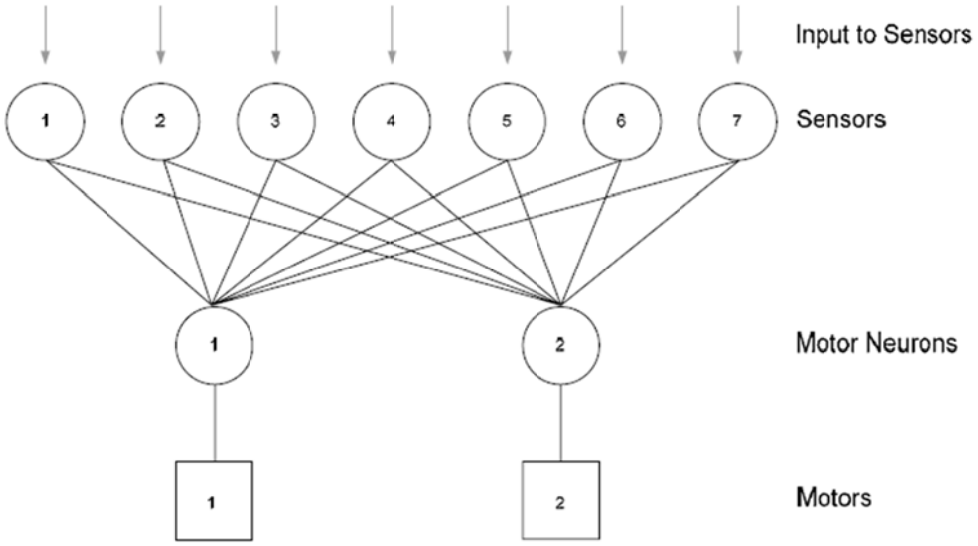
# Ejemplos de sistemas CTRNNs




## Ecuaciones

$$\frac{\partial y}{\partial t}_i = \frac{1}{\tau_i}(-y_i + I_i) \qquad \text{for the sensory neurons } (i = 1, \ldots 7)$$

$$\frac{\partial y}{\partial t}_i = \frac{1}{\tau_i}\left[-y_i + \sum_{j=1}^{7} w_{ij}\sigma(g_j(y_j + \theta_j))\right] \qquad \text{for the motor neurons } (i = 1, 2)$$

$$y_i = \sigma(g_j(y_j+\theta_j)) \qquad \text{for the outputs to the motors } (i = 1, 2)$$

Donde

$$\sigma(x) = \frac{1}{1 + e^{-g(x+\theta)}}$$

## Parámetros iniciales

y(0)=0 (condiciones iniciales)

h (euler step)= 0.1

tau=1

# Configuración de parámetros

Individuo= [w11, w21, w31, w41, w51, w61, w71, w12, w22, w23, w24, w25, w26, w27, Q1, Q2, t1, t2, t3, t4, t5, t6, t7, t8, t9, g1, g2,….]

Población= num de individuos: 20 (obtenidos por configuraciones random en los intervalos weights ∈[-6,6], biases ∈[-10,10] and gains ∈[-10,10]).

# Función fitness

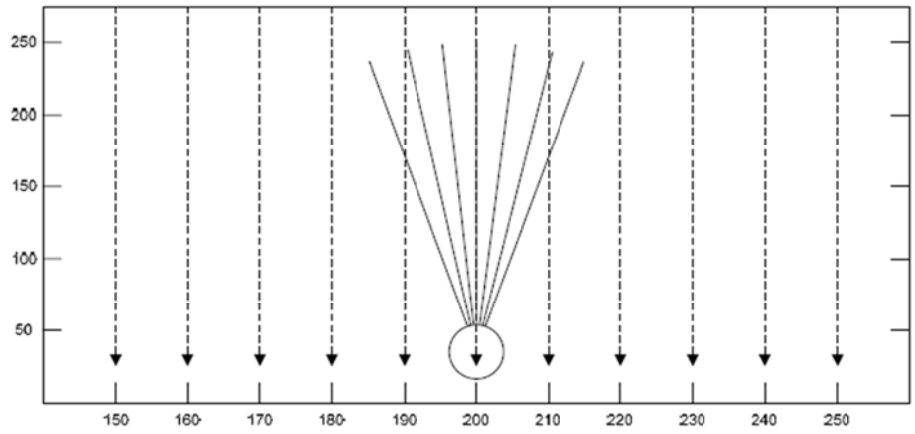
Función de éxito (según el caso)

