado en la práctica 0.

Como se describió en la práctica anterior, el movimiento del robot se rige por el siguiente modelo dinámico:

$$x_k = x_{k-1} + V_{k-1}\mathrm{T_s}\cos(\theta_{k-1})$$
$$y_k = y_{k-1} + V_{k-1}T_s\sin(\theta_{k-1})$$
$$\theta_k = \theta_{k-1} + \omega_{k-1}T_s$$

donde $\mathrm{T_s}$ es el tiempo de muestreo utilizado en el sistema, ω_k es la velocidad angular del robot y $\mathrm{V_k}$ es la velocidad lineal.

1.2. Subsistema que calcula el error en la posición del robot respecto a una consigna dada: Position_errors.

La posición destino u objetivo del robot se denotará mediante las coordenadas $refx, refy$ (bloques constantes del mismo nombre, *refx* y *refy*, del esquema general de control de la Figura 1).

Para realizar el control de posición del robot se calculan dos errores:

- Error de distancia al objetivo: $E_d = sqrt((refx - x_k)^2 + (refy - y_k)^2)$
- Error de ángulo al objetivo: $E_\theta = atan2(refy - y_k, refx - x_k) - \theta_k$

donde *atan2* representa la función arcotangente que es sensible a los diferentes cuadrantes del ángulo:

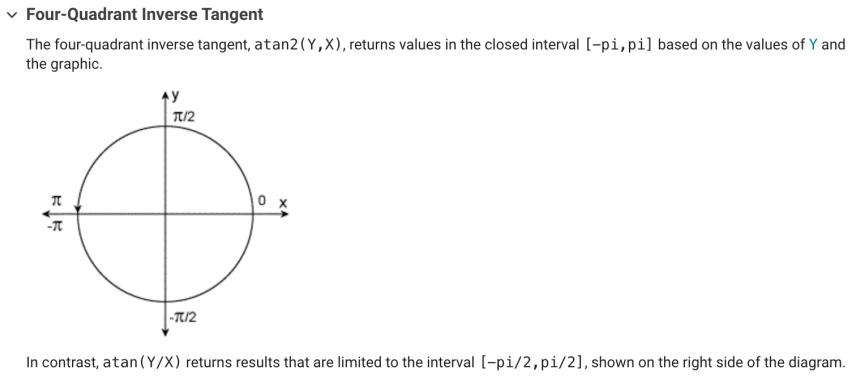


Figura 3. Ayuda de Matlab sobre *atan2*

Estos errores se calcularán en Simulink mediante el siguiente bloque:

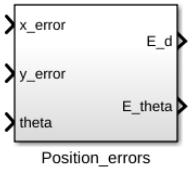


Figura 4. Subsistema para el cálculo del error de la posición del robot.

Donde *x_error* = $refx - x_k$ e *y_error* = $refy - y_k$.

Se deberá añadir una comprobación para que *E_theta* siempre tenga un valor comprendido en el intervalo [-π,π] y, en caso de que el ángulo esté fuera de estos límites, se deberá realizar la transformación del ángulo obtenido (*E_theta*) a su ángulo equivalente dentro del intervalo [-π,π].

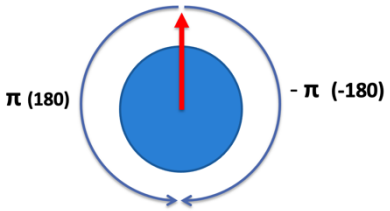


Figura 4. Sistema de coordenadas del robot

Este bloque deberá tener además una *condición de parada* de la simulación cuando el error de distancia *E_d* sea inferior a 0.01 metros. Para ello, se utilizarán los siguientes bloques conectados a *E_d* (véase Figura 5) dentro del subsistema anterior.

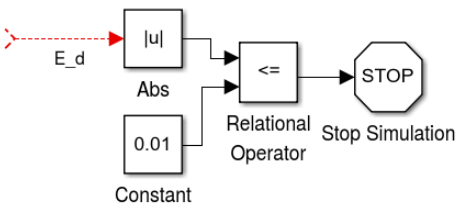


Figura 5. Condición de parada

1.3. Subsistema que implementa el controlador: Control.

Los controladores diseñados y utilizados en esta práctica se definen mediante bloques de Simulink que tienen como entrada los errores de distancia y ángulo comentados anteriormente y producen a la salida los valores de velocidades (angular y lineal) de entrada en el bloque de simulación del robot.

