Implementación de un algoritmo de red neuronal constructivo en un microcontrolador Arduino UNO.

Francisco Ortega-Zamorano

Universidad de Málaga, Departamento de L.C.C., ETSI Informática, España.
fortega@lcc.uma.es

Resumen Un algoritmo constructivo de redes neuronales receintemente introducido, C-Mantec, se implementa en un dispositivo microcontrolador. El algoritmo C-Mantec genera arquitecturas de redes neuronales muy compactas con muy buena capacidad de predicción; la combinación de ambas características hacen de este algoritmo un buen candidato para realizar el aprendizaje de un conjunto de datos sin necesidad de transmitir información a una unidad de control central. Se analizan los detalles más complejos de la implementación de dicha aplicación y se realiza una prueba con un conjunto de funciones de referencia utilizadas normalmente en el diseño de circuitos digitales para mostrar el correcto funcionamiento de la aplicación. Además se presentan tres casos de estudio con sus correspondientes evaluaciones en cuanto a su funcionamiento.

Abstract A recently proposed constructive neural network algorithm, named C-Mantec, is fully implemented in a Arduino board. The C-Mantec algorithm generates very compact size neural architectures with good prediction abilities, and thus the board can be potentially used to learn on-site sensed data without needing to transmit information to a central control unit. An analysis of the more difficult steps of the implementation is detailed, and a test is carried out on a set of benchmark functions normally used in circuit design to show the correct functioning of the implementation. Also three case studies are presented and their different evaluations.

Keywords: Constructive Neural Networks, Microcontroller, Arduino

Tutores: Leonardo Franco y José Manuel Jerez Aragonés.

1. Introducción

Diferentes tecnologías como son las redes de sensores inalámbricas [1], los sistemas empotrados [2] y los sistemas de tiempo real [3] son empleados hoy en día en todo tipo de aplicaciones industriales, en muchas de las cuales pueden usarse dispositivos microcontroladores [4]. Los recientes avances en la potencia de cálculo de este tipo de sistemas está empezando a permitir el uso de sistemas de aprendizaje, los cuales permiten ajustar el funcionamiento a medida que se reciben los datos de entrada.
Las redes neuronales son una especie de sistemas de aprendizaje, flexibles y ampliamente utilizadas que pueden ser empleadas para esta tarea. Sin embargo una desventaja de las mismas es que su fase de entrenamiento necesita una potencia de cómputo intensiva, haciendo su uso prohibitivo incluso para los sistemas modernos. En este sentido, el algoritmo de red neuronal constructiva C-Mantec recientemente propuesto tiene la ventaja de ser muy rápido en su fase entrenamiento y construcción, en comparación con los métodos estándares utilizados en redes neuronales de tipo "feed-forward´´. Además el algoritmo C-Mantec genera arquitecturas muy compactas, hecho que es útil dado que los recursos de memoria son muy limitados en los microcontroladores.
En este trabajo se ha implementado el algoritmo C-Mantec [5] en un microcontrolador, incluyendo el proceso de aprendizaje, ya que éste proceso se realiza en el propio dispositivo en lugar de realizarlo de forma externa como se hace habitualmente. El dispositivo microcontrolador elegido ha sido la placa Arduino UNO [6] , ya que es un dispositivo muy popular, económico y eficiente, siendo además de código abierto.
Los factores críticos en el momento de la ejecución del algoritmo C-Mantec son la escasez de recursos de la memoria del microcontrolador utilizado (32 KB Flash, RAM de 2 KB y 1 KB de memoria EPROM)

y la velocidad de cómputo, por lo que la aplicación se ha realizado con aritmética de punto fijo para todas aquellas variables en las que se ha podido.
El presente documento está estructurado de la siguiente manera: en primer lugar, describimos brevemente el estado del arte de las redes neuronales y los microcontroladores, para pasar a una descripción del algoritmo C-Mantec y la placa Arduino, tras esto se detalla la implementación del algoritmo, dando a continuación unos resultados intermedios de esta implementación. Posteriormente se presentan tres casos de estudios, evaluando la eficiencia de una posible implementación del sistema en dichos casos. Finalizamos con las conclusiones.

2. Antecedentes

Las Redes Neuronales Artificiales (Artificial Neural Network (ANN)) [7] son modelos matemáticos inspirados en el funcionamiento del cerebro, que puede ser utilizado en los problemas de agrupación y clasificación, y que se han aplicado con éxito en varios campos, incluyendo el reconocimiento de patrones, predicción del mercado de valores, tareas de control, diagnóstico y pronóstico médico, etc.
A pesar de años de investigación en el campo de la ANN, la selección de una arquitectura adecuada para un problema dado sigue siendo una tarea compleja [8,9,10]. Entre las diversas estrategias para resolver este problema, las Redes Neuronales Constructivas (Constructive Neural Networks (CoNN)) ofrecen la posibilidad de generar redes que crecen a medida que se analizan los datos de entrada, con lo que se tiene una arquitectura conforme al conjunto de datos [11]. Por otra parte, el procedimiento de formación de las CoNN, considerado un problema computacionalmente costoso en las redes neuronales "feed-forward" estándar, se puede hacer "on-line" y relativamente rápido. C-Mantec es un algoritmo de tipo CoNN recientemente propuesto [5] que implementa competencia entre neuronas, permitiendo que todas las neuronas aprendan durante todo el proceso de construcción de la arquitectura, ya que no congela los pesos sinápticos ya existentes, como lo hacen la mayoría de los algoritmos CoNN. C-Mantec además incorpora un proceso de filtrado de patrones para evitar problemas de sobreajuste. Estas dos características permiten al algoritmo generar arquitecturas neuronales compactas con muy buena capacidad de generalización, por lo que resulta un algoritmo adecuado para su aplicación en dispositivos con recursos limitados como microcontroladores, sistemas integrados, redes de sensores y FPGAs.
Un microcontrolador (abreviado μC, UC o MCU) es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica. Un microcontrolador incluye en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida. Cada vez existen más aplicaciones que incorporan un microcontrolador con el fin de aumentar sustancialmente sus prestaciones, reducir su tamaño y coste, mejorar su fiabilidad y disminuir el consumo. [12,13,14,15]. En la tabla 1 podemos observar los microcontroladores más usados ordenados por familias y numero de bits de su arquitectura. En particular la placa Arduino que se utilizará en este trabajo es un dispositivo de hardware libre que utiliza microcontroladores Atmel AVR , si bien pueden utilizarse otros como el Atmega168, Atmega328, Atmega1280 o ATmega8. Estas palcas ofrecen al usuario varios puertos de entrada/salida y un entorno de desarrollo fácil y amigable que permite desarrollar proyectos de la más diversa naturaleza [16,17,18]. Al ser un placa de hardware libre muchas de sus aplicaciones no son publicadas en artículos científicos sino en internet [19].
Las limitaciones más importantes de este tipo de dispositivos son la memoria y la velocidad de cómputo, por ello implementar redes neuronales ha sido una tarea compleja ya que estos algoritmos requieren un gran cantidad de memoria para poder realizar el aprendizaje de un conjunto de patrones. Esta es la razón por la cual un gran número de las aplicaciones que emplean redes neuronales en un microcontrolador utilizan el aprendizaje "off-line" es decir, se aprende la red neuronal en otro dispositivo (normalmente un PC) y después el microcontrolador emplea la red aprendida [20,21,22]. En otros casos el microcontrolador se emplea a modo de red distribuida, es decir, se genera una red de microcontroladores donde cada nodo (microcontrolador) representa una neurona y la comunicación entre nodos representa los pesos sinápticos [23,24]. La opción más compleja es implementar el proceso de aprendizaje "on-chip", es decir en el propio microcontrolador. Existen algunas aplicaciones realizadas de esta manera, pero en las cuales la arquitectura de red neuronal se ha elegido ad hoc para solucionar un problema específico[25]. Sin embargo en la implementación propuesta en este trabajo no hace falta

Cuadro 1. Microcontroladores más utilizados en aplicaciones actuales

Empresa	8 bits	16 bits	32 bits
Atmel	AVR,89Sxxxx		SAM7(ARM7TDMI),AVR32, SAM3(ARM Cortex-M3
Freescale (Motorola)	68HC05, 68HC08, 68HC11, HCS08	68HC12, 68HCS12, 68HCSX12, 68HC16	683xx, PowerPC, ColdFire
Holtek	HT8		
Intel	MCS-48,MCS51,8xC251	MCS96, MXS296	
National Semic.	COP8		
Microchip	Familias 10f2xx 12Cxx 12Fxx, 16Cxx y 16Fxx	PIC24F, PIC24H y dsPIC30FXX	PIC32
NXP Semiconduc. (Philips)	80C51	XA	Cortex-M3, Cortex-M0, ARM7, ARM9
Renesas (Mitsubishi)	78K, H8	H8S, 78K0R, R8C, R32C/M32C/M16C	RX, V850, SuperH, SH-Mobile, H8SX
STMicroelectro.	ST 62, ST 7		STM32 (ARM7)
Texas Instruments	TMS370	MSP430	C2000, Cortex-M3 (ARM), TMS570 (ARM)
Zilog	Z8, Z86E02		

elegir la arquitectura de la red a priori, la cual se realiza de forma automática, por lo que serviría para múltiples aplicaciones de diversa índole.