# Evolución genética

- Genetic Algorithm toolbox (combinaciones cruzadas, etc.)
- Gaussian mutation function (*ConstrainedGaussianMutation.m)*

# Caso 1:

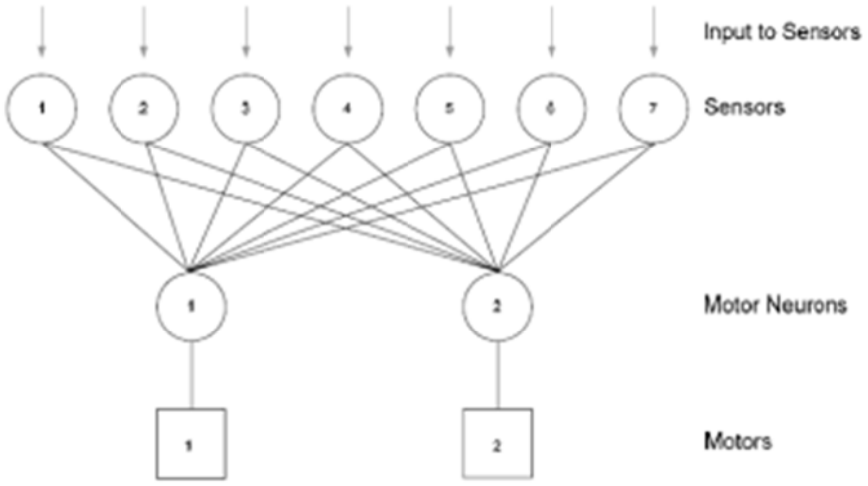
Objeto móvil que captura objetos




***Función fitness:*** minimizar la distancia final d

$$\sum_{i=1}^{NumTrials} d_i \Big/ NumTrials$$

***Red CTRNN***




***Parámetros:*** (red simétrica→ 10 parámetros)

Individuo= [w11, w21, w31, w41, w51, w61, w71, → sensores

g1, Q1 →neuronas motoras

Q'1] → motores

***Configuración de parámetros***

*Población inicial: 25 individuos random (***Función randn de Matlab)**

***Evolución:***

*Evolución: 200 generaciones*

1. Genetic Algorithm toolbox, para combinar padres y generar hijos
2. function mutationChildren, donde un elemento es mutado añadiendo un desplazamiento random con centro en él, y una distribución gaussiana a cualquier dirección. El resultado es seleccionado si su performance es mayor que la del padre.

*ConstrainedGaussianMutation.m*

```
function mutationChildren =
ConstrainedGaussianMutation(parents,options,GenomeLength,FitnessFcn,state,thisScore,th
isPopulation,scale,shrink)
```

*Valores frontera*

```
% set boundary constraints (this part changed when there were more parameters being
evolved)
LB = [-6    -6    -6    -6    -6    -6    -6    -10   -10   -10];
UB = [ 6     6     6     6     6     6     6     10    10    10];
```

*Parámetros de mutación*

*Gaussiana con media cero y desviación típica 1*

```
% set scale of Gaussian
scale = 1;

C = zeros(1,length(LB));

for i=1:length(parents)

    %Choose old parent
    parent = thisPopulation(parents(i),:);
    %Generate random vector
    A = scale .* randn(1,length(parent)) – scale / 2 + parent;
    %'Reflect' values which are over constraints
    D = A + max(LB - A, C) .* 2 + min(UB - A, C) .* 2;
    %Ensure this doesn't make it go over other constraint boundary
    mutationChildren(i,:) = min(max(D,LB),UB);

End
```

Se elige un padre
Se genera un vector random
Se mezcla
Se asegura que está entre los extremos

