

Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

**Reporte de proyecto terminal 2**

**Autonomía para un robot excavador lunar**

Carlo Eduardo Campos Sandoval

208302026

Trimestre 2013 Invierno

Abril de 2013

Dr. Andrés Ferreyra Ramírez

M. en I. Juan Carlos Olguín Rojas

## Tabla de contenido

<b>Resumen</b> .....	<b>2</b>
<b>Objetivos</b> .....	<b>2</b>
Objetivo general .....	2
Objetivos Específicos .....	2
<b>Introducción</b> .....	<b>3</b>
<b>Planteamiento del problema</b> .....	<b>5</b>
<b>Desarrollo del proyecto</b> .....	<b>6</b>
Autonomía de movimiento usando una red neuronal artificial.....	6
Implementación de la red neuronal.....	7
Proceso de construcción del prototipo .....	18
<b>Conclusiones</b> .....	<b>20</b>
<b>Bibliografía</b> .....	<b>21</b>

## Resumen

El presente trabajo resuelve la autonomía de movimientos de un robot explorador lunar a escala, mediante algoritmos que involucran paradigmas de inteligencia artificial específicamente arquitecturas de redes neuronales artificiales con aprendizaje supervisado.

Se describe también las etapas de diseño y construcción del prototipo final, las cuales constan de: Diseño, entrenamiento y validación de la arquitectura de red neuronal propuesta, programación de la red en lenguaje C++ y en código ensamblador para el micro controlador 16F876A [4], construcción mecánica, diseño electrónico del prototipo. Finalmente se realizan las pruebas de desempeño y conclusiones finales.

## Objetivos

### *Objetivo General*

Diseñar e implementar el control de movimiento de un robot excavador lunar de un punto “A” a un punto “B” y de regreso evadiendo los obstáculos que se encuentren. Usando aprendizajes supervisados, basados en arquitecturas de redes neuronales artificiales.

### *Objetivos Específicos*

Estos son los problemas a solucionar para lograr la autonomía de movimiento:

- Programar un algoritmo de una arquitectura de red neuronal cuya capacidad de aprendizaje sea la que obtenga los mejores resultados en las simulaciones.
- Implementar ese algoritmo en un robot a escala para comprobar la capacidad de aprendizaje del algoritmo.

c) Implementar el algoritmo en el modelo final del robot excavador.

## Introducción

En este trabajo, se pretende que un robot pueda encontrar la salida dentro de un escenario lleno de obstáculos, determinando cual es la ruta óptima como consecuencia de un proceso de aprendizaje, para ello el robot podrá tomar distintos caminos hasta encontrar la salida.

Este tipo de problema generalmente es resuelto suministrándole al robot una base de datos que contiene todas las posibles situaciones que podrían presentarse y sus respectivas soluciones, pero como la cantidad de datos necesarios para especificar cada posible situación crece indefinidamente, conforme aumenta el número de sensores y posibles situaciones que deberá enfrentar el autómata, es necesario contar con dispositivos de gran capacidad de almacenamiento; por el contrario, una red neuronal puede entrenarse con un número representativo de patrones y aprender el comportamiento del sistema utilizando dispositivos de menor capacidad de almacenamiento y costo.

Para la autonomía de movimiento del robot explorador, se implementara un algoritmo basado en redes neuronales artificiales (RNA) y se programará para la autonomía de movimiento. Las redes neuronales artificiales simulan el comportamiento de las neuronas biológicas especialmente en el aprendizaje que realizan los seres humanos, en ese trabajo se realizaran y probaran los algoritmos de aprendizaje supervisado con arquitecturas de redes neuronales artificiales,

donde destacan:

El Perceptrón.- Esta arquitectura consiste en una red que es entrenada fuera de línea (Aprendizaje Supervisado) que puede clasificar patrones linealmente separables.

El algoritmo del Perceptrón, puede resumirse en los siguientes pasos:

1. Se inicializa la matriz de pesos y el valor de la ganancia, por lo general se asignan valores aleatorios a cada uno de los pesos  $w_i$  y al valor  $b$
2. Se presenta el primer patrón a la red, junto con la salida esperada en forma de pares entrada/salida
3. Se calcula la salida de la red por medio de:

$$a = f(w_1p_1 + w_2p_2 + b)$$

Donde  $f$  puede ser la función *hardlim* o *hardlims*

4. Cuando la red no retorna la salida correcta, es necesario alterar el valor de los pesos, tratando de llevarlo hasta  $p$  y así aumentar las posibilidades de que la clasificación sea correcta, una posibilidad es adicionar  $p$  a  $w$  haciendo que el vector  $w$  apunte en la dirección de  $p$ , y de esta forma después de repetidas presentaciones de  $p$  a la red,

$w$  se aproximará asintóticamente a  $p$ ; este es el procedimiento adoptado para la regla de aprendizaje del Perceptrón.

