Facultad de Ciencias Exactas
y Naturales - FCEyN
Universidad de Buenos Aires - UBA
República Argentina

Licenciatura en Ciencias de la Computación

Tesis de Licenciatura
Extensión del sistema operativo lejOS

Tesistas
Pablo Fernández
Mario Sassone

Director
Roberto Bevilacqua

Noviembre de 2002

Abstract

A classic problem in robotics is the visual pattern recognition. Although this is an easy task for human beings, it presents significant difficulty in trying to do the same task automatically using a robot.

One solution for this problem can be modeled using a method of computation used in the artificial intelligence field named neural networks. The goal of a neural network is to simulate the human like performance, being them very good at interpreting vague, noisy and incomplete input.

Neural networks are trained from experience. The goal is to adapt the parameters of the network so that they perform well for patterns from outside the training set. Particularly, the back-propagation learning algorithm, used in a neural network, is currently one of the most popular methods for performing the visual pattern recognition task.

When we tried to model this kind of solution with leJOS, the operating system developed by the Internet community for the LEGO's RCX, we identified that although it presents several robotics extensions, it doesn't contain any software package related to the learning concept to enable the development of intelligent applications programs.

In consequence, the purpose of this thesis will be the design, development and test of an extension for the leJOS operating system in order to enable us to include the learning capabilities in robotics application programs.

Resumen

Un problema clásico en robótica es el de la clasificación visual de patrones. Si bien esta es una tarea simple para un ser humano, presenta un alto grado de dificultad el intentar realizar la misma de una forma automática por medio de un robot.

Una solución a este problema puede modelarse por medio de un método de computación utilizado en el campo de la inteligencia artificial denominado redes neuronales. Estas redes tienen por objetivo "simular" el comportamiento humano siendo las mismas muy adecuadas para la interpretación de información vaga e incompleta.

Mediante diferentes técnicas de aprendizaje, las redes neuronales son entrenadas por medio de la experiencia. El objetivo es adaptar la red neuronal de manera tal que pueda reconocer patrones fuera del conjunto de entrenamiento. En particular, el algoritmo de aprendizaje back-propagation en una red neuronal es uno de los métodos más populares para llevar a cabo la tarea de clasificación visual de patrones.

Al intentar modelar este tipo de solución con lejOS, el sistema operativo desarrollado por la comunidad de Internet para el RCX de LEGO, identificamos que si bien presenta una serie de extensiones orientadas a la robótica, no contiene ningún paquete dedicado específicamente al concepto de aprendizaje, que nos facilite el desarrollo de programas de aplicación dotados de "inteligencia".

En consecuencia, el propósito de la presente tesis será el diseño, desarrollo y prueba de una extensión al sistema operativo lejOS que nos permita incorporar el concepto de aprendizaje en programas de aplicación orientados a la robótica.

Tabla de Contenidos

Indice de Figuras

Introducción

Como resultado de un trabajo de investigación conjunto entre el Massachusetts Institute of Technology (MIT) y LEGO, en 1998 se creó Mindstorms, una familia de productos LEGO que impulsan la creación de robots caseros controlados por computadora.

El cerebro de estos robots es el RCX, un bloque especial que contiene un microprocesador Hitachi H8/3292. Aparte del microprocesador, el RCX contiene una ROM de 512 bytes, una RAM de 32 Kbytes, tres puertos de entrada que permiten conectar distintos tipos de sensores, tres puertos de salida que permiten conectar motores, un puerto infrarrojo y un display de cristal líquido (LCD).

Si bien el RCX viene con un software de fábrica orientado a usuarios con poca experiencia en programación, fue la comunidad de Internet la que potenció realmente el producto creando distintos sistemas operativos que permitan programar las tareas de este bloque.

En particular, lejOS es un sistema operativo opensource desarrollado en java y que puede ser comparado con Java 2 SDK de Sun Microsystems. Algunas características de este sistema operativo son: una JVM que interpreta bytecodes a ser cargados en el RCX, y una API que permite el manejo de la plataforma (sensores y motores) como así también el manejo de números de punto flotante, multithreading, arrays, eventos y recursión.