## ***Performance del objeto móvil:*** *OrientationEvolve.m*

```
function performance = OrientationEvolve(input)

% translate inputs
w(1,:) = input(1:7);
g_sensors = input(8);
bias_sensors = input(9);
bias_motors = input(10);

y_agent = 0;
X_Start = 160:20:240;

%set parameters
h=0.1;
tau = 1;
dia_obj = 26;
dia_agent = 30;
gain_motors = 1;

for a = 1:7
    w(2,a) = w(1,8-a);
end

%Initialise Sensor Neurons
sensor_old = zeros(7,1);
motorNeuron_oldA=0;
motorNeuron_oldB=0;

NumberTrials = 5;

y_obj_velocity = -10;
x_obj_velocity = 0;

for TrialNumber = 1:NumberTrials
%initialise locations/velocities
x_obj = X_Start(TrialNumber);
y_obj = 275;
x_agent = 200;

    while (y_obj - dia_obj/2) > 0

        % Get Sensor values
        I = sensor_intensity(x_obj, y_obj, x_agent, y_agent, dia_obj, dia_agent);

        %Update Sensor Neurons
        for a =1:7
            sensor_new(a) = euler_sensor(sensor_old(a),h,tau,I(a));
        end

        %Update motor Nuerons
        motorNeuron_newA =
       euler_motor(motorNeuron_oldA,sensor_new,h,tau,w(1,:),bias_sensors,g_sensors);
        motorNeuron_newB =
        euler_motor(motorNeuron_oldB,sensor_new,h,tau,w(2,:),bias_sensors,g_sensors);

        %Update motors
        motorA = sigma(motorNeuron_newA,bias_motors,gain_motors);
        motorB = sigma(motorNeuron_newB,bias_motors,gain_motors);

        %Move Agent
        x_agent_velocity = (motorA - motorB) * 5;
        x_agent = x_agent + h * x_agent_velocity;

        %Move Object
        x_obj = x_obj + h * x_obj_velocity;
        y_obj = y_obj + h * y_obj_velocity;

        sensor_old = sensor_new;
        motorNeuron_oldA = motorNeuron_newA;
        motorNeuron_oldB = motorNeuron_newB;
    end

    distance(TrialNumber) = abs(x_obj - x_agent);
    end
 performance = sum(distance) / NumberTrials;
```

Todos los parámetros están en el vector input
El valor inicial de y=0
El valor inicial de las posiciones de los objetos que caen

Parámetros como el paso de integración (0.1), el tiempo tau (1), el diámetro del objeto (26), del agente (30) y la ganancia de los motores (1).

Pesos de la neurona 2 (simétrica a la neurona 1)

Valores iniciales en los sensores del agente (cero señal) y en motores (cero movimiento)
Valores iniciales en los objetos que caen
Num de trials en el que caen objetos: 5

Para cada trial:
- El agente está en la posición inicial (200)
- Los objetos salen aleatoriamente desde la posición 275 (en 250 empiezan a detectarse)

Mientras el objeto no haya llegado al suelo (altura menos diámetro):
- Calculamos intensidad
- Actualizamos neuronas sensoras

- Actualizamos neuronas motoras
- Actualizamos motores
- Movemos el agente (diferencia de velocidades por un factor multiplicador igual a 5)
- Nueva posición del agente (antigua posición más velocidad por tiempo de paso de integración)
- Nueva posición del objeto que cae

Para medir el éxito, al finalizar el while calculamos la distancia final. La performance hará una media de los 5 trials.

```
function [sensor_intensity] = sensor_intensity(x_obj, y_obj, x_agent, y_agent,
dia_obj, dia_agent)

rad_obj = dia_obj/2;
rad_adj = dia_agent/2;
Y = y_obj - y_agent - rad_obj;
X1 = x_agent - x_obj - rad_obj * 0.7071;
X2 = x_agent - x_obj + rad_obj * 0.7071;

tanTheta(1) = 0.2679;
tanTheta(2) = 0.1317;
tanTheta(3) = 0.0875;
tanTheta(4) = 0;
tanTheta(5) = -0.0875;
tanTheta(6) = -0.1317;
tanTheta(7) = -0.2679;

cosTheta(1) = 0.9659;      Theta=15º
cosTheta(2) = 0.9914;      Theta=7,5º
cosTheta(3) = 0.9962;      Theta=3,75º
cosTheta(4) = 1;           Theta=0º
cosTheta(5) = 0.9962;
cosTheta(6) = 0.9914;
cosTheta(7) = 0.9659;

funcSinTanTheta(1) = 0.1986;
funcSinTanTheta(2) = 0.1145;
funcSinTanTheta(3) = 0.0799;
funcSinTanTheta(4) = 0;
funcSinTanTheta(5) = -0.0951;
funcSinTanTheta(6) = -0.1488;
funcSinTanTheta(7) = -0.3373;

for a = 1:7
    %if (tan(theta(a)) > X1/Y) & (tan(theta(a)) < X2/Y)
     if (tanTheta(a) > X1/Y) & (tanTheta(a) < X2/Y)
        %can see object
        %Distance = (Y / cos(theta(a))) + (dia_obj * (1-sin(theta(a))) *
tan(theta(a))) - dia_agent/2;
        Distance = (Y / cosTheta(a)) + (dia_obj * funcSinTanTheta(a)) - rad_adj;
        if Distance <= 0
            %object touching or 'inside' agent
            sensor_intensity(a) = 10;
        else if Distance < 221
                sensor_intensity(a) = min(10, 10/Distance);
            else
                %object too far away
                sensor_intensity(a) =0;
            end
        end
   else
        %can't see object
        sensor_intensity(a) = 0;
    end
end
```