El proceso de aprendizaje del Perceptrón puede definirse en tres reglas, las cuales cubren la totalidad de combinaciones de salidas y sus correspondientes valores esperados. Estas reglas utilizando la función de transferencia hardlim, se expresan como sigue:

$$\text{Si } t = 1 \text{ y } a = 0, \text{ entonces } w_{1w}^{\text{nuevo}} = w_{1w}^{\text{anterior}} + p \quad (1.2)$$

$$\text{Si } t = 0 \text{ y } a = 1, \text{ entonces } w_{1w}^{\text{nuevo}} = w_{1w}^{\text{anterior}} - p \quad (1.3)$$

$$\text{Si } t = a, \text{ entonces } w_{1w}^{\text{nuevo}} = w_{1w}^{\text{anterior}} \quad (1.4)$$

Las tres condiciones anteriores pueden ser escritas en forma compacta y generalizarse para la utilización de las funciones de transferencia hardlim o hardlims, generalización que es posible introduciendo el error en las reglas de aprendizaje del Perceptrón:

$$e = t - a \quad (1.5)$$

Por lo tanto:

$$\text{Si } e = 1, \text{ entonces } w_{1w}^{\text{nuevo}} = w_{1w}^{\text{viejo}} + p \quad (1.6)$$

$$\text{Si } e = -1, \text{ entonces } w_{1w}^{\text{nuevo}} = w_{1w}^{\text{anterior}} - p \quad (1.7)$$

$$\text{Si } e = 0, \text{ entonces } w_{1w}^{\text{nuevo}} = w_{1w}^{\text{anterior}} \quad (1.8)$$

En una sola expresión la ley puede resumirse así:

$$w_{1w}^{\text{nuevo}} = w_{1w}^{\text{anterior}} + ep = w_{1w}^{\text{anterior}} + (t - a)p \quad (1.9)$$

Y extendiendo la ley a las ganancias

$$b^{\text{nueva}} = b^{\text{anterior}} + e \quad (1.10)$$

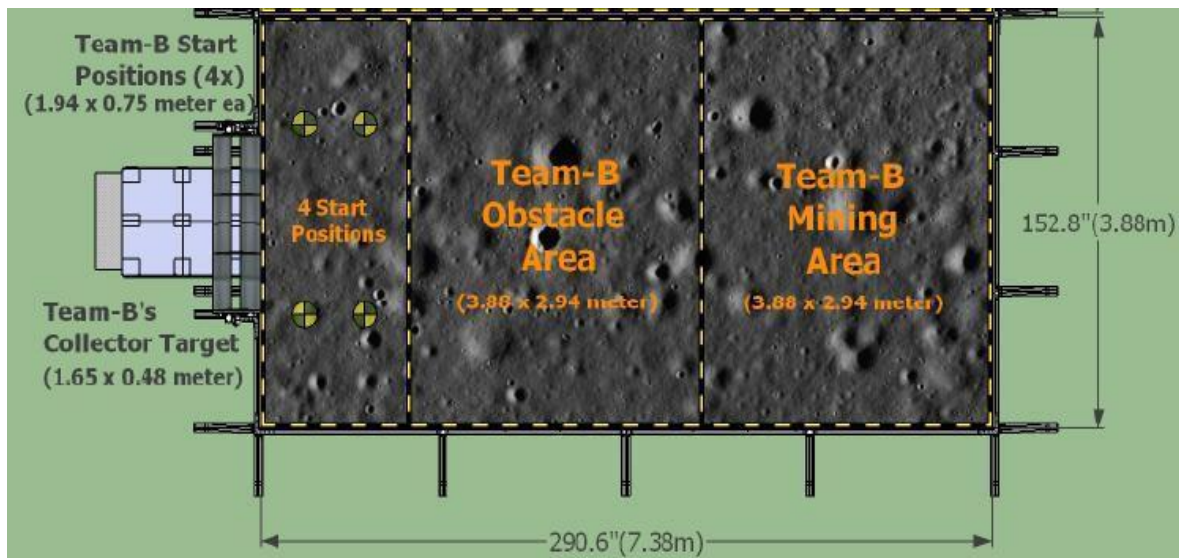
El perceptrón emplea principalmente dos funciones de transferencia, Función Escalón[8] con salidas 1 y 0, o Escalón Simétrica[8] con salidas 1 y -1; su uso depende del valor de salida que se espera para la red, es decir si la salida de la red es unipolar o bipolar; sin embargo hablando estrictamente del Robot la Función Escalón simétrica es preferida sobre

la Función Escalón, ya que al tener un cero multiplicando algunos de los valores resultantes del producto de las entradas por el vector de pesos, ocasiona que estos no se actualicen y que el aprendizaje sea más lento.

## Planteamiento del Problema

Se requiere diseñar y fabricar un robot explorador lunar con autonomía de movimiento, capaz de evadir obstáculos y recorrer un área especificada por las reglas definidas en la competencia LUNABOTICS PROJECT UAM 2012.