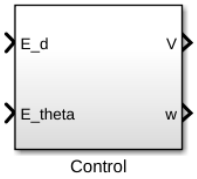


Figura 6. Subsistema de control.

Este controlador de posición de tipo “caja negra” se pone a disposición de los alumnos junto con el enunciado (“controlblackbox.slx”).

1.4. Bloques de E/S.

En el esquema de la Figura 1, se dispone de un bloque de visualización de la trayectoria efectuada por el robot en el plano X-Y cuyas características y resultado se muestra en la Figura 7.

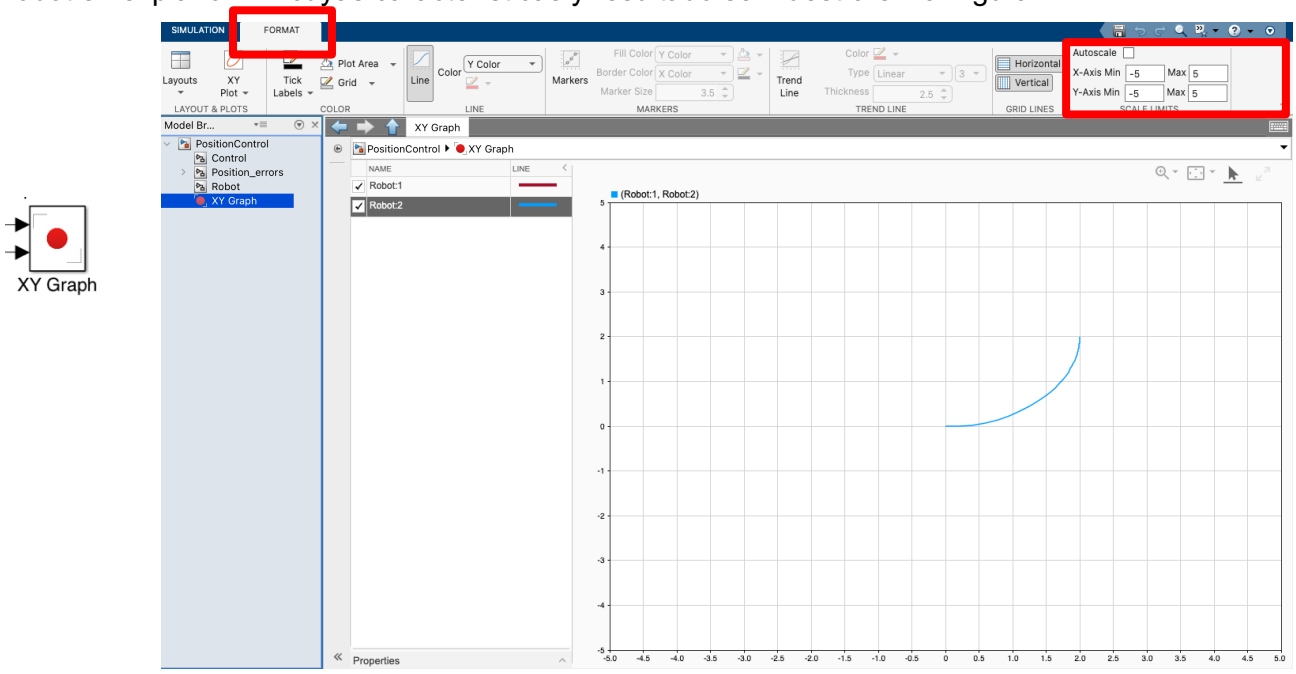
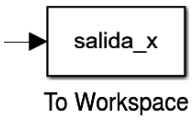


Figura 7. Bloque de visualización de la trayectoria.

También se dispone de bloques para exportar señales de la simulación al Workspace del entorno de Matlab. Se utilizará el formato “*Structure With Time*”. A continuación, se muestra el bloque que genera la coordenada x del robot.