Para calcular la intensidad que le llega al sensor (proporcional a la distancia del objeto)
Son 7 rayos de longitud máxima 220, uniformemente distribuidos en ángulos π/6 (30º)

Parámetros:
Cos (45º): 0.7071
Theta: 15º, 7,5º, 3,75º, y 0º

¿por qué?
Por cuestiones de geometría, sólo podrán ser vistos los objetos por el agente cuando
`tan(theta(a)) > X1/Y) & (tan(theta(a)) < X2/Y`

En ese caso, calculamos la distancia
`Distance = (Y / cosTheta(a)) + (dia_obj * funcSinTanTheta(a)) - rad_adj`

En función de la distancia se calcula la intensidad:
Caso 1: agente y objeto se tocan
Caso 2: agente ve a objeto
Caso 3: objeto muy lejano
Caso 4: objeto ya ha caído sin impactar con el agente / aún no ha llegado a su zona de visibilidad (?)

```
function [y_new_sensor] = euler_sensor(y_old_sensor, h, tau, I)
y_new_sensor = y_old_sensor + (h/tau) * (I - y_old_sensor);
```

$$\frac{\partial y}{\partial t}_i = \frac{1}{\tau_i}(-y_i + I_i)$$

Actualizamos la señal en las neuronas sensoras en el paso siguiente

```
function [y_new_motor] = euler_motor(y_old, sensor_row, h, tau, w_row, bias,gain)
W = 0;
for a = 1:7
      W = w_row(a) * sigma(sensor_row(a),bias,gain) + W;
end
y_new_motor = y_old + (h/tau) * (-y_old + W );

function [sigma] = sigma(y,bias,gain)
sigma = 1 / (1+exp( (- y - bias)*gain) );
```

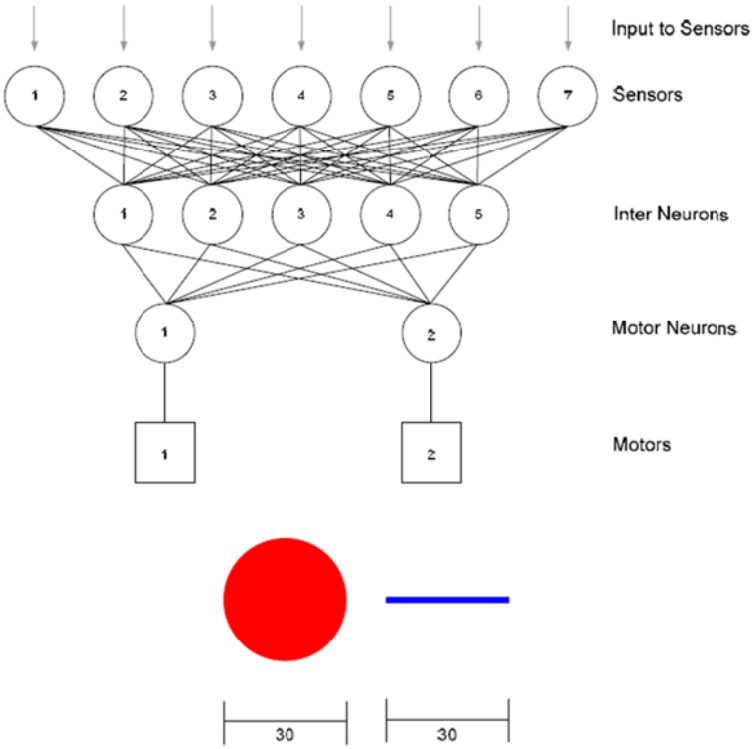
Actualizamos la señal en las 2 neuronas motoras

$$\frac{\partial y}{\partial t}_i = \frac{1}{\tau_i}\left[-y_i + \sum_{j=1}^{7} w_{ij}\sigma(g_j(y_j + \theta_j))\right]$$