En esta competencia se trabaja sobre un escenario que recrea las condiciones reales de la luna, a este escenario se le denomina Lunarena (Ver Fig. 1)[2].



*Fig. 1: Vista de la Lunarena usada en la competencia[2].*

La competencia consiste en recolectar material llamado Regolit de la manera más eficiente, en una zona lunar simulada, donde se evalúan los siguientes parámetros:

- Cantidad de material recolectado
- Innovaciones en la Ingeniería
- Tele-operación
- Autonomía de excavación y movimientos
- Diseño mecánico y geométrico
- Espíritu de equipo

Evidentemente se trata de un problema multidisciplinario y en este trabajo nos enfocamos a resolver solamente la autonomía de movimientos del robot excavador.

## Desarrollo del proyecto

### *Autonomía de Movimiento usando una Red Neuronal Artificial*

Para realizar este análisis, debido a la naturaleza bipolar de la salida, se utiliza una arquitectura de red neuronal tipo perceptrón con función de activación a la salida *hardlims*, la razón es porque cuando los sensores detecten que el objeto se encuentra cerca, será representado por medio de un 1 y cuando se detecte que un objeto se encuentra a una distancia mayor que la predeterminada se dirá que el objeto está lejos lo cual se indica con un -1.

A la red se le presentaran inicialmente 7 patrones de carácter binario como se puede ver en la tabla 1, para los cuales dependiendo de las lecturas de los sensores se le indicará al robot qué hacer específicamente.

Es decir, los patrones de entrenamiento para la red serían:

S1	S2	S3	S4	M1	M2
-1	-1	-1	-1	1	1
-1	1	-1	-1	1	-1
1	-1	-1	-1	-1	1
1	-1	1	-1	1	-1
-1	-1	-1	1	-1	1
1	1	-1	-1	1	1
-1	-1	1	-1	1	-1

*Tabla 1. Patrones de entrenamiento para el aprendizaje supervisado.*

A la red neuronal se le presentan los siete patrones de la tabla 1, utilizamos una red neuronal con cuatro entradas para cada salida (ver fig.2).

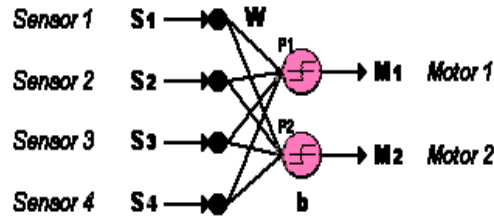


Fig. 2 Muestra una arquitectura neuronal de cuatro entradas y dos salidas

La siguiente tabla 2, indica la posición espacial de cada uno de los sensores y motores:

<i>Elemento</i>	<i>Ubicación</i>
Sensor1 Ultrasónico SRF-05	Sensor Izquierdo
Sensor2 Ultrasónico SRF-05	Sensor Derecho
Sensor3 Ultrasónico SRF-05	Sensor ubicado Adelante
Sensor4 Ultrasónico SRF-05	Sensor ubicado Atrás
Motor 1	Motor Izquierdo
Motor 2	Motor Derecho

### **Implementación de la red neuronal**

Se empezó programando la red neuronal en el lenguaje c++, el funcionamiento del programa consiste en dos partes:

#### 1.- Entrenamiento de la red neuronal

```

void redmotor (void)
{
    string fileprueba;
    bool pat = true;
    float Pt[7][6];
    float W1[4], W2[4];
    float t[7][2];
    float b1, b2, n1, n2, a1, a2, e1, e2, eta = 0.5f;
    float errorP, errorG, errorEpoca;

    // semilla para generar los numero aleatoreos
    srand((unsigned)time(NULL));

    cout <<"Red neuronal para motores \n\n";
    cout <<"Ingrese el nombre del archivo:";
    cin >> fileprueba;
    cout << endl;

    // archivo .txt en el cual estan los patrones con los que se entrenara la red
    neuronal
    ifstream in(fileprueba);

    if (!in)
    {