Definición del problema

Un problema clásico en robótica es el de la clasificación visual de patrones, como por ejemplo, el reconocimiento de caracteres o de formas. Si bien esta es una tarea simple para un ser humano, presenta un alto grado de dificultad el intentar realizar la misma de una forma automática por medio de un robot.

La solución a este problema se puede modelar por medio de un método de computación utilizado en el campo de la inteligencia artificial denominado redes neuronales. Estas redes tienen por objetivo “simular” el comportamiento humano siendo las mismas muy adecuadas para la interpretación de información vaga e incompleta.

Las redes neuronales son entrenadas por medio de la experiencia. En lo que se denomina aprendizaje supervisado, una red neuronal es adaptada para identificar una serie de patrones por medio de un conjunto de patrones de entrenamiento. El objetivo es adaptar la red de manera tal que pueda reconocer patrones fuera del conjunto de entrenamiento. En particular, el algoritmo de aprendizaje back-propagation en una red neuronal multicapa es uno de los métodos más populares para llevar a cabo la tarea de clasificación visual de patrones.

Al intentar modelar este tipo de solución con lejOS, se identificó que si bien este sistema operativo cuenta con una serie de extensiones orientadas a la robótica, no existe ningún paquete dedicado específicamente al concepto de aprendizaje. Si bien la API del sistema operativo lejOS contiene un conjunto de paquetes que permite controlar el hardware del RCX (sensores y motores), tiene sentido incluir y definir una API orientada al aprendizaje, que facilite el desarrollo de aplicaciones dotadas de "inteligencia".

Propósito

En consecuencia, el propósito de la presente tesis será el desarrollo de una extensión al sistema operativo lejOS que permita incorporar el concepto de aprendizaje en sus programas. Para ello, estas serán las principales actividades a llevar a cabo:

- Abstraer y modelar las clases que implementen el algoritmo de redes neuronales identificado como back-propagation.
- Diseñar y construir las clases junto a sus métodos para permitir el desarrollo de programas que implementen las mismas.
- Presentar un ejemplo real del funcionamiento de esta extensión.

Organización de nuestro trabajo

A los fines de facilitar la lectura de la presente tesis, a continuación presentaremos una breve descripción de las secciones que componen la misma:

Introducción

En esta sección haremos un primer repaso del objetivo de nuestro trabajo, presentando el marco de referencia y el problema a resolver.

Marco de referencia

En esta sección presentaremos información general relativa a Lego Mindstorms, Lejos y redes neuronales. Esta información permitirá al lector tener una introducción general a los conceptos necesarios para poder entender el objetivo de nuestra tesis.

Enfoque de la solución

A lo largo de esta sección describiremos cuál es el enfoque que adoptamos en la resolución del problema establecido en la sección anterior. Nuestra solución comprende dos aspectos principales: la arquitectura de software utilizada y el hardware o modelo de robot construido para demostrarla.

Demostración

En esta sección presentaremos el caso de ejemplo que permite presentar nuestra solución propuesta.

Conclusiones y trabajos futuros

Entendemos que nuestro trabajo tiene varios puntos que pueden ser desarrollados con mayor detalle. En esta sección describiremos cuáles pueden ser las posibles extensiones a nuestro trabajo.

Referencias

En esta sección se mencionarán todas las fuentes de información utilizadas en el desarrollo de nuestro trabajo. Las mismas incluyen libros, websites, etc.

Marco de Referencia

Lego Mindstorms

Unidad de procesamiento central (RCX)

El RCX contiene un microprocesador Hitachi H8/3292. Este microprocesador tiene una frecuencia de reloj en el RCX de 16 Mhz. Si bien el mismo aparece como extremadamente lento comparado con la frecuencia de reloj de una PC actual, es lo suficientemente rápido como para prender y apagar motores, leer la entrada de los sensores, y tomar decisiones. A continuación detallaremos cada uno de los componentes de de la arquitectura técnica del RCX.