3. C-Mantec, algoritmo constructivo de red neuronal

C-Mantec (Competitive Majority Network Trained by Error Correction) es un nuevo algoritmo constructivo de redes neuronales que utiliza la competencia entre las neuronas y una regla modificada de aprendizaje del perceptrón (thermal perceptron) para construir arquitecturas compactas con una buena capacidad de predicción. La novedad de C-Mantec es que las neuronas compiten por el aprendizaje de los datos entrantes, y este proceso permite la creación de arquitecturas neuronales muy compactas. Los estado de activación (S) de las neuronas en la capa oculta depende de las N señales de entrada,ψ_i, y de los valores de los N pesos sinápticos (ω_i) de la siguiente forma:

$$S = \begin{cases} 1(ON) & \text{if } h \geq 0 \\ 0(OFF) & \text{otherwise} \end{cases} \tag{1}$$

Donde h es el potencial sináptico de la neurona y queda definido como:

$$h = \sum_{i=0}^{N} \omega_i \cdot \psi_i \tag{2}$$

En la regla del perceptrón termal, la modificación de los pesos sinápticos, $\Delta\omega_i$, se realiza en línea (después de la presentación de un único patrón de entrada) de acuerdo con la siguiente ecuación:

$$\Delta\omega_i = (t - S)\ \psi_i\ T_{fac} \tag{3}$$

Donde t es el valor objetivo para un patrón de entrada determinada y ψ representa el valor de la entrada i conectado a la salida mediante el peso ω_i. La diferencia entre las regla de aprendizaje del perceptrón estándar y la del perceptrón termal es que ésto último incorpora el factor T_{fac}. El cálculo del valor de éste factor, se muestra en ecuación 4, dicho valor depende del potencial sináptico(h) y de una temperatura introducida artificialmente (T) que disminuye a medida que el proceso de aprendizaje avanza.

$$T_{fac} = \frac{T}{T_0} e^{-\frac{|\phi|}{T}} \tag{4}$$

Dicha disminución se realiza conforme a la ecu. 5.

$$T = T_0 \cdot (1 - \frac{I}{I_{max}}), \tag{5}$$

donde I es el contador de ciclos por neurona que cuantifica las iteraciones ("ciclo de aprendizaje") del algoritmo en esa neurona y I_{max} es el número máximo de estas iteraciones. Un ciclo de aprendizaje del algoritmo es el proceso que se inicia cuando un patrón determinado se presenta a la red y acaba después de la modificación de los pesos sinápticos por parte de la neurona seleccionada para reconocer el patrón (ya sea una existente o una nueva neurona).
C-Mantec, como algoritmo CNN, tiene además la ventaja de generar la topología de la red on-line mediante la adición de nuevas neuronas durante la fase de entrenamiento, lo que lleva a tiempos de entrenamiento más rápido y arquitecturas más compactas. El algoritmo C-Mantec tiene 3 parámetros que se establecen en el momento de iniciar el procedimiento de aprendizaje. Varios experimentos han demostrado que el algoritmo es muy robusto frente a los cambios de los valores de los parámetros y por lo tanto C-Mantec funciona bastante bien en una amplia gama de valores. Los tres parámetros son:

- I_{max}: número máximo de iteraciones permitidas para cada neurona en la capa oculta por ciclo de aprendizaje.
- g_{fac}: factor de crecimiento que determina cuándo detener un ciclo de aprendizaje e incluir una nueva neuronas en la capa oculta.
- ϕ: determina si un patrón de entrada se considera ruidoso y se retira del conjunto de datos de entrenamiento de acuerdo con la siguiente condición:

$$\forall X \in \{X_1, X_2, ..., X_N\}, delete(X) \mid NTL \geq (\mu + \phi \cdot \sigma), \tag{6}$$

donde N representa el número de patrones del conjunto de datos, NTL es el número de veces que el patrón X ha sido aprendido por la red en el ciclo de aprendizaje actual, y el par $\{\mu, \sigma\}$ corresponde a la media y la varianza de la distribución normal que representa el número de veces que cada patrón del conjunto de datos se ha aprendido durante el ciclo de aprendizaje. Este procedimiento de aprendizaje se basa esencialmente en la idea de que los patrones son aprendidos por las neuronas de la capa oculta de la arquitectura, cuya salida difiere del valor objetivo (clasificado erróneamente la entrada) y para el cual su temperatura interna es mayor que el valor de ajuste g_{fac}. En el caso en el que más de un perceptrón termal de la capa oculta satisfaga dicha condición en una iteración dada, el perceptrón con la temperatura más alta es el candidato seleccionado para reconocer el patrón entrante. Una nueva neurona solo se añade a la red cuando no hay perceptron termales que cumpla con estas condiciones, con lo que se inicia un nuevo ciclo de aprendizaje.
En la figura 1 puede observarse el diagrama de flujo del algoritmo descrito, mostrándose las funciones más relevantes en los recuadros de la imagen, la toma de decisiones como los rombos y los estados más importantes, inicio y fin como los óvalos.

3.1. Extensión a problemas multiclase

El algoritmo C-Mantec es un algoritmo de clasificación binario, por lo que para poder utilizarlo en la clasificación de conjuntos de datos con salida multiclase se aplicaran tres esquemas muy conocidos [26,27]:Uno-contra-todos, Uno-contra-Uno y P-contra-Q. Los tres métodos obtienen un clasificador de K clases usando estrategias que combinan M clasificadores binarios y un modulo de decisión simple que tiene como entrada la salida de los clasificadores. Los M clasificadores son entrenados independientemente con diferentes conjuntos de entrada Ce_i que variarán en función de la estrategia elegida y del conjunto de entrada original Ce.

$$Ce = \{(\mathbf{X}, C_i) / \mathbf{X} = (x_1, ..., x_N),\ C_i \in \{C_1, ..., C_k\}\} \tag{7}$$

donde $(\mathbf{X}, C_i)$ es una tupla del conjunto de entrenamiento compuesta por un vector de entrada $\mathbf{X}$ de tamaño N y una clase de salida C_i perteneciente al conjunto de clases posibles $\{C_1, ..., C_k\}$.
El clasificador que se ha seleccionado para poder hacer un clasificador multiclase ha sido P-contra-Q. Este esquema puede ser visto como el punto medio entre los otros dos esquemas, ya que en este caso cada red $CMantec_i$, separará un conjunto clases P_i del resto $Q_i = K - P_i$, siendo K el número de clases del problema. Esta red separa dos grupos de clases entre ellas, pero no separa las clases que pertenecen

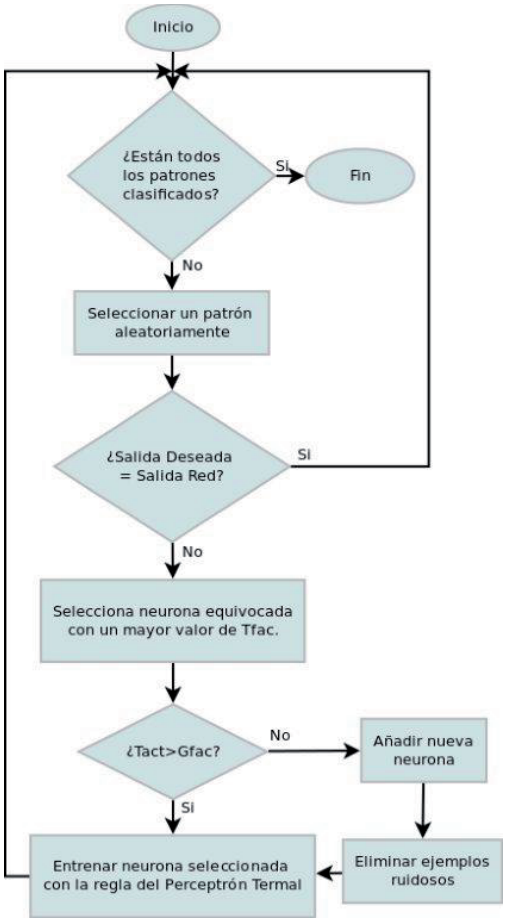

Figura 1. Diagrama de flujo del algoritmo constructivo de redes neuronales C-Mantec.

al mismo grupo, por lo que serán necesarias más redes C-Mantec que nos permita discriminarlas. Por lo tanto, en este esquema serán necesarias tantas redes como hagan falta hasta conseguir discriminar todas las clases. El esquema OAA podría ser visto como un tipo de esquema PAQ donde cada P_i sólo contiene una clase y Q_i el resto de clases. El número de redes C-Mantec mínimo para poder clasificar K patrones es $M = \log(K)$ siguiendo una codificación binaria. Sin embargo, esta codificación mínima, aunque es muy eficiente en términos de redes a generar, no lo es tanto en términos de generalización ya que no genera ninguna redundancia, por lo que es beneficioso utilizar un número mayor de redes $CMantec_i$.

$$Ce_i = \{(\mathbf{X}, S_i(\mathbf{X}, C_e, Cod)) / \mathbf{X} = (x_1, ..., x_N)\} \tag{8}$$

$$S_i(X, Ce) = \begin{cases} 1 & Si\ C_i\ \in\ Pi \\ -1 & Si\ C_i\ \in\ Qi \end{cases} \tag{9}$$

Al igual que en el caso OAA, el modulo de decisión tendrá que decidir cuál será la clase de salida a partir del resultado generado por cada una de las M redes C-Mantec. Cada una de las K clases espera un vector de salida de tamaño M asociado a esa clase a la que nos referiremos como vector prototipo de la clase k (V_k). Si el vector resultante de la salida de las M redes, $\mathbf{V}$, no coincide con ninguno de los K vectores prototipo, se escogerá como salida aquella clase k que tenga un vector $\mathbf{V}_k$ más cercano a $\mathbf{V}$ en distancia Hamming, y en caso de empate, puede ser resuelto escogiendo la clase con mayor probabilidad a priori o, en caso de que no pudiera ser resuelto, escogiendo una aleatoriamente. Se ha

añadido redundancia a este método usando $K+log\,(K)$ redes, siguiendo las K primeras una codificación similar a la utilizada en el método OAA y una codificación binaria para las $log\,(K)$ siguientes.