```





```

corrida // imprimo en pantalla los patrones que se entranan en cada

cout <<"Patrones a entrenar:";
for(int c = 0; c < 4; c++)
{
    cout <<" "<<Pt[h][c]<<" ";
}

cout << endl;

// empieza el algoritmo de entrenamiento de la red neuronal tipo
perceptron

//+++++
++++

// n1 y n2 representan las neuronas del modelo matematico del
perceptron
n1 = Pt[h][0]*W1[0] + Pt[h][1]*W1[1] + Pt[h][2]*W1[2] +
Pt[h][3]*W1[3] + b1;
n2 = Pt[h][0]*W2[0] + Pt[h][1]*W2[1] + Pt[h][2]*W2[2] +
Pt[h][3]*W2[3] + b2;

// a1 y a2 son las salidas donde se guardara el valor de n1 y n2
despues // de aplicarse la función de tranferencia tipo hardlims
a1 = hardlims(n1);
a2 = hardlims(n2);

// evaluación del error, de las salidas obtenidas con las
deseadas

e1 = t[h][0] - a1;
e2 = t[h][1] - a2;

errorP = (e1 + e2) / 2.0f;

cout <<"Salidas deseadas: "<<t[h][0]<<" | "<<t[h][1]<<" \n";
cout <<"Salidas obtenida: "<<a1<<" | "<<a2<<" \n";

cout <<"el error de N1 es: "<<e1<<" | N2 es: "<<e2<<" \n";

// reajuste de pesos si el error de cada neurona fue diferente
de 0

if(e1 != 0)
{
    cout <<"neurona 1";
    W1[0] = W1[0] + (e1*Pt[h][0]);
    W1[1] = W1[1] + (e1*Pt[h][1]);
    W1[2] = W1[2] + (e1*Pt[h][2]);
    W1[3] = W1[3] + (e1*Pt[h][3]);
    b1 = b1 + e1;
}

if(e2 != 0)
{
    cout <<" neurona 2";
    W2[0] = W2[0] + (e2*Pt[h][0]);
    W2[1] = W2[1] + (e2*Pt[h][1]);
    W2[2] = W2[2] + (e2*Pt[h][2]);
    W2[3] = W2[3] + (e2*Pt[h][3]);
    b2 = b2 + e2;
}

```

```

    }

    errorEpoca = errorEpoca + (errorP * errorP);
    cout << endl << endl;

//+++++
++++
    }
    errorG = errorEpoca / 7;
}

//una vez terminado el entrenamiento, se crea el archivo pesos.txt en donde
se guardaran los pesos obtenidos.
ofstream pesos("pesos.txt", ios::out);

cout << "Pesos de la primera neurona para el archivo pesos:\n";
for (int a = 0; a < 4; a++)
{
    pesos << "\t W1[" << a << "] = " << W1[a] << " \n";
}

pesos << "\t la ganancia b1 es: " << b1 << " \n";
cout << endl << endl;

cout << "Pesos de la segunda neurona para el archivo pesos:\n";
for (int b = 0; b < 4; b++)
{
    pesos << "\t W2[" << b << "] = " << W2[b] << " \n";
}
pesos << "\t la ganancia b2 es: " << b2 << " \n";
cout << endl << endl;
pesos.close();

// esta función recibe los pesos obtenidos y aqui se realizara la validación
de la red
validacion_red(W1,W2,b1,b2);
}

```

## 2.- Validación de la red neuronal

```

void validacion_red(float WC1[4], float WC2[4], float bc1, float bc2)
{
    bool pat = true;
    char opcion[10];
    float varp, varm;
    float prueba[4];
    float motores[2];
    float n1, n2, a1, a2;

    while(pat)
    {
        cout << "Red neuronal para clasificacion \n";

        cout << "1.- Clasificar\n";
        cout << "2.- Salir\n";
        cout << "Opcion:";

        cin >> opcion;
    }
}

```

```

if((strcmp(opcion, "1")) == 0)
{
    // el usuario ingresara combinaciones que no se entrenaron en la
red
    // y esta los procesara
    cout <<"Ingrese cualquier combinacion.";

    for (int i = 0; i < 4; i++)
    {
        varp = 0.0;
        cout <<"\n:";
        cin >> varp;
        prueba[i] = varp;
    }

    cout <<"Ingrese el valor esperado para los motores\n";

    for (int j = 0; j < 2; j++)
    {
        varm = 0.0;
        cout <<"\n:";
        cin >> varm;
        motores[j] = varm;
    }

    // funcionamiento de la red neuronal con los pesos finales
    n1 = prueba[0]*WC1[0] + prueba[1]*WC1[1] + prueba[2]*WC1[2] +
prueba[3]*WC1[3] + bc1;
    n2 = prueba[0]*WC2[0] + prueba[1]*WC2[1] + prueba[2]*WC2[2] +
prueba[3]*WC2[3] + bc2;

    a1 = hardlims(n1);
    a2 = hardlims(n2);

    cout <<"Salidas deseadas: "<<motores[0]<<" | "<<motores[1]<<"
\n";
    cout <<"Salidas obtenida: "<<a1<<" | "<<a2<<" \n";

}

else if ((strcmp(opcion, "2")) == 0)
{
    pat = false;
}

else
{
    cout <<"Ingrese una opcion valida\n";
}

}

```

Se inicia ingresando el nombre del archivo .txt el cual contiene los patrones con los que se entrenara la red neuronal.

```
C:\Users\TLZG KH\Documents\visual studio 2010\Projects\RN_PT\Debug\RN_PT.exe
Programa Red Neuronal Proyceto terminal
Red neuronal para motores

Ingrese el nombre del archivo:patrones.txt

Pesos aleatorios de la primera neurona :
    W1[0] = 0.8
    W1[1] = 0.5
    W1[2] = 0.2
    W1[3] = 0.2

Pesos aleatorios de la segunda neurona
    W2[0] = 0.7
    W2[1] = 0.3
    W2[2] = 0.5
    W2[3] = 0.5

Epoca 0 -----
Patrones a entrenar: -1, -1, -1, -1,
Salidas deseadas: 1 | 1
Salidas obtenida: -1 | -1
el error de N1 es: 2 | N2 es: 2
neurona 1 neurona 2
```