Definimos un vector W que es el sumatorio de la expresión (multiplica pesos por la señal procesada por su función de activación gaussiana)

# Caso 2:

- Objeto móvil que captura objetos circulares y huye de objetos-líneas.
- La red neuronal necesita ser más compleja (capa intermedia y sin simetría)




*ObjectDiscriminateInterNeuronsEvolve.m*

```
function performance = ObjectDiscriminateInterNeuronsEvolve(input)

% translate inputs
WIn(1,:) = input(1:7);
WIn(2,:) = input(8:14);
WIn(3,:) = input(15:21);
WOut(1,:) = input(22:26);
WOut(2,:) = input(27:31);
g_sensors = input(32);
bias_sensors = input(33);
g_interneurons = input(34:38);
bias_interneurons = input(39:43);
bias_motors(1) = input(44);
bias_motors(2) = input(45);

y_agent = 0;
X_Start = 140:10:240;

%set parameters
h=0.1;
tau = 1;
dia_obj = 30;
dia_agent = 30;
gain_motors = 1;

%Bilateral symmetry
for a = 1:7
    WIn(5,:) = WIn(1,8-a);
    WIn(4,:) = WIn(2,8-a);
end
```

Parámetros libres a evolucionar: de los 45 en total

- Bias and gain: 2 (neuronas sensoras iguales)
- Bias and gain: 10 (inter-neuronas diferentes)
- WIN: Pesos simétricos: 21 (de los 35, dos inter-neuronas son iguales)
- WOUT: Pesos no simétricos: 10 ( de las interneuronas a las motoras)
- Bias motores: 2 (gain motores=1)

Parámetros iniciales

Obtención de parámetros por las condiciones de simetría

Distinguimos dos objetos:

- Obj_type=0 (el círculo)
- Obj_type=1 (la línea)

```
%Initialise Sensor Neurons
sensor = zeros(7,1);
interNeuron = zeros(5,1);
motorNeuronA=0;
motorNeuronB=0;

NumberTrials = 9;

y_obj_velocity = -5;
x_obj_velocity = 0;

for TrialNumber = 1:9

    obj_type = 0;

    %initialise locations/velocities
    x_obj = X_Start(TrialNumber);
    y_obj = 275;
    x_agent = 200;

    while (y_obj - dia_obj/2) > 0

        % Get Sensor values
        I = sensor_intensity(x_obj, y_obj, x_agent, y_agent, dia_obj,
dia_agent,obj_type);

        %Update Sensor Neurons
        for a =1:7
            sensor(a) = euler_sensor(sensor(a),h,tau,I(a));
        end

        %update inter Neurons
        for i = 1:5
            interNeuron(i) =
euler_IN(interNeuron(i),sensor,h,tau,WIn(i,:),bias_sensors,g_sensors);
        end

        %Update motor Neurons
        motorNeuronA =
euler_motor(motorNeuronA,interNeuron,h,tau,WOut(1,:),bias_interneurons,g_interneurons)
;


        motorNeuronB =
euler_motor(motorNeuronB,interNeuron,h,tau,WOut(2,:),bias_interneurons,g_interneurons)
;

        %Update motors
        motorA = sigma(motorNeuronA,bias_motors(1),gain_motors);
        motorB = sigma(motorNeuronB,bias_motors(2),gain_motors);

        %Move Agent
        x_agent_velocity = (motorA - motorB) * 5;
        x_agent = x_agent + h * x_agent_velocity;

        %Keep agent within bounds of environment
        x_agent = max(0,x_agent);
        x_agent = min(400,x_agent);

        %Move Object
        %x_obj = x_obj + h * x_obj_velocity;
        y_obj = y_obj + h * y_obj_velocity;

    end

    distance(TrialNumber,1) = abs(x_agent - x_obj);
```