Redes C-Mantec						Clase
Red_0	Red_1	Red_2	Red_3	Red_4	Red_5	
P_0	Q_1	Q_2	Q_3	Q_4	Q_5	C_0
Q_0	P_1	Q_2	Q_3	Q_4	P_5	C_1
Q_0	Q_1	P_2	Q_3	P_4	Q_5	C_2
Q_0	Q_1	Q_2	P_3	P_4	P_5	C_3

Cuadro 2. Ejemplo del esquema P-contra-Q (PAQ) en el cual un problema de 4 clases ha sido codificado por 6 redes binarias.

La tabla 2 muestra un ejemplo de cómo es posible codificar un problema multiclase de $K = 4$ usando el esquema PAQ con $M = K + \log(K)$ redes C-Mantec, en las cuales las 4 primeras ($Red_0 ... Red_3$) utilizan una codificación de tipo OAA, mientras que las dos últimas (Red_4 y Red_5) utilizan una codificación de tipo binaria. Con esta codificación el conjunto de entrenamiento Ce_i de una Red_i debería corresponderse con la columna i de la tabla y la fila j de la clase que se desea codificar siguiendo la ecuación 8.

4. Placa Arduino UNO

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares. El Arduino Uno es una placa con microcontrolador basada en el chip ATmega328 [28]. Tiene 14 pines con entradas/salidas digitales (6 de las cuales pueden ser usadas como salidas PWM), 6 entradas analógicas, un cristal oscilador a 16Mhz, conexión USB, entrada de alimentación, una cabecera ISCP, y un botón de reset. El ATmega328 tiene 32KB de memoria flash para almacenar código (2KB son usados para el arranque del sistema (bootloader)), dispone de 2 KB de memoria SRAM y 1KB de EEPROM.
El Arduino Uno dispone de un sistema de comunicación para su uso con un ordenador, otro Arduino, u otros microcontroladores. El ATmega328 ofrece comunicación serie (UART), la cual está disponible en los pines digitales 0 (Rx) y 1 (Tx), además dispone de comunicación I2C y SPI. Para todas estos sistemas de comunicación la plataforma Arduino proporciona las librerías para su uso. Arduino es un descendiente de la plataforma de código abierto *Wiring* por lo que se programa usando este lenguaje, similar a C++ con algunas ligeras modificaciones. Se emplea bajo el entorno de *processing*. En la Tabla 3 se puede observar las especificaciones del microcontrolador Atmega328 correspondiente al Arduino UNO.

Cuadro 3. Especificaciones más relevantes del microcontrolador Atmega328 utilizado en la placa Arduino UNO

	Atmega328
Voltaje operativo	5 V
Voltaje de entrada recomendado	7 - 12 V
Voltaje de entrada límite	6 - 20 V
Pines de entrada y salida digital	14 (6 proporcionan PWM)
Pines de entrada analógica	6
Intensidad de corriente	40 mA
Memoria Flash	32KB (2KB reservados para el bootloader)
SRAM	2 KB
EEPROM	1 KB
Frecuencia de reloj	16 MHz

Las placas Arduino se comercializan finalizadas o pueden ser ensambladas por parte de los usuarios ya que la información del diseño hardware está disponible de manera pública. Las características físicas son, una longitud y un ancho de 6.8 and 5.3 cm respectivamente, más el conector USB y el de alimentación que se extiende más allá de la dimensión anterior. Una imagen del Arduino UNO puede observarse en la figura 2.

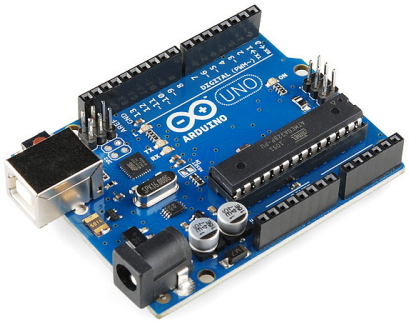

Figura 2. Imagen de la placa Arduino UNO usada para la implementación del algoritmo C-Mantec.

5. Implementacion del algoritmo C-Mantec

El algoritmo C-Mantec implementado en el código wiring es transmitido por USB desde el programa correspondiente en el ordenador hacia la placa. El proceso implementado en el microcontrolador consta de dos fases, la fase de aprendizaje, donde se calcula la red neuronal a partir de los datos guardados en el microcontrolador y la fase de ejecución donde se calcula las diferentes salidas de los patrones de entrada en función de la red neuronal aprendida previamente. Los dos estados son seleccionados en el microcontrolador mediante un pin E/S.
El estado de aprendizaje consta de dos acciones claramente diferenciadas, la carga de patrones y el cálculo de la red neuronal propiamente dicha. Los patrones pueden ser cargados on-line mediante los pines de E/S de la placa o mediante una comunicación serie por el puerto USB, pero en ambos casos deben ser guardados en la memoria EEPROM. A continuación se explican las principales cuestiones técnicas para la implementación del algoritmo de acuerdo con las dos fases antes mencionadas:

5.1. Carga de patrones

Es necesario almacenar los diferentes patrones en la memoria de la placa debido a que el proceso de aprendizaje es cíclico (utiliza el conjunto de patrones de forma repetida). Para funciones booleanas sólo es necesario almacenar las salidas, ya que las entradas corresponden a la tabla de verdad formada por la posición que dichas entradas representan. Al tratarse de un clasificador binario sólo es necesario almacenar un bit, por lo que las diferentes posiciones en la memoria para generar la tabla de verdad se realizarán a nivel de bit y no de Byte.
Por ejemplo, para un caso con 8 entradas, el patrón de entrada podría ser “01101001” (105 en valor decimal) con salida ’1’. En este caso se debe almacenar un ’1’ en la posición de memoria 105 de la EEPROM que corresponde al valor en decimal del patrón de entrada. La placa Arduino Uno tiene 1 KB de memoria EEPROM, es decir, 8192 bits (2^{13}); por lo que el número de entradas en las funciones booleanas está limitada a 13.
En el caso de que se empleen tablas de verdades incompletas, ya sea por la eliminación de patrones ruidosos o por la naturaleza de la función, la memoria se dividirá en dos partes; una para identificar la existencia de los patrones y otra para identificar las salidas de dichos patrones. En este caso el número máximo de entradas se reduce a 12.
Si la función es de patrones de valores reales es necesario saber de antemano el número de bits utilizado para representar dicho valor. Se han utilizado 8 bits para representar los valores de las funciones,

teniendo en cuenta que deben estar normalizados entre 0 y 1. Las entradas se almacenan en las primeras posiciones de memoria y las salidas en las últimas, para maximizar el número de patrones que se pueden almacenar en una determinada memoria. A continuación se muestra la ecuación para poder calcular el número máximo de patrones permitido en función del número de entradas que se disponga:

$$N_P \cdot N_I + N_P/8 \leq 1024\ , \tag{10}$$

donde N_I es el número de entradas y N_P es el número de patrones. N_P depende del número de entradas y el número de bits utilizados para cada entrada.

5.2. Aprendizaje de la red neuronal

C-Mantec es un algoritmo constructivo por lo que va agregando nuevas neuronas conforme se van requiriendo; dicha acción no es de fácil implantación en un microcontrolador ya que el manejo de la memoria se hace de forma estática por lo que hay que definir el número máximo de neuronas en el sistema. Se ha establecido un tamaño máximo de memoria para almacenar las variables asociadas a cada neurona, 1KB de memoria SRAM, por lo que se ha dejado disponible otro KB de memoria para almacenar las diferentes variables del algoritmo.
Los microcontroladores son dispositivos cuya velocidad de procesamiento es limitada, por lo que para contribuir a la mayor rapidez del aprendizaje de una red neuronal se ha cambiado la forma de almacenamiento de las variables asociadas a las neuronas y se ha pasado de representación en punto flotante (la utilizada en este tipo de algoritmo para representar los pesos sinápticos y las salidas de la red) a una representación de punto fijo. Este cambio en el paradigma representativo de las variables provoca cambios sustanciales en la forma de programar este algoritmo, aunque como contrapartida obtenemos una mayor velocidad de aprendizaje y un tamaño menor de cada variable ya que se pasa de tipo float a tipo integer en los pesos sinápticos.
Las variables asociadas a las neuronas y sus diferentes tamaños y representaciones son:

- T_{fac}: debe ser una variable tipo *float* y ocupa 4 bytes.
- Número de iteraciones: un valor entero con un rango entre 1.000 y 100.000 iteraciones, por lo que debe ser de tipo *long*, 4 bytes.
- Pesos sinápticos: En casi todos los cálculos está involucrada ésta variable por lo que para acelerar los cálculos se elige tipo *enteros*de 2 bytes de longitud.
- Potencial sináptico (h): Se calcula a raíz de un sumatorio de los pesos sinápticos, por lo que para no saturar este valor se utiliza un tipo *long* de 4 bytes de longitud.