Una vez ingresado se mostraran los pesos aleatorios con los que comenzara a entrenar la red neuronal seguidamente de los resultados de las épocas de entrenamiento, el cual al final de cada patrón se mandara a imprimir en que neurona hubo reajuste de pesos

```
C:\Users\TLZG KH\Documents\visual studio 2010\Projects\RN_PT\Debug\RN_PT.exe
Epoca 0 -----
Patrones a entrenar: -1, -1, -1, -1,
Salidas deseadas: 1 | 1
Salidas obtenida: -1 | -1
el error de N1 es: 2 | N2 es: 2
neurona 1 neurona 2

Patrones a entrenar: -1, 1, -1, -1,
Salidas deseadas: 1 | -1
Salidas obtenida: 1 | 1
el error de N1 es: 0 | N2 es: -2
neurona 2

Patrones a entrenar: 1, -1, -1, -1,
Salidas deseadas: -1 | 1
Salidas obtenida: 1 | 1
el error de N1 es: -2 | N2 es: 0
neurona 1

Patrones a entrenar: 1, -1, 1, -1,
Salidas deseadas: 1 | -1
Salidas obtenida: -1 | 1
el error de N1 es: 2 | N2 es: -2
neurona 1 neurona 2
```

El número de épocas necesarias para entrenar a la red fueron de 4, a partir de la 5 época ya no hay reajuste de pesos

```
C:\Users\TLZG KH\Documents\visual studio 2010\Projects\RN_PT\Debug\RN_PT.exe
Patrones a entrenar: -1, -1, -1, 1,
Salidas deseadas: -1 ; 1
Salidas obtenida: 1 ; 1
el error de N1 es: -2 ; N2 es: 0
neurona 1

Patrones a entrenar: 1, 1, -1, -1,
Salidas deseadas: 1 ; 1
Salidas obtenida: 1 ; -1
el error de N1 es: 0 ; N2 es: 2
neurona 2

Patrones a entrenar: -1, -1, 1, -1,
Salidas deseadas: 1 ; -1
Salidas obtenida: 1 ; -1
el error de N1 es: 0 ; N2 es: 0

Epoca 1 -----
Patrones a entrenar: -1, -1, -1, -1,
Salidas deseadas: 1 ; 1
Salidas obtenida: -1 ; 1
el error de N1 es: 2 ; N2 es: 0
neurona 1
```

C:\Users\TLZG KH\Documents\visual studio 2010\Projects\RN\_PT\Debug\RN\_PT.exe

```
Epoca 4 -----  
Patrones a entrenar: -1, -1, -1, -1,  
Salidas deseadas: 1 | 1  
Salidas obtenida: 1 | 1  
el error de N1 es: 0 | N2 es: 0
```

```
Patrones a entrenar: -1, 1, -1, -1,  
Salidas deseadas: 1 | -1  
Salidas obtenida: 1 | -1  
el error de N1 es: 0 | N2 es: 0
```

```
Patrones a entrenar: 1, -1, -1, -1,  
Salidas deseadas: -1 | 1  
Salidas obtenida: -1 | 1  
el error de N1 es: 0 | N2 es: 0
```

```
Patrones a entrenar: 1, -1, 1, -1,  
Salidas deseadas: 1 | -1  
Salidas obtenida: 1 | -1  
el error de N1 es: 0 | N2 es: 0
```