---

```
    %initialise locations/velocities
    x_obj = X_Start(TrialNumber);
    y_obj = 275;
    x_agent = 200;
    sensor = zeros(7,1);
    interNeuron = zeros(5,1);
    motorNeuronA=0;
    motorNeuronB=0;

    obj_type = 1;

    while (y_obj - dia_obj/2) > 0
```

```
        % Get Sensor values
        I = sensor_intensity(x_obj, y_obj, x_agent, y_agent, dia_obj,
dia_agent,obj_type);

        %Update Sensor Neurons
        for a =1:7
            sensor(a) = euler_sensor(sensor(a),h,tau,I(a));
        end

        %update inter Neurons
        for i = 1:5
            interNeuron(i) =
euler_IN(interNeuron(i),sensor,h,tau,WIn(i,:),bias_sensors,g_sensors);
        end

        %Update motor Neurons
        motorNeuronA =
euler_motor(motorNeuronA,interNeuron,h,tau,WOut(1,:),bias_interneurons,g_interneurons)
;
        motorNeuronB =
euler_motor(motorNeuronB,interNeuron,h,tau,WOut(2,:),bias_interneurons,g_interneurons)
;

        %Update motors
        motorA = sigma(motorNeuronA,bias_motors(1),gain_motors);
        motorB = sigma(motorNeuronB,bias_motors(2),gain_motors);

        %Move Agent
        x_agent_velocity = (motorA - motorB) * 5;
        x_agent = x_agent + h * x_agent_velocity;

        %Keep agent within bounds of environment
        x_agent = max(0,x_agent);
        x_agent = min(400,x_agent);

        %Move Object
        %x_obj = x_obj + h * x_obj_velocity;
        y_obj = y_obj + h * y_obj_velocity;

    end

    distance(TrialNumber,2) = min(abs(x_agent - x_obj),50);

end

performance = (sum(distance(:,1)) + 450 - sum(distance(:,2)))/4.5 - 100;
```

La performance es una solución de compromiso entre objetos circulares y lineas

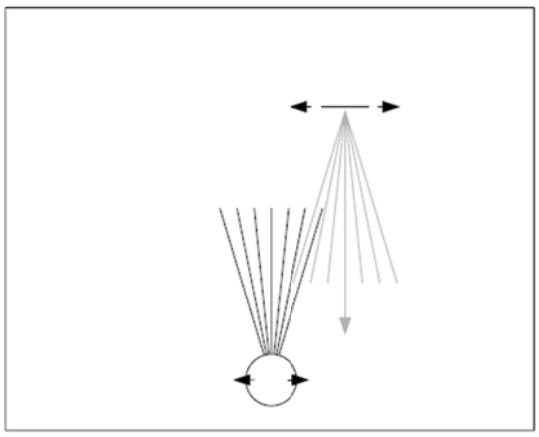
```
function [sensor_intensity] = sensor_intensity(x_obj, y_obj, x_agent, y_agent,
dia_obj, dia_agent, obj_type)

rad_obj = dia_obj/2;
rad_adj = dia_agent/2;
Y = y_obj - y_agent - rad_obj;

cosTheta(1) = 0.9659;
cosTheta(2) = 0.9914;
cosTheta(3) = 0.9962;
cosTheta(4) = 1;
cosTheta(5) = 0.9962;
cosTheta(6) = 0.9914;
cosTheta(7) = 0.9659;

tanTheta(1) = 0.2679;
tanTheta(2) = 0.1317;
tanTheta(3) = 0.0875;
tanTheta(4) = 0;
tanTheta(5) = -0.0875;
tanTheta(6) = -0.1317;
tanTheta(7) = -0.2679;

if obj_type == 0

    %object is a circle
    X1 = x_agent - x_obj - rad_obj * 0.7071;
    X2 = x_agent - x_obj + rad_obj * 0.7071;

    funcSinTanTheta(1) = 0.1986;
    funcSinTanTheta(2) = 0.1145;
    funcSinTanTheta(3) = 0.0799;
    funcSinTanTheta(4) = 0;
    funcSinTanTheta(5) = -0.0951;
    funcSinTanTheta(6) = -0.1488;
    funcSinTanTheta(7) = -0.3373;

    for a = 1:7
        %if (tan(theta(a)) > X1/Y) & (tan(theta(a)) < X2/Y)
         if (tanTheta(a) > X1/Y) & (tanTheta(a) < X2/Y)
            %can see object
            %Distance = (Y / cos(theta(a))) + (dia_obj * (1-sin(theta(a))) *
tan(theta(a))) - dia_agent/2;
            Distance = (Y / cosTheta(a)) + (dia_obj * funcSinTanTheta(a)) - rad_adj;
            if Distance <= 0
                %object touching or 'inside' agent
                sensor_intensity(a) = 10;
            else if Distance < 221
                    sensor_intensity(a) = min(10, 10/Distance);
                else
                     %object too far away
                     sensor_intensity(a) =0;
              end
            end
else
```