De acuerdo con las definiciones anteriores, el número máximo de neuronas (N_N) que pueden ser utilizadas debe cumplir la siguiente premisa:

$$4 \cdot N_N + 4 \cdot N_N + 2 \cdot N_N \cdot (N_I + 1) + 4 \cdot N_N \leqslant 1024, \tag{11}$$

donde N_I es el número de entradas. Para el cálculo más desfavorable en las funciones booleanas (cuando se disponen de 13 entradas) el número de máximas neuronas permitidas en la implementación es 25.
Los pesos sinápticos y el potencial sináptico se han implementado con 10 bits de precisión para la parte decimal, por lo que el valor de los pesos estará entre 32 y -32. Si se vieran como valores enteros sería entre -32768 y 32767.
El cálculo de T_{fac} se hace usando un tipo punto flotante de datos, ya que requiere una operación exponencial que sólo se puede hacer con este tipo, pero dicho cálculo también implica a otro tipo de datos (enteros),por lo que se deben realizar diferentes conversiones.
La primera conversión se da en la Ecu. 3, en ella habrá que convertir el valor T_{fac} en un valor tipo punto fijo, para ello se multiplica el T_{fac} por 1024 y se cambia su representación a número entero, perdiéndose sólo 2^{-11} de la precisión del valor. La segunda conversión se realiza en la Ecu. 4 donde hay que convertir el potencial sináptico (h) a punto flotante para que pueda ser calculada la exponencial del T_{fac}. En este caso sólo habría que hacer un desplazamiento del potencial sináptico 10 bits a la derecha y después convertirlo a representación punto flotante para que pueda ser operado dentro del cálculo del T_{fac}. Este proceso tiene un inconveniente, la saturación de los pesos sinápticos, ya que cuando éstos son más grandes que 32 y menores a -32 se produce un desbordamiento de la variable, pudiéndose provocar errores graves de cálculo. Para evitarlo se ha realizado una función por la cual cuando los pesos sinápticos alcancen un valor superior a 30 o inferior a -30 todos los pesos son divididos por 2, es decir un desplazamiento a la derecha de 1 bit. Este cambio no afecta en absoluto al procedimiento de la red ya que a las redes neuronales no les afecta este tipo de escalado.

6. Resultados de implementación

En la figura 1 pueden observarse las diferentes funciones del algoritmo C-Mantec. En ella se puede comprobar que dicho algoritmo consta de cinco funciones claramente diferenciadas de las cuales tres de ellas (el cálculo del valor del potencial sináptico y consiguiente salida de la red, el cálculo del T_{fac} y el cambio de los pesos sinápticos) difieren su implementación dependiendo si se realizan con punto fijo o con punto flotante. Mientras que las demás funciones (añadir neurona y el filtrado) es similar en ambas implementaciones.
Se han aislado las funciones en las que las implementaciones en punto fijo y punto flotante difieren, y se ha comprobado el tiempo medio que tardan dichas funciones, según se tenga una implementación u otra. Se han ejecutado las funciones 50 veces y se ha calculado el tiempo medio que tarda cada función en ser resuelta.
En la figura 3 se puede observar el tiempo de cada función con las dos implementaciones (izquierda) y el número de veces que es más rápida la implementación en punto fijo respecto a la implementación en punto flotante. (derecha). En la figura superior izquierda se puede observar el tiempo que tarda el algoritmo en calcular el potencial sináptico y las salidas de la red, en función del número de neuronas. Hay que tener en cuenta que esta función es dependiente del número de entradas. En la central vemos el tiempo que tarda el algoritmo en modificar los pesos de una neurona determina; dicha función es dependiente del número de entradas. En la figura inferior se calcula el valor T_{fac} de cada neurona.
Además se ha probado el correcto funcionamiento de la implementación del algoritmo C-Mantec en la placa Arduino mediante la comparación de resultados, en términos de neuronas obtenidas, de la implementación del microcontrolador y la de la aplicación utilizada en el PC. La prueba también se llevó a cabo para analizar los efectos de una representación de punto fijo que disminuye la precisión de los pesos sinápticos.
Un conjunto de 12 funciones booleanas de salida binaria del MCNC benchmark se ha utilizado para probar la arquitectura de la red del algoritmo C-Mantec, más las funciones XOR de 2 y 3 entradas, haciendo un total de 14 funciones booleanas a estudiar. El algoritmo C-Mantec se realizó con los siguientes valores de parámetros: $g_{fac} = 0{,}05$ y $I_{max} = 1000$ Tabla 4 muestra los resultados obtenidos con el microcontrolador para el conjunto de las funciones de referencia. Las dos primeras columnas indican el nombre de la función y el número de entradas. Las columnas tercera, cuarta y quinta muestran el tamaño de la red neuronal de la implementación en el PC, de la implementación en punto fijo y de la implementación en punto flotante respectivamente. Mientras que las dos últimas columnas muestras el tiempo que tardan las diferentes implementaciones en el microcontrolador. Para realizar dicha tabla se ha lanzado el algoritmo 50 veces por función con el fin de que los resultados sean estadísticamente significativos.

Cuadro 4. Numero de neuronas y tiempo de aprendizaje de la síntesis de un conjunto de funciones para las implementaciones en punto fijo (entero) y punto flotante, comparada además con lo dicho en el paper original.

Función	Nº Entradas	Nº Neuronas			Tiempo (s)	
		Teoría	Entero	Flotante	Entero	Flotante
XOR2	2	$2,0 \pm 0,0$	$2,0 \pm 0,0$	$2,0 \pm 0,0$	$0{,}36 \pm 0{,}01$	$0{,}57 \pm 0{,}03$
XOR3	3	$3,0 \pm 0,0$	$3,0 \pm 0,0$	$3,0 \pm 0,0$	$1{,}4 \pm 0{,}11$	$2{,}22 \pm 0{,}26$
cm82af	5	$3,0 \pm 0,0$	$3,0 \pm 0,0$	$3,0 \pm 0,0$	$1{,}84 \pm 0{,}28$	$3{,}15 \pm 0{,}64$
cm82ag	5	$3,0 \pm 0,0$	$3,64 \pm 0,63$	$3,6 \pm 0,7$	$4{,}11 \pm 1{,}88$	$8{,}1 \pm 4{,}4$
cm82ah	5	$3,0 \pm 0,0$	$3,0 \pm 0,0$	$3,0 \pm 0,0$	$1{,}85 \pm 0{,}21$	$3{,}35 \pm 0{,}63$
z4ml24	7	$1,0 \pm 0,0$	$1,0 \pm 0,0$	$1,0 \pm 0,0$	$0{,}23 \pm 0{,}05$	$0{,}47 \pm 0{,}13$
z4ml25	7	$3,1 \pm 0,0$	$3,11 \pm 0,40$	$3,11 \pm 0,40$	$3{,}36 \pm 1{,}04$	$6{,}43 \pm 2{,}05$
z4ml26	7	$3,0 \pm 0,0$	$3,0 \pm 0,0$	$3,0 \pm 0,0$	$2{,}67 \pm 0{,}45$	$5{,}39 \pm 1{,}11$
9symml	9	$3,0 \pm 0,0$	$3,0 \pm 0,0$	$3,0 \pm 0,0$	$4{,}43 \pm 0{,}58$	$12{,}8 \pm 2{,}061$
alu2k	10	$11,2 \pm 0,0$	$12,7 \pm 0,97$	$12,3 \pm 0,71$	$220 \pm 50{,}1$	969 ± 260
alu2m	10	$2,0 \pm 0,0$	$2,0 \pm 0,0$	$2,0 \pm 0,0$	$3{,}94 \pm 0{,}21$	$13{,}1 \pm 0{,}88$
alu2n	10	$1,0 \pm 0,0$	$1,0 \pm 0,0$	$1,0 \pm 0,0$	$0{,}41 \pm 0{,}15$	$0{,}99 \pm 0{,}41$
alu2o	10	$11,2 \pm 0,0$	$13,0 \pm 0,75$	$12,4 \pm 0,78$	$312 \pm 59{,}6$	1444 ± 824
alu2p	10	$3,0 \pm 0,0$	$3,0 \pm 0,0$	$3,0 \pm 0,0$	$20{,}8 \pm 43{,}7$	$84{,}1 \pm 17{,}2$