C:\Users\TLZG KH\Documents\visual studio 2010\Projects\RN\_PT\Debug\RN\_PT.exe

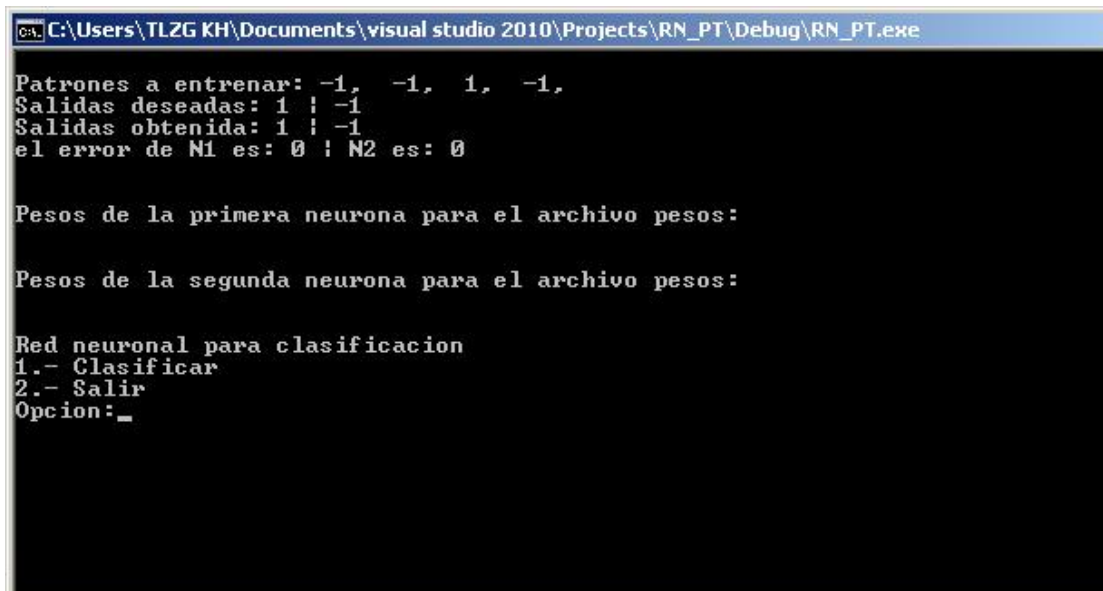
```
Patrones a entrenar: -1, -1, -1, 1,  
Salidas deseadas: -1 | 1  
Salidas obtenida: -1 | 1  
el error de N1 es: 0 | N2 es: 0
```

```
Patrones a entrenar: 1, 1, -1, -1,  
Salidas deseadas: 1 | 1  
Salidas obtenida: 1 | 1  
el error de N1 es: 0 | N2 es: 0
```

```
Patrones a entrenar: -1, -1, 1, -1,  
Salidas deseadas: 1 | -1  
Salidas obtenida: 1 | -1  
el error de N1 es: 0 | N2 es: 0
```

```
Epoca 5 -----  
Patrones a entrenar: -1, -1, -1, -1,  
Salidas deseadas: 1 | 1  
Salidas obtenida: 1 | 1  
el error de N1 es: 0 | N2 es: 0
```

Una vez terminado el entrenamiento los pesos se guardan en el archivo “pesos.txt” y se da la opción al usuario de validar otros patrones con los cuales la red no fue entrenada o la opción de salir del programa.



```
C:\Users\TLZG KH\Documents\visual studio 2010\Projects\RN_PT\Debug\RN_PT.exe
Patrones a entrenar: -1, -1, 1, -1,
Salidas deseadas: 1 | -1
Salidas obtenida: 1 | -1
el error de N1 es: 0 | N2 es: 0

Pesos de la primera neurona para el archivo pesos:

Pesos de la segunda neurona para el archivo pesos:

Red neuronal para clasificacion
1.- Clasificar
2.- Salir
Opcion: _
```

Una vez guardados los pesos de la red neuronal estos pueden usarse en el programa que va a ir grabado en el micro controlador del robot, el compilador mikroC PRO [5] for pic traduce de código c a código ensamblador, lo cual también genera el archivo .hex el cual es el archivo que se graba en el micro controlador a través del programa PICKit 2 versión 2.61 [6] con un grabador PIC nano v1.

Se tiene al principio del programa las directivas #define lo cual asigna el remplazo de tokens de los puertos a los que se tienen conectados los sensores y motores para facilitar la programación.

```
#define echo_s1 PORTB.B4
#define echo_s2 PORTB.B7
#define echo_s3 PORTB.B6
#define echo_s4 PORTB.B5
#define motor2_0 PORTC.B0
#define motor2_1 PORTC.B3
#define motor1_0 PORTC.B4
#define motor1_1 PORTC.B5
```

El programa inicia con la configuración de los puertos B y C del PIC16f876A el cual con el registro TRIS los configura como salida de datos digitales .

```
//configuración inicial de los puertos del PIC16F876A
CMCON = 7; //des habilitación de todas las comparaciones
TRISB = 0; //Puertos B del PIC configurados como salida de datos
TRISC = 0; //Puertos C del PIC configurados como salida de datos
```

```
PORTB = 0;    // 0 lógico en todos los puertos B
PORTC = 0;    // 0 lógico en todos los puertos C
```

Posteriormente se pone un retraso de 3000 milisegundos (3 segundos) para acomodar el robot en la pista una vez conectado a la batería, y se pone un cero lógico a los motores, lo cual producirá que no se mueva el robot

```
Delay_ms (3000);
motor1_0 = 0;
motor1_1 = 0;
motor2_0 = 0;
motor2_1 = 0;
```

El robot empezara con la activación de los sensores de uno en uno comenzando con el izquierdo, derecho, adelante y atrás. Una vez obtenida la distancia medida por el sensor se comparara con la distancia mínima que se definió lo cual definirá el valor de la señal para la red neuronal.

```
while(1)
{
    get_dist1 ();
    if (dist1 <= 7.0) {
        sig1 = 1.0;
    }
    else {
        sig1 = -1.0;
    }

    get_dist2 ();
    if (dist2 <= 7.0) {
        sig2 = 1.0;
    }
    else {
        sig2 = -1.0;
    }

    get_dist3 ();
    if (dist3 <= 7.0) {
        sig3 = 1.0;
    }
    else {
        sig3 = -1.0;
    }

    get_dist4 ();
    if (dist4 <= 7.0) {
        sig4 = 1.0;
    }
    else{
        sig4 = -1.0;
    }
}
```



```

    }
    redmotor (sig1, sig2, sig3, sig4);
    Delay_ms(200);
}

```

Una vez llamada la función de la red neuronal, se pone un retraso de 200 milisegundos y se vuelve a repetir el funcionamiento de los sensores y la red. Todo esto en un ciclo infinito.