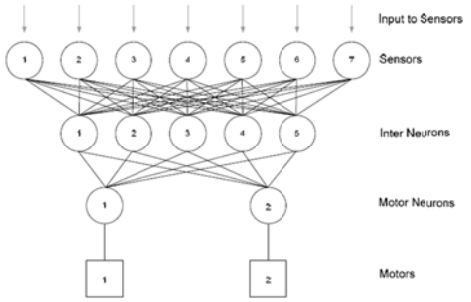
```
                %can't see object
                sensor_intensity(a) = 0;
                end
                end
        else
                %object is a line
                X1 = x_obj - x_agent + rad_obj;
                X2 = x_obj - x_agent - rad_obj;
                for a = 1:7
                        if (tanTheta(a) > X1/Y) & (tanTheta(a) < X2/Y)
                        Distance = (Y / cosTheta(a));
                        if Distance <= 0

                %object touching or 'inside' agent
                sensor_intensity(a) = 10;
                else if Distance < 221
                        sensor_intensity(a) = min(10, 10/Distance);
                else
                %object too far away
                sensor_intensity(a) = 10;
                end
                end
        else
                %can't see object
                sensor_intensity(a) = 0;
                end
                end

end

function [y_new_sensor] = euler_sensor(y_old_sensor, h, tau, I)
y_new_sensor = y_old_sensor + (h/tau) * (I - y_old_sensor);

function [y_new_motor] = euler_motor(y_old, interNeuron, h, tau, w_row, bias,gain)
W = 0;
for a = 1:5
     W = w_row(a) * sigma(interNeuron(a),bias(a),gain(a)) + W;
end

y_new_motor = y_old + (h/tau) * (-y_old + W );

function [y_new_IN] = euler_IN(y_old, sensor, h, tau, w_row, bias,gain)
W = 0;
for a = 1:7
    W = w_row(a) * sigma(sensor(a),bias,gain) + W;
end

y_new_IN = y_old + (h/tau) * (-y_old + W );

function [sigma] = sigma(y,bias,gain)
sigma = 1 / (1+exp( (- y - bias)*gain) );
```

Estas funciones actualizan la señal de activación de todas las neuronas

# Caso 3:

- Dos objetos con tareas opuestas






*DualAgentSystem.m*

```
function DualAgentSystem(input)
% translate inputs
WIn(1,:) = input(1:7);
WIn(2,:) = input(8:14);
WIn(3,:) = input(15:21);
WOut(1,:) = input(22:26);
WOut(2,:) = input(27:31);
g_sensors = input(32);
bias_sensors = input(33);
g_interneurons = input(34:38);
bias_interneurons = input(39:43);
bias_motors(1) = input(44);
bias_motors(2) = input(45);

y_agent = 0;
X_Start = 160:2:240;
L = length(X_Start);

%set parameters
h=0.1;
tau = 1;
dia_obj = 30;
dia_agent = 30;
gain_motors = 1;

%Bilateral symmetry
for a = 1:7
    WIn(5,:) = WIn(1,8-a);
    WIn(4,:) = WIn(2,8-a);
end

%Initialise Sensor Neurons
sensor_one = zeros(7,1);
sensor_two = zeros(7,1);
interNeuron_one = zeros(5,1);
interNeuron_two = zeros(5,1);
motorNeuronA_one=0;
motorNeuronB_one=0;
motorNeuronA_two=0;
motorNeuronB_two=0;

y_obj_velocity = -2.5; (se reduce en un factor 2)

for TrialNumber = 1:L
    %initialise locations/velocities
    x_obj = X_Start(TrialNumber);
    y_obj = 275;
    x_agent = 200;

    elapsedTime = 0 - h;
    g = 0;
```

Parámetros libres a evolucionar: de los 45 en total

- Bias and gain: 2 (neuronas sensoras iguales)
- Bias and gain: 10 (inter-neuronas diferentes)
- WIN: Pesos simétricos: 21 (de los 35, dos inter-neuronas son iguales)
- WOUT: Pesos no simétricos: 10 ( de las interneuronas a las motoras)
- Bias motores: 2 (gain motores=1).