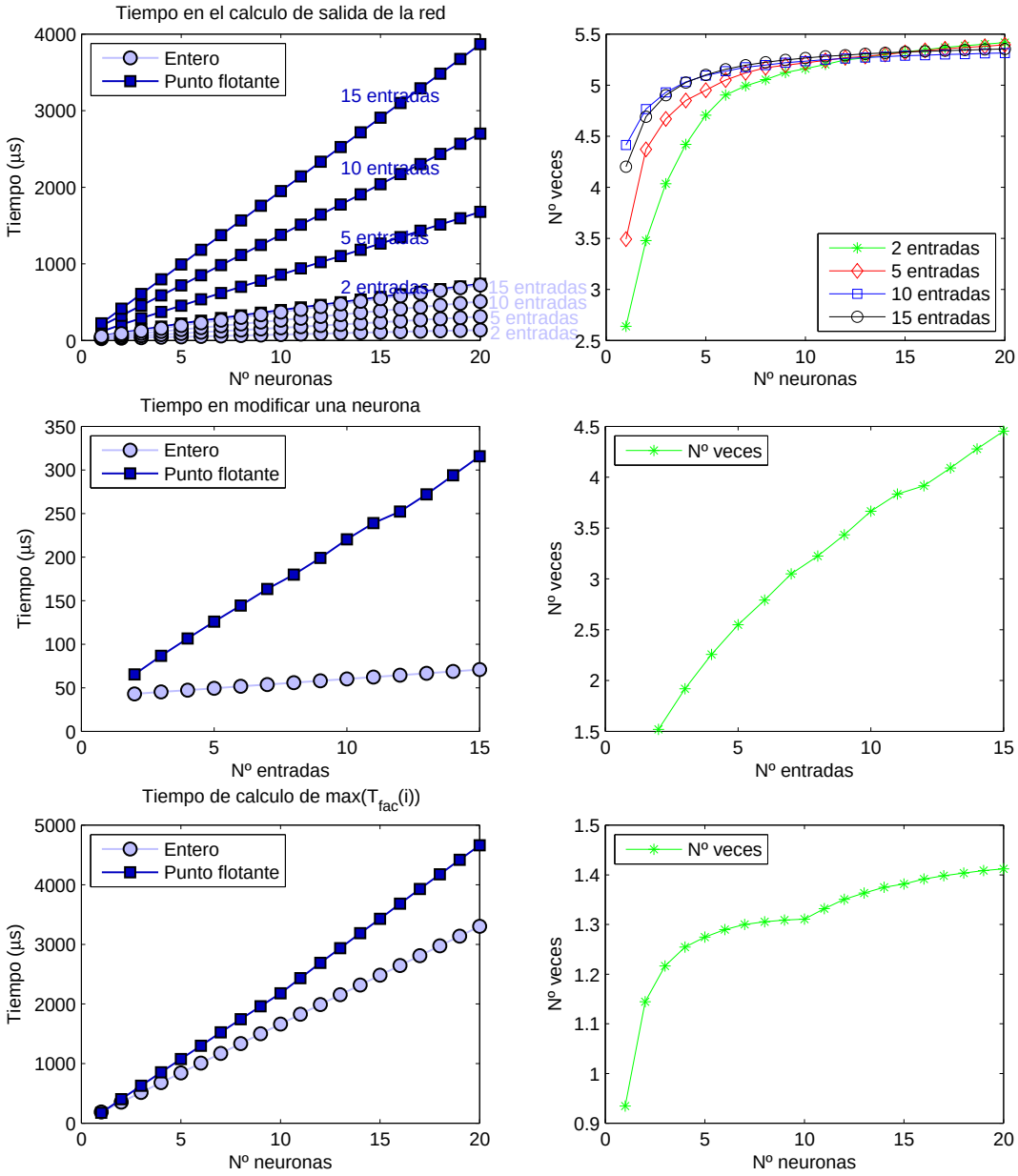

Figura 3. Tiempo de cálculo de las diferentes funciones del algoritmo C-Mantec con las dos implementaciones posibles (gráficas izquierdas) y el número de veces que es más rápida la representación de punto fijo que la de punto flotante (gráficas derechas).

En la figura 4 puede observarse, en la gráfica superior, los diferentes tiempos de aprendizaje de todas las funciones del conjunto seleccionado y, en la gráfica inferior, una función del número de veces que es más rápido la representación en punto fijo que la de punto flotante . La gráfica superior se ha representado en escala logarítmica para el eje y (tiempo de aprendizaje) para que pueda ser visualizada mejor. Para la obtención de dichos datos se ha lanzado cada implementación 50 veces y se han calculados los valores medios que son los representados en la figura.

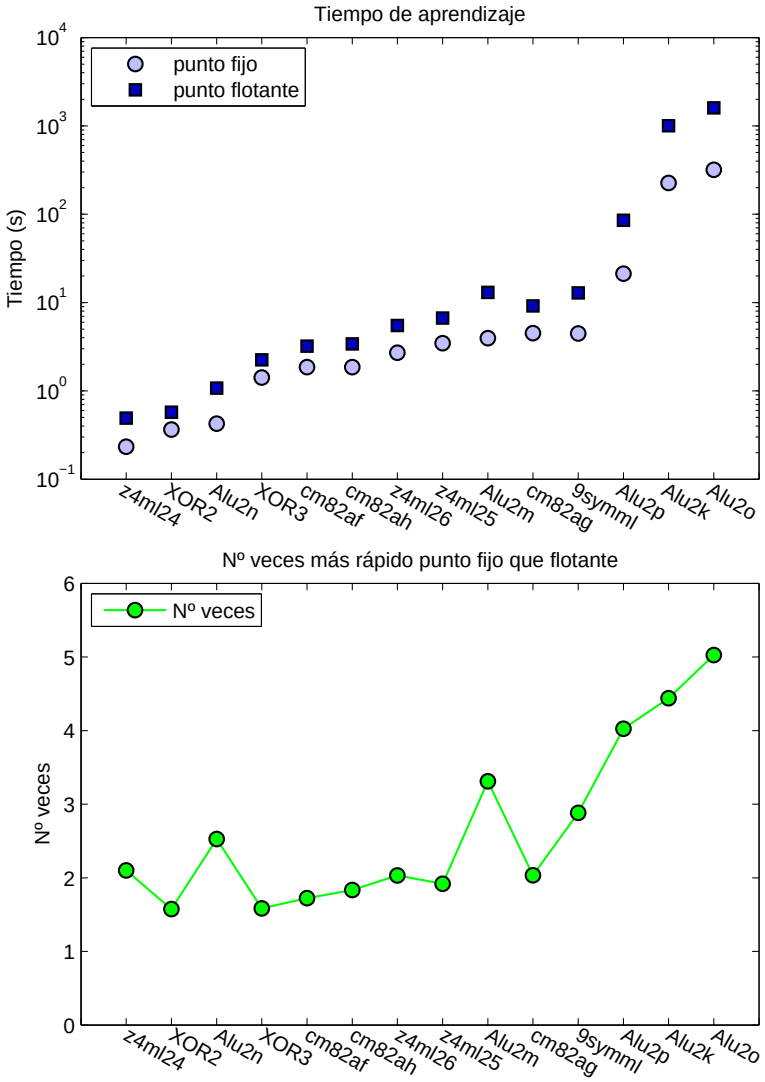

Figura 4. Tiempo de aprendizaje para cada función del conjunto de prueba (gráfica superior) y número de veces que es más rápida la representación de punto fijo que la de punto flotante (gráfica inferior)

En la Fig. 5 se puede observar la evolución temporal de una función determinada. Se ha escogido la Alu2k por ser una función que precisa de un mayor número de neuronas en la capa oculta para ser aprendida y se ha comprobado el tiempo que tarda en llegar a una neurona determina. Por lo que la gráfica superior muestra el tiempo que tarda en llegar a una neurona determinada para ambas implementaciones y en la gráfica inferior el número de veces que es más rápida la implementación en punto fijo que en punto flotante.

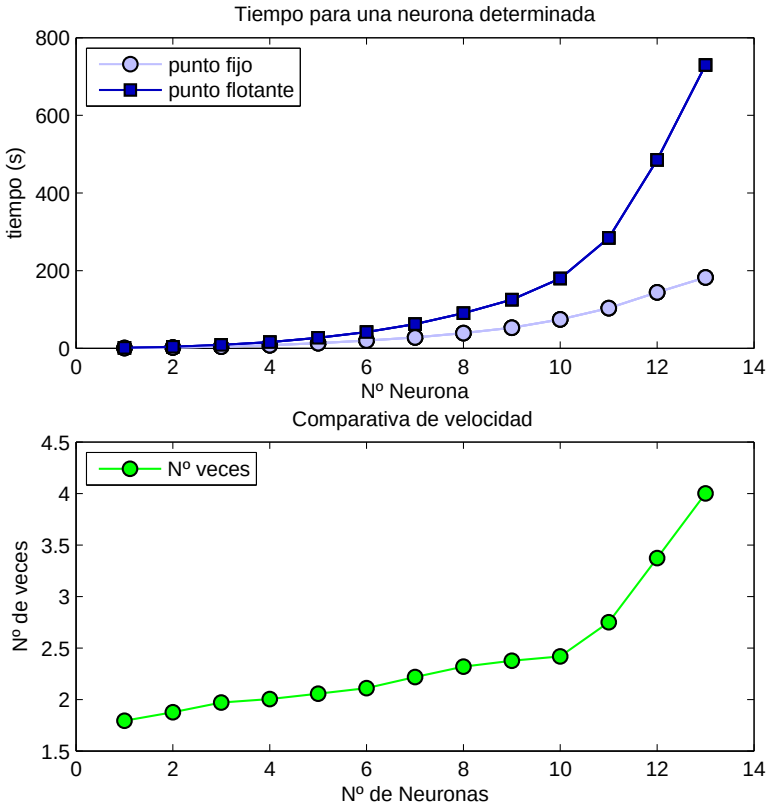

Figura 5. Evolución temporal de la función Alu2K según el tiempo que tarda la aplicación en obtener un tamaño de red determinado (gráfica superior) y el número de veces que es más rápida la representación en punto fijo que la de punto flotante en dicha evolución.

7. Casos de estudios

Con el fin de comprobar la eficiencia de la utilización de redes neuronales en aplicaciones donde sean necesarios el uso de microcontroladores para sensar y/o actuar sobre el entorno, se han considerado 3 casos de estudios de naturaleza dispar, que requieren diferentes tipos de soluciones. El primer caso es la detección de incendios mediante una alarma, para ello se dispone de patrones booleanos que conforman una tabla de verdad completa; el segundo caso es la activación de una válvula de riego según unas variables determinadas ambientales, las cuales se han discretizado para poder ser estudiadas conformando una tabla de verdad, siendo lo más probable que no esté completa; y por último el problema de detección de caídas de personas, este caso formado por patrones reales de los cuales sólo podrán ser aprendidos un número determinado de ellos.

7.1. Alarma de incendios

Existen muchos casos en los que los microcontroladores se usan como sensores/actuadores para controlar una serie de variables y proceder en consecuencia. Un caso muy común es la utilización de este tipo de sistemas en la fabricación de alarmas, como por ejemplo las de detección de incendios.

El sistema capta las diferentes variables encargadas de la detección de incendios (temperatura, humo y gas), se definen ciertos umbrales para los cuales los niveles de dichas variables son peligrosos y el sistema toma una decisión conforme a dichas variables. (Ver Fig. 6).