Figura 1 – Ladrillo RCX y sus componentes

ROM

Este microprocesador contiene 16 Kbytes de memoria de solo lectura (ROM) y 512 bytes de memoria on-board de acceso aleatorio. La memoria ROM contiene algoritmos para bajar el firmware, presentar datos en el display de cristal líquido (LCD) y comunicarse con los motores y sensores. Este tipo de programas es comúnmente referido como firmware, pero LEGO llama de la misma manera también a la parte de los programas que carga en el RCX. La memoria ROM contiene también algunos programas que pueden ser utilizados como testers del RCX. La memoria RAM es utilizada para almacenar instrucciones para la CPU.

RAM

El RCX contiene 32 Kbytes de memoria RAM. En esta memoria se almacenan dos componentes: el firmware y los programas de usuario. El firmware es esencialmente una extensión de la ROM, excepto que éste puede ser actualizado y reemplazado para proveer mayor funcionalidad. Si bien los 32 KBytes aparecen como pequeños comparados con la cantidad de memoria de una PC de hoy en día, es lo suficientemente amplio para la mayoría de las aplicaciones de robótica. Es más el RCX contiene más memoria de la que fue utilizada por el módulo lunar utilizado en las misiones a la Luna.

Puertos de entrada

El RCX contiene tres puertos de entrada que aceptan una variedad de sensores tales como sensores de tacto, luz, temperatura y rotación.

Puertos de salida

El RCX contiene tres puertos de salida utilizados para controlar actuadores. Estos puertos son utilizados para controlar motores como así también luces. También son utilizados para controlar pequeños actuadotes, como válvulas neumáticas y sensores caseros.

Botones

Existen cuatro botones en el RCX: On-Off, Run, View y Prgm. Estos botones permiten una interacción del usuario con el robot para controlar la ejecución del programa.

Puerto de comunicaciones infrarrojo serial

El principal propósito del puerto infrarrojo es para bajar programas al RCX. El puerto permite una transmisión de datos de doble vía, lo que permite que el RCX también pueda enviar datos de regreso a la PC. Este puerto puede ser utilizado también para permitir que dos RCX se comuniquen entre sí.

Speaker

El speaker permite producir una variedad de sonidos simples de frecuencia y duración variables.

LCD

El LCD por medio del firmware original es utilizado para presentar el estado interno del RCX. Por ejemplo puede presentar valores de los sensores de entrada, indica si un programa está en ejecución o permite que el programador envíe información numérica al display.

Baterías

El RCX es alimentado por medio de seis baterías AA.