- Posición inicial: agente en la línea de abajo (y=0) y en posiciones que van desde 160 al 240 (ver figura).
- Se define la longitud del espacio de movimiento como todos los puntos por los que puede valorar el agente (80 cm recorridos en saltos de 2: 160 puntos).
- Bilateral simetría: los pesos que faltan se copian de los conocidos
- Se definen las estructuras de datos (inicialmente a cero)

- Condiciones iniciales: el objeto en unos de los puntos para empezar a caer

```
    while (y_obj - dia_obj/2) > 0
        g = g + 1;
        elapsedTime = elapsedTime + h;

        % Get Sensor values
 I_one = sensor_intensity(x_obj, y_obj, x_agent, y_agent, dia_obj,
 dia_agent,1);
I_two_temp = sensor_intensity(-x_agent, -y_agent, -x_obj, -y_obj, dia_agent,
 dia_obj,1);
        for f = 1:7
            I_two(f) = I_two_temp(8-f);
        end

        %Update Sensor Neurons
        for a =1:7
            sensor_one(a) = euler_sensor(sensor_one(a),h,tau,I_one(a));
            sensor_two(a) = euler_sensor(sensor_two(a),h,tau,I_two(a));

        end

        %update inter Neurons
        for i = 1:5
            interNeuron_one(i) =
 euler_IN(interNeuron_one(i),sensor_one,h,tau,WIn(i,:),bias_sensors,g_sensors);
            interNeuron_two(i) =
 euler_IN(interNeuron_two(i),sensor_two,h,tau,WIn(i,:),bias_sensors,g_sensors);
        end

        %Update motor Neurons
        motorNeuronA_one =
 euler_motor(motorNeuronA_one,interNeuron_one,h,tau,WOut(1,:),bias_interneurons,g_inter
 neurons);
        motorNeuronB_one =
 euler_motor(motorNeuronB_one,interNeuron_one,h,tau,WOut(2,:),bias_interneurons,g_inter
 neurons);
        motorNeuronA_two =
 euler_motor(motorNeuronA_two,interNeuron_two,h,tau,WOut(1,:),bias_interneurons,g_inter
 neurons);
        motorNeuronB_two =
 euler_motor(motorNeuronB_two,interNeuron_two,h,tau,WOut(2,:),bias_interneurons,g_inter
 neurons);

        %Update motors
        motorA_one = sigma(motorNeuronA_one,bias_motors(1),gain_motors);
        motorB_one = sigma(motorNeuronB_one,bias_motors(2),gain_motors);
        motorA_two = sigma(motorNeuronA_two,bias_motors(1),gain_motors);
        motorB_two = sigma(motorNeuronB_two,bias_motors(2),gain_motors);

        %Move Agent
        x_agent_velocity = (motorA_one - motorB_one) * 5;
        x_agent = x_agent + h * x_agent_velocity;

        %Move Object
        x_obj_velocity = (motorA_two - motorB_two) * 5;
        x_obj = x_obj + h * x_obj_velocity;
        y_obj = y_obj + h * y_obj_velocity;

        %Keep agent within bounds of environment
        x_agent = max(0,x_agent);
        x_agent = min(400,x_agent);

        %Keep object within bounds of environment
        x_obj = max(0,x_obj);
        x_obj = min(400,x_obj);

        elapsedTimePlot(g) = elapsedTime;
        separation(TrialNumber,g) = x_obj - x_agent;

    end

 end

 hold on;
 for r = 1:L
     plot(separation(r,:),vertPosObject(r,:),'b');
 end

 hold off;
```

I_one corresponde al agente circulo (agente)
I_two corresponde al agente línea (objeto)