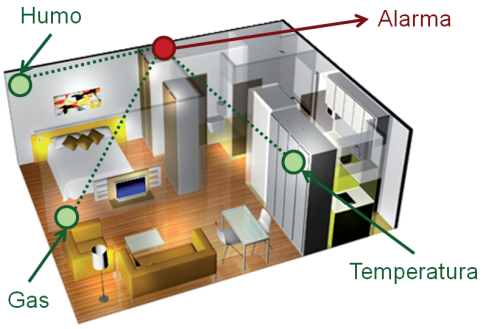

Figura 6. Representación esquemática de una habitación en la cual se ha instalado una alarma de incendios.

Sería lógico suponer que esta acción podría llevarse a cabo mediante la utilización de programación tradicional con el empleo de sentencias "if". El problema surge cuando la decisión tomada de antemano no es la adecuada, ya sea por un mal estudio inicial o por la utilización de dicho sistema en entornos diferentes a los previstos. Por ejemplo si se desarrolla un sistema para la detección de incendio en una habitación y el sistema se coloca en estancias con altas concentraciones de humo, como podría ser la cocina, en este caso el sistema activará la alarma en muchas situaciones en las que no debería.
Siguiendo el método de programación tradicional sería necesario desconectar el sistema, rehacer el estudio previo para adaptarlo a las nuevas condiciones, cargar el nuevo código y restaurar el sistema.
En cambio si utilizamos una red neuronal para este escenario lo único que tendríamos que hacer es que el sistema aprenda el estudio previo y si éste es equivocado, por cualquier motivo, simplemente tendríamos que indicarle su error y el sistema modificará sus patrones, reaprendiendo su nuevo estado sin ningún tipo de interacción o estudio sobre el nuevo escenario; y en particular sin necesidad de interrumpir el sistema.

7.2. Predicción meteorológica

A lo largo de la historia siempre se han intentado hacer predicciones meteorológicas para conocer los cambios climáticos y poder tomar decisiones correctas sobre los cultivos (ya sean tiempo de recolección, momento de riego o tipo de cultivo). Automatizar esta predicción ha sido un problema ampliamente abordado por las redes neuronales.
El sistema captará las diferentes variables ambientales (velocidad (V_V) y dirección (D_V) del viento, temperatura (t), humedad (H) y radiación solar (R)) y se discretizarán sus valores en diferentes escalas dependiendo de qué variable sea. Y la decisión a tomar puede ser la necesidad o no de regar un tipo de cultivo.(ver Fig.7).

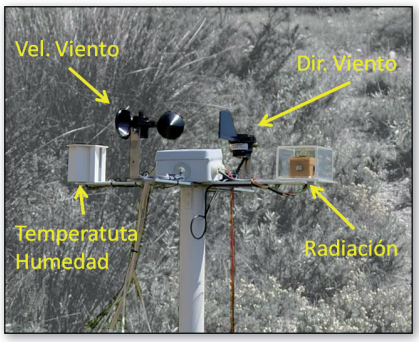

Figura 7. Fotografía de un sistema de sensores encargado de captar las diferentes variables ambientales y actuar en consecuencia.

A priori no se conoce la función de predicción meteorológica para cualquier situación geográfica y además habrá que tener en cuenta sobre qué tipo de cultivo se va a realizar la predicción. Esto hace que un estudio a priori del escenario sea prácticamente imposible.

Con una red neuronal sólo tenemos que ver la evolución de los diferentes cultivos e ir modificando la red neuronal conforme al histórico de datos. Aunque se disponga de una red inicial de partida gracias a un estudio general sobre condiciones climáticas para todo tipo de cultivo, esta red evolucionará conforme a las condiciones microclimáticas del lugar y las necesidades de lo cultivado. Este hecho puede hacer que no dispongamos de todos los datos necesarios para generar de forma completa la tabla de verdad de los patrones del sistema.

7.3. Detección de caídas

Existe una necesidad de hacer un seguimiento a ciertas personas que viven solas, ya sean personas mayores u otras con algún tipo de minusvalía, ya que dichas personas son dependientes y no pueden valerse por ellas mismas. La necesidad principal de seguimiento surge cuando estas personas sufren una caída o un periodo prolongado de inactividad. Hay diferentes formas de abordar este problema, como por ejemplo con cámaras de videovigilancia, aunque este tipo de sistemas tienen un gran inconveniente, la sensibilidad a los cambios de luz producidos en una estancias.

Una forma más eficiente es captar las caídas de una persona utilizando un sensor. Primero colocamos a una persona un acelerómetro de 3 ejes para controlar el ángulo de inclinación y el movimiento (ver Fig. 8). Esta configuración detecta cualquier movimiento anómalo en el individuo que la usa, tales como un fuerte movimiento hacia abajo o de hecho cualquier movimiento brusco o violento.

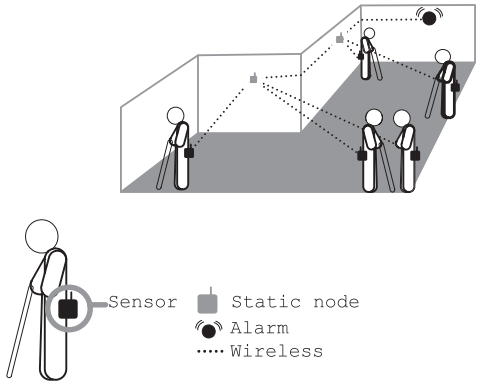

Figura 8. Implementación de un sistema para supervisar las caídas.

Extraer la lógica que describe el comportamiento de las caídas es una tarea ardua y requiere demasiado tiempo. Sin embargo con las redes neuronales esta tarea se simplifica, ya que para extraer una regla de decisión que defina si una persona se ha caído o no, sólo se precisa de unos patrones de entrenamiento y que la red neuronal aprenda a partir de esos patrones de referencia.

C-Mantec necesita datos reales para ser entrenado por lo que para construir la red neuronal que luego prodiga las caídas, se precisa de una recolección de datos previa para poder tener un conjunto de patrones con los que poder entrenar, por lo que el desarrollador no tiene que saber la relación entre los datos recogidos con el fin de detectar la caída.

Después de generar la estructura del modelo de red neuronal, se probó utilizando otros patrones para comprobar la fiabilidad de los resultados de la clasificación. Si la arquitectura es correcta, el modelo de red neuronal interpretará correctamente los datos proporcionados por los sensores y decidirá si ésta información corresponde a una caída o no.

8. Evaluación

Con el fin de comprobar la eficiencia de la solución empleada para los diferentes casos de estudio planteados, se ha realizado un evaluación de cada uno de ellos, comprobando el tiempo de re-aprendizaje, y el consumo del Arduino para poder realizado, y así conocer si es más rentable realizar el reaprendizaje de nuevos escenarios en el propio microcontrolador en lugar de realizarlos por otros métodos.

8.1. Alarma de incendios

La realización de este caso de estudio requiere acoplar diferentes sensores (temperatura, humo y gas) al Arduino, estos sensores se activarán ('1') cuando los niveles medidos superen un determinado umbral, que puede ser modificado y dependerá de cada sensor. Por ejemplo el sensor de temperatura se activará ('1') cuando el valor medido sea superior a 40º C, mientras que para temperaturas inferiores consideraremos que se encuentra desactivado ('0').
Con la activación de los sensores se conforma una tabla de verdad que el sistema deberá aprender para poder decidir en qué estado de alarma se encuentra. Hemos especificado ocho estados de alarma diferentes, ya que éste es el caso más desfavorable, puesto que cada entrada de la tabla de verdad dispone de un estado diferente, dicho caso será el estado inicial del sistema. Los estados de alarma se especifican del 8 menos peligroso al 1 el más peligroso. En la tabla 5 puede observarse la tabla de verdad generada para satisfacer el estado inicial. Donde la columna "alarma" indica el grado de peligro, y la columna "representación" indica la representación binaría que tendrá cada alarma. Cabe destacar que la representación binaria se realiza ya que el algoritmo C-Mantec es un clasificador binario y de esta forma se puede realizar una clasificación multiclase.

Cuadro 5. Tabla de verdad del estado inicial del sistema para la detección incendio.

Gas	Humo	Temperatura	Alarma	Representación
0	0	0	8	0 0 0
0	0	1	7	0 0 1
0	1	0	6	0 1 0
0	1	1	5	0 1 1
1	0	0	4	1 0 0
1	0	1	3	1 0 1
1	1	0	2	1 1 0
1	1	1	1	1 1 1

En primera instancia se ha comprobado el tiempo que tarda el sistema en aprender el estado inicial, con cuántas neuronas ha sido capaz de resolverlo y la potencia consumida en hacerlo. En la tabla 6 pueden observarse los datos mencionados.

Cuadro 6. Valores de tiempo, consumo y número de neuronas del proceso de aprendizaje del estado inicial.

Nº Neuronas	Tiempo (ms)	Consumo(nAh)
$3{,}0 \pm 0{,}0$	$8{,}228 \pm 4{,}88$	$73{,}13 \pm 43{,}4$

Después se ha comprobado el tiempo que tarda el sistema en re-aprender un cambio en el estado de una alarma de una entrada concreta de la tabla, con el fin de evaluar la eficiencia de esta implementación en comparación con una programación tradicional en la que se necesitaría recargar el código programado. En la Fig. 9 puede observarse el tiempo que tarda y su correspondiente consumo en el proceso de re-aprendizaje de un cambio en el estado de una alarma, en función del estado inicial de dicha alarma (eje abscisa); además se puede observar el tiempo y consumo medios producidos por el proceso de re-aprendizaje.