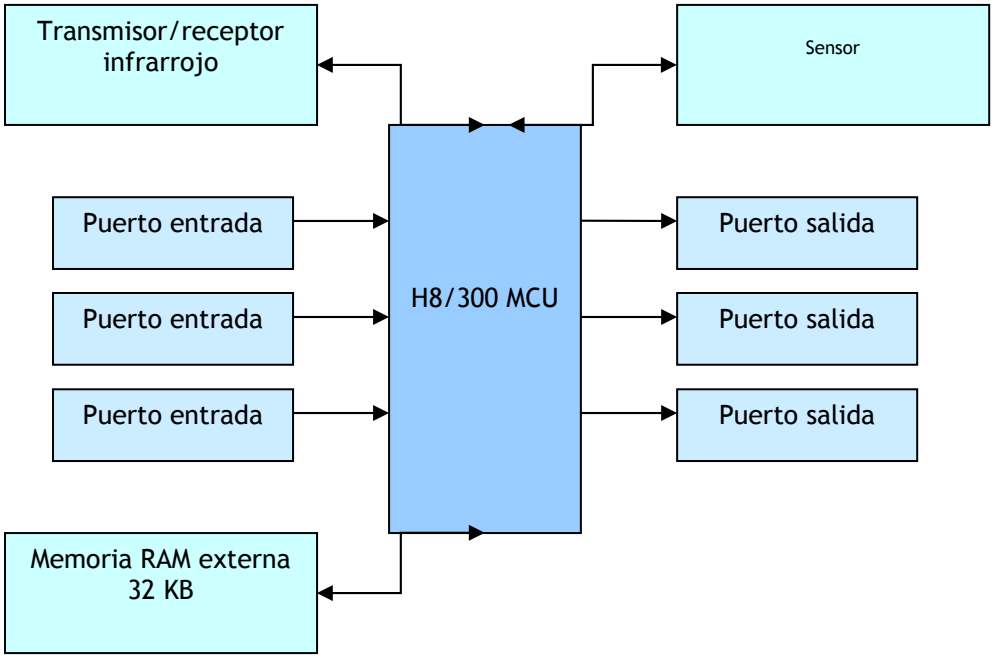

Figura 2 – Arquitectura hardware RCX

Torre Infrarrojos (IR)

Mindstorms utiliza una forma novedosa para transmitir los programas de la PC al RCX. En lugar de enviarlos por medio de un cable, los datos son transmitidos por medio de señales infrarrojas, de la misma manera que un control remoto de televisión. La torre IR también puede ser utilizada para intercambiar datos con el RCX mientras se ejecuta un programa. La torre IR puede también enviar comandos al RCX permitiendo que la PC actúe como un controlador remoto.

Motores

Los motores eléctricos del RCX contienen una serie de engranajes que le otorgan un mayor torque al eje. Esto resulta en una rotación más lenta del motor llegando a tener un máximo de 350 rpm. La energía del motor puede ser controlada por medio del software. El RCX lleva a cabo este control por medio de un enfoque identificado como modulación de ancho de pulsos y que consiste básicamente en aplicar tensión al motor a intervalos regulares. Estos intervalos pueden regularse, obteniendo como resultado final una regulación en la energía aplicada al motor.

Sensores

Los sensores del RCX obtienen datos del mundo real y los envían al RCX para que éste pueda tomar decisiones. Los sensores son sumamente importantes para el robot ya que permiten obtener conclusiones acerca del ambiente externo y permiten determinar que hacer en consecuencia. El RCX puede estar conectado con el mundo real por medio de cuatro tipos de sensores:

Sensor de luz

El sensor de luz consiste de un LED y un fototransistor que responde a la luz que recibe. Este sensor lee valores dentro del rango de 0 a 100 siendo 100 el valor correspondiente a la lectura con más brillo.

Sensor de tacto

Este es uno de los sensores más sencillos. Es utilizado para detectar cuándo el robot entra en contacto con alguna superficie, como por ejemplo cuando un robot móvil choca contra una pared. Este sensor lee dos únicos valores: 0 cuando no está contacto con ningún objeto y 1 en caso contrario.

Sensor de rotación

Este sensor permite medir la rotación relativa de un eje que se encuentre conectado al mismo. Permite obtener cuál es el ángulo de rotación midiéndolo contra una posición base. Este sensor lee valores que van de 0 a 15, representando cada paso un incremento angular de 22.5 grados.

Sensor de temperatura

Por medio de este sensor es posible capturar la temperatura ambiente. El mismo lee valores que pueden ser convertidos a grados Celsius o Fahrenheit en el rango de -20 a 70 C o -3 a 157 F.

lejOS

Antes de entrar en detalle con la descripción de las funcionalidades del sistema operativo lejOS es necesario presentar la arquitectura de software del RCX estándar.

Veamos primero cuál es el modelo lógico sobre el que se basa el comportamiento del RCX. Este modelo se presenta como una serie de capas donde la capa inferior es el hardware conteniendo un procesador Hitachi H8300. Este procesador ejecuta instrucciones en código de máquina. También existen algunos componentes adicionales que tienen por tarea la de convertir señales de entrada desde tres puertos de entrada.

La siguiente capa es la que contiene la memoria ROM. Esta capa tiene un conjunto de instrucciones que provee todas las funcionalidades básicas de la unidad RCX tal como control de puertos, display, y comunicaciones por medio del puerto Infra rojo (IR). Esta memoria ROM puede ser comparada con el BIOS de una computadora personal que arranca el sistema y se comunica con sus

periféricos. Sin éste comportamiento de bajo nivel el RCX no podría comunicarse con el mundo externo.