La función de los sensores envía un pulso de un 1 lógico de una duración de 10 microsegundos, configura el puerto correspondiente como entrada de datos digitales, se incrementa un contador el tiempo que dure el pulso que regresa el sensor y finalmente se configura el mismo puerto como salida de datos y se procesa el contador dividiendo entre 58 para obtener la distancia. Esto con las siguientes variables globales.

```

float dist1, dist2, dist3, dist4;
unsigned int TF, TF2, TF3, TF4;

```

Función que tiene cada sensor, solamente cambia el puerto asignado.

```

void get_dist1()
{
    echo_s1 = 0;                /* Pulso de 10 us que se*/
    Delay_us(5);                /* le envía al sensor */
    echo_s1 = 1;                /* para realizar el */
    Delay_us(10);               /* seseó repetidas */
    echo_s1 = 0;                /* veces */

    TRISB4_bit = 1;            //el puerto pasa a ser entrada de datos
    TF = 0;
    dist1 = 0.0;
    while (!echo_s1);

    while(echo_s1){

        TF += 7;
    }

    TRISB4_bit = 0;            //el puerto vuelve a ser salida de datos
    dist1 = TF;
    dist1 = (dist1/58);
}

```

En la función de la red se procesa cada señal con los pesos adquiridos anteriormente, se procesa cada neurona con la función de transferencia tipo Hardlims y se compara la salida para enviar las señales necesarias para poner en reversa al motor si la salida fue -1 o para las señales para avanzar hacia adelante si la salida fue 1.

```

void redmotor(float s1, float s2, float s3, float s4)
{
    float W1[4], W2[4], b1, b2;

```

```

float n1, n2;
//pesos finales que se obtuvieron en el entrenamiento de la red
W1[0] = 2.7;
W1[1] = -5.6;
W1[2] = 2.9;
W1[3] = -5.8;
b1 = 2.1;
W2[0] = -3.9;
W2[1] = 4.9;
W2[2] = -3.8;
W2[3] = 0.2;
b2 = 0.8;

n1 = s1*W1[0] + s2*W1[1] + s3*W1[2] + s4*W1[3] + b1; //neurona del motor
izquierdo
n2 = s1*W2[0] + s2*W2[1] + s3*W2[2] + s4*W2[3] + b2; //neurona del motor
derecho

//motor 1 izq;
if (hardlims(n1) == -1)
{
    //reversa
    motor1_0 = 1;
    motor1_1 = 0;
}
else
{
    motor1_0 = 0;
    motor1_1 = 1;
}

//motor 2 der
if (hardlims(n2) == -1)
{
    //reversa
    motor2_0 = 1;
    motor2_1 = 0;
}
else
{
    motor2_0 = 0;
    motor2_1 = 1;
}
}

```

Función de transferencia.

```

int hardlims (float salida)
{
    if(salida >= 0.0) {
        return 1;
    }
}

```

```



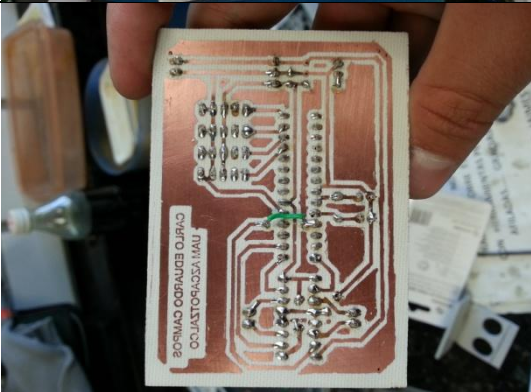
else {
    return -1;
}
}

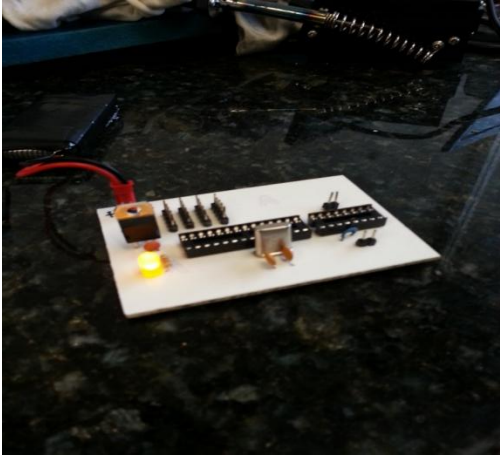