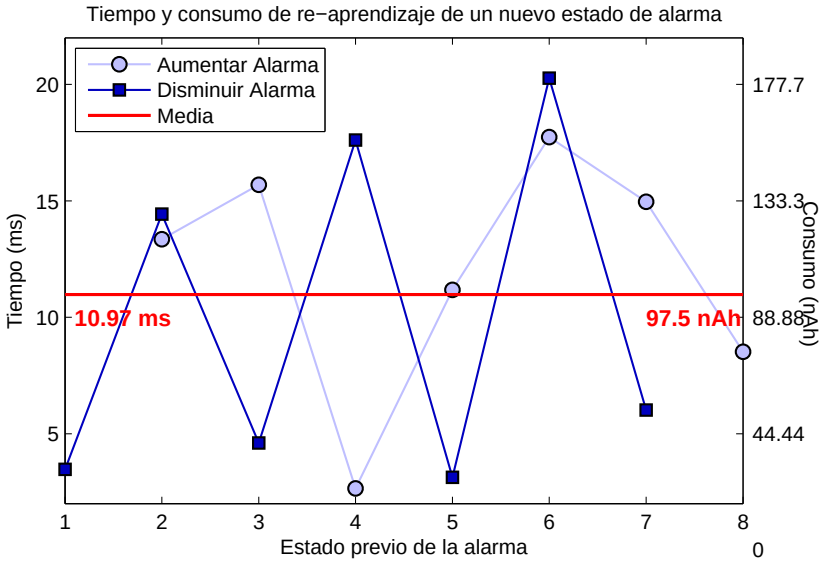

Figura 9. Tiempo y consumo del proceso de re-aprendizaje de cambiar un estado de una determinada alarma ya sea aumentar o disminuir, en el eje de abscisa se representa la gravedad de la alarma.

8.2. Predicción climática

Con el fin de evaluar la implementación de la predicción climática para la apertura de una válvula de riego en un cultivo determinado, se deben discretizar las diferentes variables. Se dispone solamente de 12 bits para realizar la representación, ya que lo normal es que no se dispongan de toda la información para completar la tabla de verdad. Se han escogido 2 bits para la dirección del viento(norte, sur, este, oeste), 2 bits para la velocidad del viento ($\leq 1{,}5\ m/s$, $(1{,}5-3]\ m/s$, $(3-5]\ m/s$, $\geq 7{,}5\ m/s$), 2 bits para la humedad ($\leq 20\,\%$, $(20-50]\,\%$, $(50-80]\,\%$, $\geq 80\,\%$),2 bits para la radiación ($\leq 150\ W/m^2$, $(150-400]\ W/m^2$, $(400-800]\ W/m^2$, $\geq 800\ W/m^2$)y 4 bits para la temperatura($-5:2{,}5:35\ ^oC$).

Gracias a la discretización realizada a las variables, se pueden realizar instancias que corresponden a una entrada de la tabla de verdad, por lo que se considera un patrón de la función a aprender. Las instancias son del tipo:

- Si $15 \leq t \leq 17{,}5$ AND $Vv \leq 1{,}5\ m/s$ AND $Dv = (N)$ AND $50\,\% \leq H \leq 80\,\%$ AND $R \geq 800\ W/m^2$ ->"abierta".

Esta instancia corresponde con la entrada de la tabla de verdad "0100 00 00 10 11" que tiene como salida '1'. La tabla de verdad del sistema tiene por lo tanto 4096 entradas diferentes, el caso peor es el que se desconocen todas las instancias del sistema y no se tiene ninguna entrada de la tabla a rellenar. Dichas entradas se irán rellenando conforme el sistema vaya recopilando información del entorno, ésta se puede extraer de forma automática, aunque la opción más viable es mediante un "experto" (persona encargada del cultivo") que indique al sistema que salidas deben tener los diferentes estado conforme el sistema vaya pasando por ellos.

Para poder realizar un estudio más completo hemos completado la tabla de verdad, después se ha comprobado el tiempo que tarda el sistema en aprender cada uno de las entradas y la precisión que tiene la red neuronal resultante comparada con la tabla completa. Las entradas se han ido aprendiendo de manera aleatoria para considerar el caso peor. En la Fig. 10 se puede comprobar el tiempo que tarda el sistema en aprender todas las entradas y el correspondiente consumo por parte del Arduino en realizarlo (superior), Además se puede observar la precisión conforme más instancias son aprendidas (inferior).

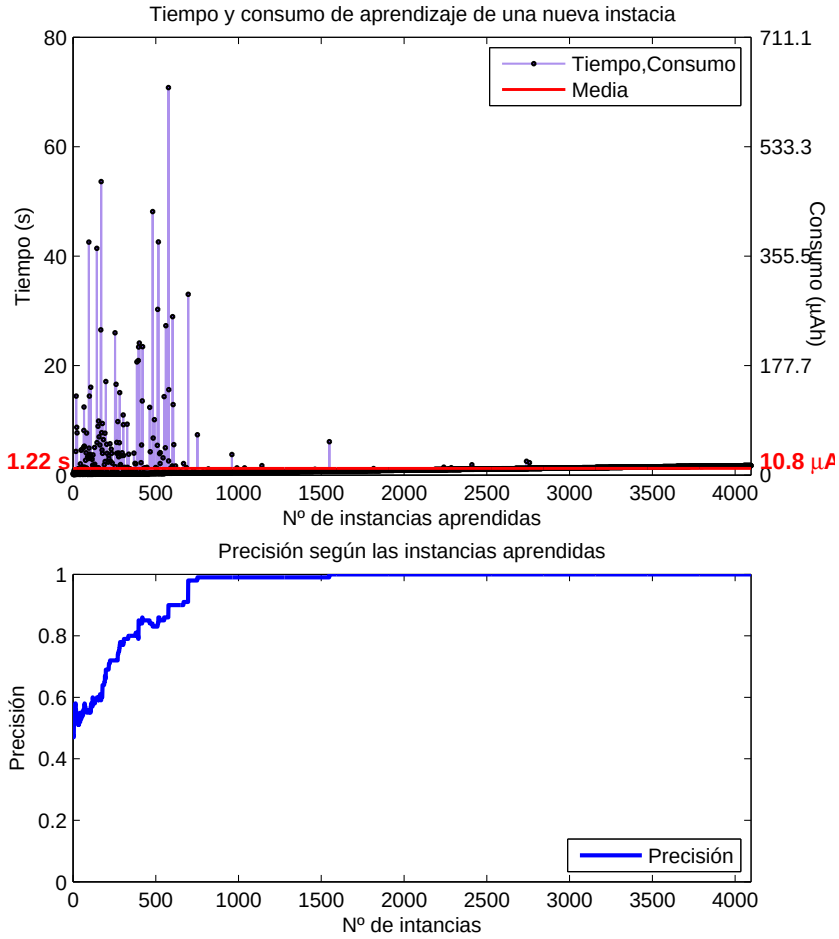

Figura 10.

Con el fin de evitar el aprendizaje de los patrones iniciales, se podría hacer un estudio precio y empezar el sistema con un número de entradas ya aprendidas, esto supondría un ahorro de tiempo y consumo para el sistema. Además se podría especificar instancias que abarquen más de una entrada de verdad de nuestra tabla para tener menos pasos que aprender. Por ejemplo la instancia:

- Si $15 \leq t \leq 25$ AND $Vv \leq 1{,}5\ m/s$ AND $Dv = (N, W)$ AND $50\,\% \leq H \leq 80\,\%$ AND $R \geq 800\ W/m^2$ ->"abierta"

corresponde a 8 entradas de la tabla de verdad ya que abarca un rango mayor de temperaturas (4 franjas) y mayor rango en cuanto a la dirección del viento (2 direcciones diferentes) hacen que esta instancia corresponda con 8 entradas diferentes de la tabla de verdad. De esta forma se podría conseguir un avance significativo en la eficiencia de la aplicación.

8.3. Predicción de caídas

La evaluación de este caso ha requerido de diferentes actuaciones, ya que es necesario la utilización de la placa Arduino junto a un sensor acelerómetro que captura la posición del dispositivo en los ejes (x,y,z). La posición captada es muestreada por el microcontrolador que normaliza los datos para que éstos tengan un rango entre 0 y 255, con el fin de almacenarlos en una variable tipo "byte" y así ser almacenados en menos espacio.
Se considerará una caída cuando en un breve periodo de tiempo (2 segundos) la posición del dispositivo pasa de un estado inicial vertical a un estado final horizontal. Por lo tanto el microcontrolador deberá

almacenar 2 posiciones (estado inicial y final), cada una de las cuales tiene 3 variables (x,y,z) por lo que el sistema pasa a ser una función con 6 variables.

El número de patrones que pueden ser almacenados en el sistema debe cumplir la ecu. 10, por lo cual el número máximo de patrones, en este caso, es de 167 patrones de 6 variables. Debido a esto por mucho que se dispongan más patrones para ser aprendidos, si queremos aprender un nuevo patrón debemos desaprender otros, esto hará que el sistema pierda cierta precisión en la toma de decisiones.

Conseguir patrones de test es el primer paso para comprobar la eficiencia de nuestra implementación. Primero se ha colocado el microcontrolador junto con el acelerómetro en una persona y se realiza la acción de caída reiterativamente para poder guardar esos patrones. Se han obtenido 4400 patrones de test o referencia.

Después comprobamos como evolucionaría el sistema si partimos de 0 con el microcontrolador y poco a poco se van captando diferentes patrones. En la figura 11 "superior" puede observarse la evolución de la precisión de la red neuronal que se va calculando conforme se obtienen los diferentes patrones, y en la "inferior" puede observarse el tiempo que tarda en aprender cada patrón y el consumo de la placa Arduino en realizarlo. Hay que tener en cuenta que a partir del patrón 167 es cuando para añadir un patrón se ha de eliminar otro.