A continuación de la capa anterior, se ubica el firmware. La palabra "firmware" identifica una clase de software que normalmente no es alterable por el usuario final aunque en este caso existe una importante excepción como se verá más adelante. La tarea del firmware es la de interpretar bytecodes y convertirlos en instrucciones de código de máquina, llamando a las rutinas de la ROM para realizar operaciones estándares del sistema. El firmware es almacenado en memoria RAM y se carga junto con la primera instalación del LEGO MINDSTORMS por medio del puerto IR.

Por encima del firmware están las clases base de lejOS y el programa de usuario. Este programa contiene el código y los datos de la aplicación desarrollados por medio del lenguaje RCX estándar. Este lenguaje se programa en la PC y convierte el código RCX en un formato más compacto e interpretable por el procesador del RCX (estos son los bytecodes).

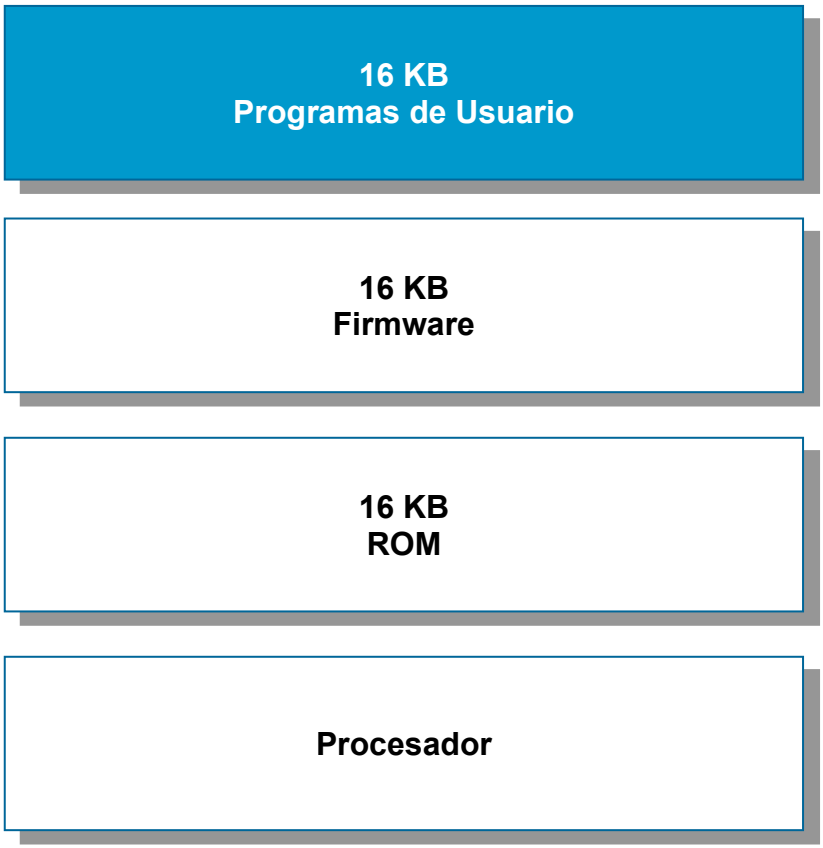

Figura 3 - Arquitectura lógica del RCX

La memoria RAM tiene una capacidad de 32Kbytes y está dividida lógicamente de la siguiente forma: 16 Kbytes para el firmware, 16 Kbytes para el almacenamiento de programas de usuario y el resto para la interpretación de los bytecodes y la ejecución del programa. Cuando el RCX se apaga, la memoria RAM permanece conectada a la fuente de energía con lo cual, tanto

el firmware como los programas de usuario permanecen cargados en la unidad RCX.

El proceso de programación del RCX se puede resumir en los siguientes pasos:

- En la PC se programa en código RCX por medio de un lenguaje visual.
- Las instrucciones del código RCX son transformadas en instrucciones de bajo nivel (bytecodes).
- El código en formato bytecodes es cargado en la memoria RAM de la unidad RCX por medio del puerto IR.
- El firmware interpreta los bytecodes y los convierte en código de máquina utilizando las rutinas de la ROM.
- El procesador ejecuta el código de máquina.