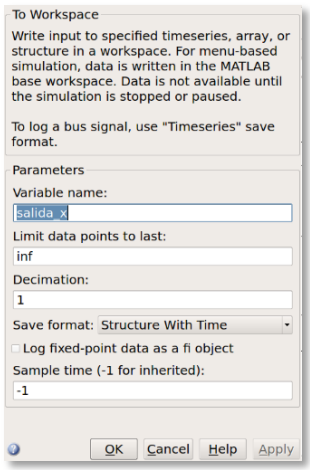


Figura 8. Salida de datos al Workspace.

Para que estas señales se exporten de acuerdo con el código que se utiliza posteriormente, es necesario desactivar la línea "*single simulation output*" dentro de las opciones de configuración del modelo ("*Model Settings*") en su apartado "*Data Import/Export*".

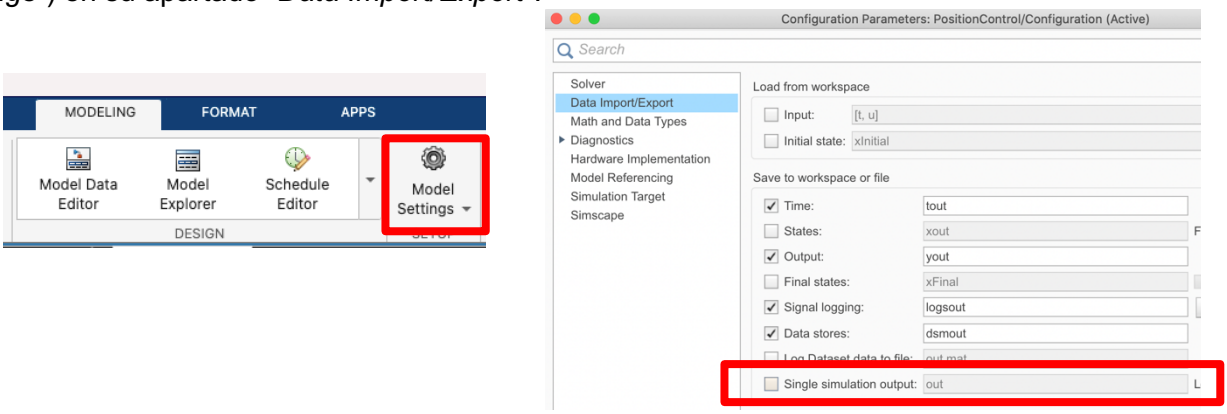


Figura 9. Desactivar la opción "Single simulation output"

2. Desarrollo de la práctica

Como se indicó al comienzo del documento, el objetivo de esta primera parte de la práctica es diseñar un controlador basado en una red neuronal que emule el comportamiento del controlador caja negra ("controlblackbox.slx"). Para ello se piden los siguientes apartados:

a) Implemente el esquema de la Figura 1, incluyendo todos los bloques principales. Utilice como controlador el proporcionado en "controlblackbox.slx". Configure los parámetros de la simulación (menú "Simulation/Model Simulation Parameters") tal y como se muestra en la Figura 10. Guarde el esquema de Simulink con un nombre reconocible, por ejemplo "PositionControl.slx".

Figura 10. Parámetros de la simulación.

b) En el entorno de programación de Matlab, cree un script nuevo con el nombre *"RunPositionControl.m"*. Con el siguiente código se puede simular el diagrama *PositionControl* desde el entorno de comandos de Matlab, configurando el punto destino del robot mediante las variables refx y refy y el tiempo de muestreo Ts.

```
%Tiempo de muestreo
Ts=100e-3
% Referencia x-y de posicion
refx=2.0;
refy=2.0;
% Ejecutar Simulacion
sim('PositionControl.slx')
```

c) Ejecute el script *RunPositionControl* y compruebe que se generan las variables que contienen las salidas y entradas del controlador (variables *E_d E_theta V* y *W*) y las salidas del robot durante la simulación (*salida_x, salida_y, salida_theta*).