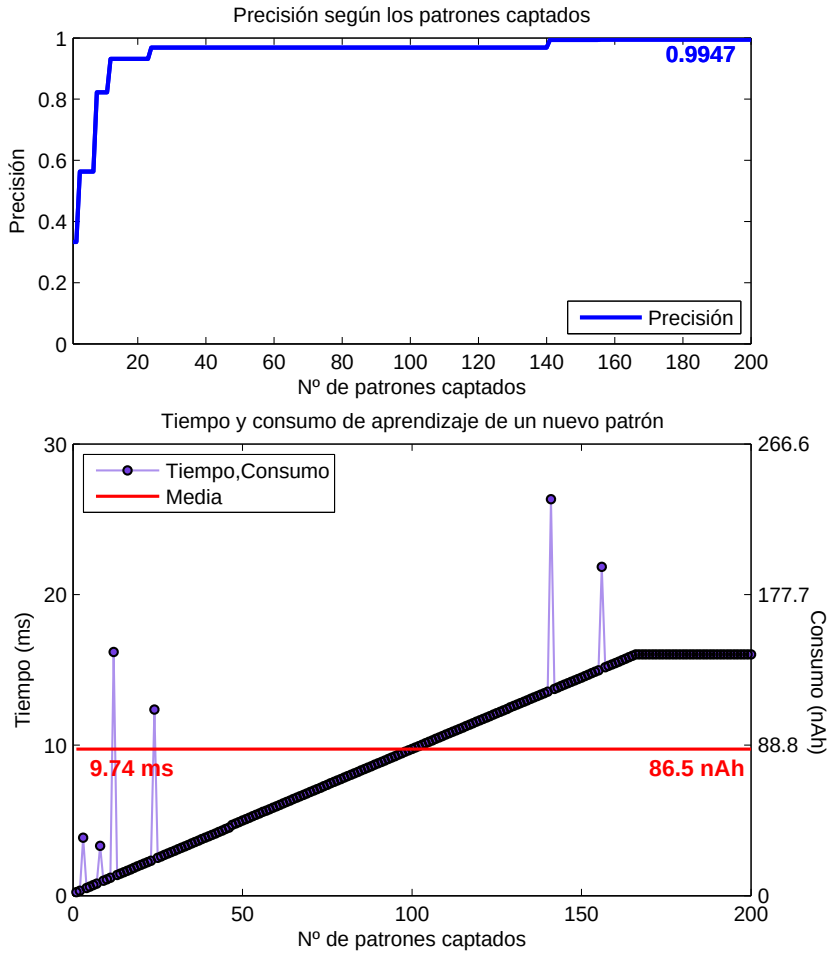

Figura 11. Precisión de la red neuronal obtenida en función del número de patrones que se hayan aprendido(supeior) y tiempo de aprendizaje y consumo en realizarlo (inferior).

9. Conclusiones

Se ha aplicado en este trabajo el algoritmo de red neuronal constructivo C-Mantec a una placa Arduino, solucionando los principales problemas que surgen a la hora de programar el mismo, los cuales vienen dado por las limitaciones de la placa en cuanto a tamaño de memoria y velocidad de cómputo. En este sentido se han propuesto diversas modificaciones de la implementación para suplir dichas carencias. Se ha analizado también, el número máximo de patrones que pueden ser almacenados, ya sean estos representado por valores reales o booleanos, observándose como las modificaciones propuestas maximizan el número de patrones que se pueden almacenar. Para el caso de patrones booleanos, se ha llevado a cabo una comparación entre una programación en representación de punto flotante, utilizada tradicionalmente en estos algoritmos, y la modificación en punto fijo que se ha propuesto, quedando demostrado que la implementación propuesta para el aprendizaje en todos los casos analizados, siempre es más rápida que la programación tradicional.
También se ha comprobado la correcta implementación del algoritmo, y para ello se han comparado los resultados obtenidos con los datos del artículo original [5], observándose que a medida que el número de entradas de las funciones aumentan, la aplicación Arduino necesita sólo una pequeña cantidad adicional de neuronas en el aprendizaje. Este efecto puede estar relacionado con la limitación inherente a la representación en punto fijo que se utiliza para los pesos sinápticos. Los efectos de redondeo no deberían, en principio, degradar el funcionamiento del algoritmo, pero si es verdad que afecta en cuanto al número de iteraciones necesarias para lograr la convergencia.
Para finalizar, se ha analizado la posibilidad de adaptar el sistema a diferentes escenarios, elegidos de naturaleza dispar para que se deban resolver con enfoques diversos, con el fin de demostrar su polivalencia. Se ha comprobado que el tiempo empleado en cada caso es significativamente más bajo, con lo que el consumo en su realización también lo será. Puede concluirse que el método propuesto parece una muy buena solución para diferentes aplicaciones, ya que compite claramente en tiempo y consumo con los métodos tradicionales, en particular en el proceso de adaptación a escenarios cambiantes.

Agradecimientos

El autor agradece el apoyo de la Junta de Andalucía a través del proyecto P10-TIC-5770.

Referencias

1. Yick, J., Mukherjee, B., Ghosal, D.: Wireless sensor network survey. Comput. Netw. **52**(12) (August 2008) 2292–2330
2. Marwedel, P.: Embedded System Design. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
3. Kopetz, H.: Real-Time Systems: Design Principles for Distributed Embedded Applications. 1st edn. Kluwer Academic Publishers, Norwell, MA, USA (1997)
4. Andersson, A.: An Extensible Microcontroller and Programming Environment. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science (2003)
5. Subirats, J., Franco, L., Jerez, J.: C-mantec: A novel constructive neural network algorithm incorporating competition between neurons. Neural Networks **26** (2012) 130–140
6. Oxer, J., Blemings, H.: Practical Arduino: Cool Projects for Open Source Hardware. Apress, Berkely, CA, USA (2009)
7. Haykin, S.: Neural networks: a comprehensive foundation. Prentice Hall (1994)
8. Gómez, I., Franco, L., Jerez, J.: Neural network architecture selection: Can function complexity help? Neural Processing Letters **30** (2009) 71–87
9. Hunter, D., Hao, Y., Pukish, M., Kolbusz, J., Wilamowski, B.: Selection of proper neural network sizes and architectures – a comparative study. IEEE Transactions on Industrial Applications **8** (2012) 228–240
10. Lakshmi, K., Subadra, M.: A survey on fpga based mlp realization for on-chip learning. In: International Journal of Scientific & Engineering Research. Volume 4. (2013)
11. Franco, L., Elizondo, D., Jerez, J.: Constructive Neural Networks. Springer-Verlag, Berlin (2009)

12. Salewski, F., Taylor, A.: Fault handling in fpgas and microcontrollers in safety-critical embedded applications: A comparative survey. In: Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, IEEE Computer Society (2007) 124–131
13. Buccella, C., Cecati, C., Latafat, H.: Digital control of power converters - a survey. IEEE Trans. Industrial Informatics **8**(3) (2012) 437–447
14. Malinowski, A., Yu, H.: Comparison of embedded system design for industrial applications. IEEE Trans. Industrial Informatics **7**(2) (2011) 244–254
15. Vieira, M.A.M., Coelho, Silva, D., da Mata, J.M.: Survey on wireless sensor network devices. In: Proc. IEEE Conf. Emerging Technologies and Factory Automation ETFA '03. Volume 1. (2003) 537–544
16. Cela, A., Yebes, J.J., Arroyo, R., Bergasa, L.M., Barea, R., López, E.: Complete low-cost implementation of a teleoperated control system for a humanoid robot. Sensors **13**(2) (2013) 1385–1401
17. Kornuta, J.A., Nipper, M.E., Brandon Dixon, J.: Low-cost microcontroller platform for studying lymphatic biomechanics in vitro. Journal of Biomechanics (November 2012)
18. Zachariadou, K., Yiasemides, K., Trougkakos, N.: A low-cost computer-controlled arduino-based educational laboratory system for teaching the fundamentals of photovoltaic cells. European Journal of Physics **33**(6) (2012)
19. alt1040: 10 usos creativos que podemos darle a arduino. http://alt1040.com/2013/04/usos-creativos-de-arduino
20. Urda, D., Canete, E., Subirats, J.L., Franco, L., Llopis, L., Jerez, J.M.: Energy-efficient reprogramming in wsn using constructive neural networks. International Journal of Innovative, Computing, Information and Control **8** (2012) 7561–7578
21. E. Canete, E., Chen, J., R.Luque, Rubio, B.: Neuralsens: A neural network based framework to allow dynamic adaptation in wireless sensor and actor networks. J. Network and Computer Applications **35**(1) (2012) 382–393
22. Farooq, U., Amar, M., ul Haq, E., Asad, M.U., Atiq, H.M.: Microcontroller based neural network controlled low cost autonomous vehicle. In: Proceedings of the 2010 Second International Conference on Machine Learning and Computing. ICMLC '10, Washington, DC, USA, IEEE Computer Society (2010) 96–100
23. S.Kumaravel, P.Neelamegam, R.Vasumathi: Article:distributed chloride prediction system using neural network and pic18f452 microcontrollers in water analysis. International Journal of Computer Applications **8**(14) (October 2010) 15–20
24. Susnea, I.: Distributed neural networks microcontroller implementation and applications. Studies in informatics and control **21**(2) (2012) 165–172
25. Aleksendrić, D., Jakovljević, I., Irović, V.: Intelligent control of braking process. Expert Syst. Appl. **39**(14) (October 2012)
26. Ou, G., Murphey, Y.L.: Multi-class pattern classification using neural networks. Pattern Recogn. **40**(1) (January 2007) 4–18
27. Subirats, J.L., Jerez, J.M., Gómez, I., Franco, L.: Multiclass pattern recognition extension for the new c-mantec constructive neural network algorithm. Cognitive Computation **2**(4) (2010) 285–290
28. Atmel: Datasheet 328. http://www.atmel.com/Images/doc8161.pdf