Como se puede observar, el proceso de programación sigue el modelo de capas descrito; partiendo desde el código del usuario final hasta el código de más bajo nivel de la ROM.

Expansión del RCX

Existen básicamente dos alternativas para expandir el RCX: la primera de ellas es la utilización de un ambiente de programación alternativo que tenga mejor performance y mayor flexibilidad. La segunda presenta un enfoque más radical que implica el reemplazo del firmware original, partiendo de la ventaja de que está almacenado en la memoria RAM y que puede ser accedido y modificado.

Ambiente de programación alternativo

El código RCX si bien disfruta de un buen grado de poder y control es una herramienta algo limitada.

Un esfuerzo muy grande fue puesto en superar estas limitaciones por parte de la comunidad internacional de robótica de LEGO. Luego de la aparición de este producto, se presentaron los resultados de un trabajo de ingeniería reversa que revelaba detalles del hardware RCX y que revelaba la lista de bytecodes del intérprete. A partir de este momento, comienzan a surgir nuevos lenguajes de programación que utilizan el firmware original pero con las siguientes características:

- El enfoque consiste en la realización de una nueva interfase a los bytecodes
- Por lo general, son lenguajes de texto en lugar de visuales como el código RCX
- Están orientados a sintaxis de lenguajes de programación populares como C o Java.

- Persiguen la filosofía del concepto freeware por medio de licencias públicas.

Entre algunos de estos lenguajes se encuentran NQC (Not Quite C) desarrollado por Dave Baum y Java API desarrollado por Darío Laverde por ejemplo. Otra alternativa sería Robolab que fue desarrollado por LEGO, National Instruments y la Universidad de Tufts.

Reemplazo del firmware original

Una de las formas de superar todas las desventajas del código RCX surge a partir de la posibilidad de que el firmware es almacenado en RAM. Entre algunas de las desventajas del firmware original se encuentran la imposibilidad de trabajar con números de punto flotante, un límite de 32 variables en un programa y la imposibilidad de manejar estructuras del tipo array. Como consecuencia, algunos lenguajes reemplazan por completo el firmware estándar. Existen muchas ventajas a partir de este enfoque como por ejemplo la posibilidad de superar el límite de 16 Kbytes de espacio para programas y datos como así también obtener mejoras en la performance.

Entre algunos de estos lenguajes se encuentran legOS desarrollado por Markus Noga, pbForth desarrollado por Ralph Hempel y lejOS una evolución del proyecto Tiny VM de José Solorzano.