d) Ejecute el siguiente código para mostrar la trayectoria del robot mediante el comando plot de Matlab

```
% Mostrar
x=salida_x.signals.values;
y=salida_y.signals.values;
figure;
plot(x,y);
grid on;
hold on;
```

e) Realice N=30 simulaciones del controlador proporcionado mediante un bucle donde se varían los valores de *refx* y *refy* de manera aleatoria dentro del entorno de 10x10 metros. En cada simulación se deberá guardar el valor a lo largo del tiempo de las entradas (*E_d* y *E_theta*) y salidas (*V* y *W*) del bloque controlador. Genere la matriz "*inputs*" de tamaño 2XN, donde se acumulen los valores de *E_d* y *E_theta*. Del mismo, modo genere la matriz "*outputs*", donde se acumulen los valores obtenidos de las variables *V* y *W*.

```
% Generar N posiciones aleatorias, simular y guardar en variables
N=30
E_d_vec=[];
E_theta_vec=[];
V_vec=[];
W_vec=[];
for i=1:N
        refx=10*rand-5;
        refy=10*rand-5;
        sim('PositionControl.slx')
        E_d_vec=[E_d_vec;E_d.signals.values];
        E_theta_vec=[E_theta_vec;E_theta.signals.values];
        V_vec=[V_vec; V.signals.values];
        W_vec=[W_vec; W.signals.values];
end
inputs=[E_d_vec'; E_theta_vec'];
outputs=[V_vec'; W_vec'];
```

f) Diseñe una red neuronal con una capa oculta de tal manera que dicha red se comporte como el controlador proporcionado. El número de neuronas de la capa oculta se deberá encontrar mediante experimentación. Justifique el valor elegido. El siguiente ejemplo muestra los comandos de Matlab para realizar dicho entrenamiento a partir de los vectores *inputs*, *outputs*.

```
% Entrenar red neuronal con 10 neuronas en la capa oculta
net = feedforwardnet([10]);
net = configure(net,inputs,outputs);
net = train(net,inputs,outputs);
```

Discuta los resultados del entrenamiento y observe el comportamiento de la red en los subconjuntos de entrenamiento, test y validación.

g) Genere, mediante la orden *gensim* de Matlab, un bloque con la red neuronal propuesta:

```
% Generar bloque de Simulink con el controlador neuronal
gensim(net,Ts)
```

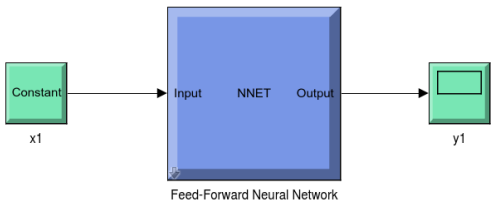


Figura 11. Bloque generado por gensim.

h) Cree un nuevo archivo de simulación “PositionControlNet.slx” con los mismos bloques utilizados en “PositionControl.slx” y donde se utilice la red neuronal en lugar del bloque controlador (Figura 12). Se utilizan bloques multiplexores y demultiplexores para adaptar las señales de entrada y salida como se muestra en la Figura.

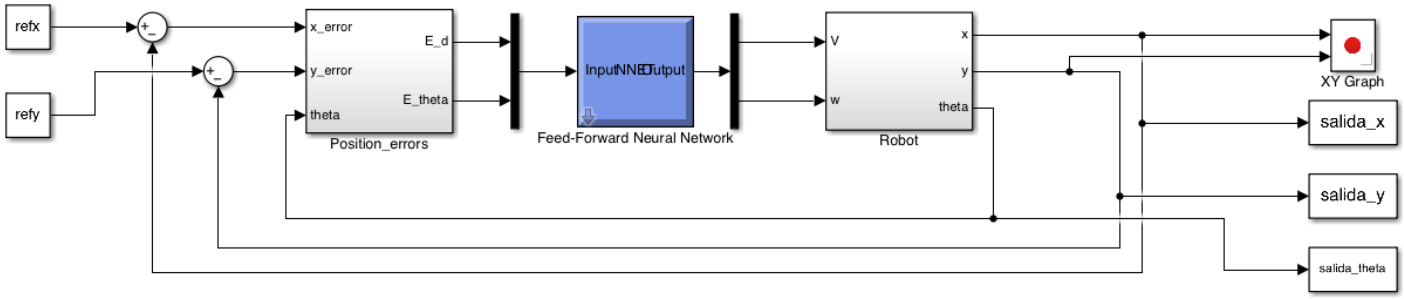


Figura 12. Esquema del controlador neuronal de posición.

i) Compare el comportamiento de “PositionControlNet.slx” con “PositionControl.slx” para diferentes valores de refx y refy. Calcule el error entre las trayectorias realizadas por ambos controladores. Se recomienda realizar un script de Matlab para automatizar dicha comparación y mostrar los resultados.