```

Finalmente se procedió a la construcción del prototipo en el cual se implemento la red neuronal


**Proceso de Construcción del Prototipo**

- ❖ Diseño electrónico (PuenteH, Drivers etc) en tarjeta impresa

Elemento	Descripción
	<p>La construcción del prototipo se realizó en el laboratorio de Sistemas Neurodifusos de la UAM-A, ubicado en el edificio G-313/Cub.05</p> <p>Y en el Laboratorio de Control de Procesos ubicado en el edificio W.</p>
	<p>Hoja con los circuitos impresos Para grabar en la placa fenólica</p>
	<p>Placa de cobre con el circuito ya grabado</p>

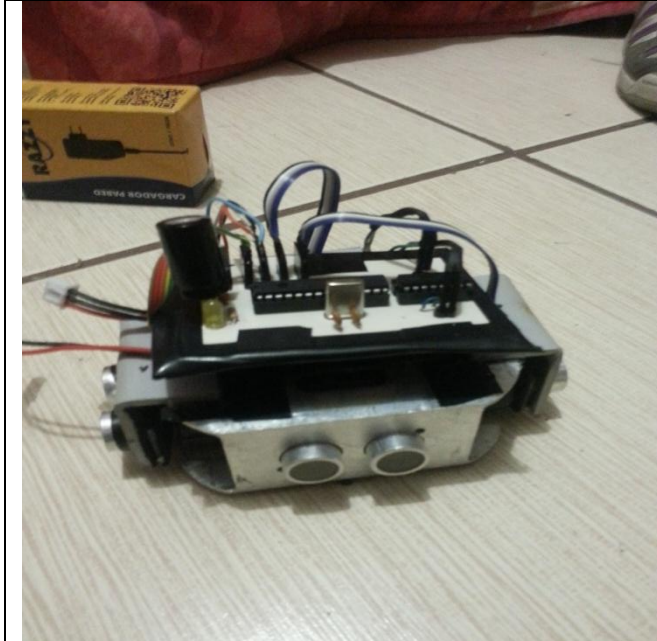
	<p>Placa de cobre con los zocalos para el PIC 16F876A, el driver L293d [], regulador, cristal y capacitores ya soldados. Conectada a una batería de 8 V con un led que indica que si hay paso de corriente</p>
	<p>Sensores ultrasónicos SRF05[3] con sus bases para montarse en el robot, el sensor restante ya está adentro de la base de aluminio del robot</p>

❖ Construcción del robot en aluminio resistente

Elemento	Descripción
	<p>Soldando los componentes que iran en la placa de cobre</p>



Base principal de aluminio con un sensor y los dos moto reductores integrados



Ensamblaje de las placa y los sensores en la base de aluminio, aquí se presenta el robot en su etapa final de construcción

Una vez terminado el robot se probó en un ambiente improvisado que simula la lunaarena de la competencia y el robot paso de la zona inicial a la zona de excavación evadiendo los obstáculos. Los videos se incluirán en los CD.

## Conclusiones

Dependiendo del problema a solucionar es como se debe diseñar la red neuronal que se va a implementar. Una vez teniendo los patrones adecuados para el ambiente de ese problema la red neuronal responde de manera efectiva a las situaciones que no se le presentaron en el entrenamiento.

El funcionamiento de la red neuronal es independiente del estado de la robótica en el que se implemente, obviamente con algoritmos y sistemas más complejos se pueden pulir los detalles que afectan al funcionamiento general del robot. En este caso el comportamiento del robot fue el óptimo para las pruebas de la red neuronal.



## Bibliografía

- [1] Introducción a las redes neuronales artificiales [En línea]. Disponible: <http://hugo-inc.net/16.net/RNA/Unidad%201/1.6.html>
- [2] Kennedy Space Center Visitor Complex (Agosto 22, 2012) “NASA’s Fourth Annual Lunabotics Mining Competition Rules & Rubrics 2013”. [En línea]. Disponible: [http://www.nasa.gov/pdf/390619main\\_lunaboticsrules.pdf](http://www.nasa.gov/pdf/390619main_lunaboticsrules.pdf)
- [3] SRF05 Ultra Sonic Ranger Technical Documentation [En línea]. Disponible: <http://www.robot-electronics.co.uk/htm/srf05tech.htm>
- [4] PIC16F87XA Data Sheet [En línea]. Disponible: <http://ww1.microchip.com/downloads/en/devicedoc/39582b.pdf>
- [5] mikroC PRO for PIC [En línea]. Disponible: <http://www.mikroe.com/mikroc/pic/>
- [6] PICkit 2 versión 2.61 [En línea]. Disponible: [http://www.microchip.com/stellent/idcplg?IdcService=SS\\_GET\\_PAGE&nodeId=1406&dDocName=en023805](http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en023805)