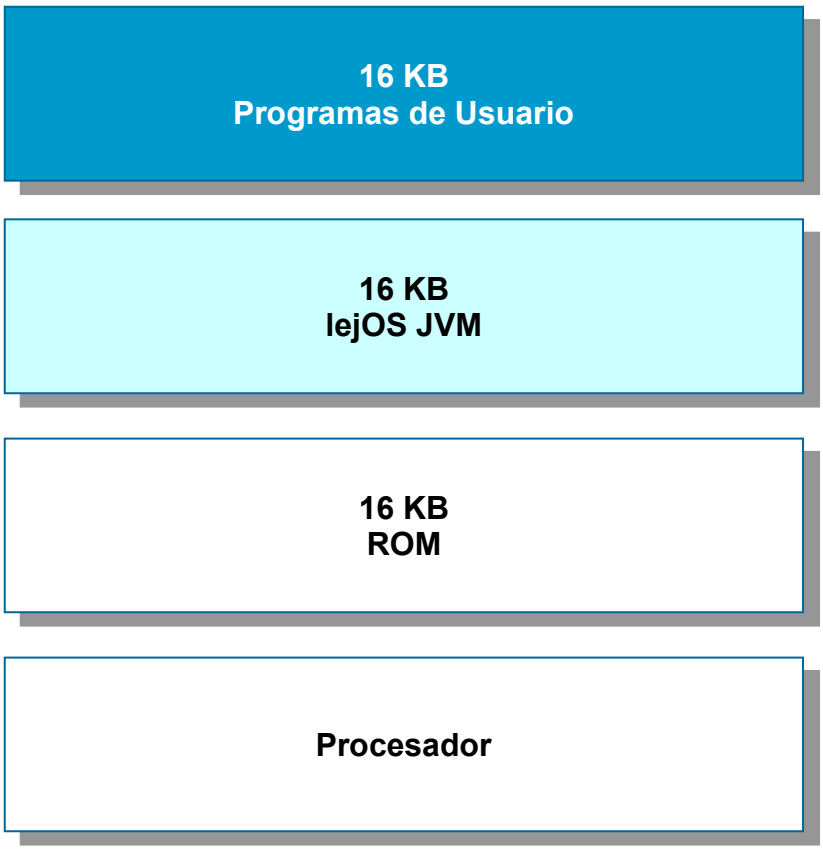

Figura 4 - Arquitectura lógica del RCX con lejOS

A continuación veremos en detalle las características principales de este lenguaje.

LejOS en detalle

lejOS puede ser comparado con el Java 2 SDK creado por Sun Microsystems en los siguientes aspectos:

- Ambos están disponibles para bajar libremente por Internet.
- Ambos utilizan compiladores que transforman el código fuente en instrucciones de bytecodes (de hecho lejOS utiliza el compilador de Sun *javac* para producir sus bytecodes.
- Ambos incluyen una máquina virtual java (JVM) que es un pequeño intérprete que lee los bytecodes y ejecuta instrucciones en la CPU.
- Ambos contienen una interfase de programación de aplicaciones (API) que es un conjunto de clases que pueden ser utilizadas por los programadores para obtener funcionalidad avanzada.

Sin embargo, existen algunas diferencias entre el Java SDK 2 y lejOS:

- Desde el punto de vista de desarrollo de producto, Java 2 SDK fue desarrollado por una compañía multimillonaria, mientras que lejOS fue desarrollado por un grupo de desarrolladores en Internet a ningún costo.
- Java 2 SDK viene en tres presentaciones: dos para computadores con recursos abundantes y uno para dispositivos pequeños como pagers, celulares y handhelds con memoria disponible entre 128 y 512 KB. Por su parte, lejOS fue pensado para correr en un dispositivo (el RCX) con sólo 32KB de memoria.

lejOS como cualquier lenguaje de programación controla el flujo del programa. La API de lejOS refleja la funcionalidad del Código RCX, permitiendo el control de motores, sensores, y otros elementos del RCX. LejOS también incluye una versión reducida de la API estándar de Java que contiene clases que soportan la programación básica como la clase vector. A continuación examinaremos con más detalle las funcionalidades de lejOS.

JVM

La JVM es análoga al firmware estándar de Lego; de hecho, la JVM es el firmware. Es el componente que debe ser cargado en el RCX para acomodar los programas Java. La JVM conoce como interpretar el bytecode que es cargado en el RCX y que se comunica con el firmware onboard (ROM) para hacer que el RCX haga lo que uno desea.

Memoria

Como se mencionara anteriormente, el RCX tiene un total de 32 KBytes de memoria RAM de los cuales 4 KBytes están fuera de los límites por que son utilizados por las rutinas ROM. Esto deja cerca de 28 KBytes libres para ser explotados. lejOS ocupa un espacio de 16 KBytes lo que implica que existen 12 KBytes de memoria libre para código de usuario.

Números en punto flotante

lejOS es el único lenguaje de RCX que permite números en punto flotante. Esto le da la capacidad de representar números fraccionarios como así también funciones trigonométricas como tangente, coseno y seno. Debido a la necesidad de optimizar memoria, lejOS no soporta números de 64 bits.

Threads

El multithreading permite que diferentes partes de un código se ejecuten aproximadamente al mismo tiempo, compartiendo la CPU. El esquema de threading de lejOS es muy completo permitiendo los mecanismos de sincronización e interrupción. Debido a que el RCX es monoprocesador, se utiliza un esquema preemptivo para permitir el multitheading.

En robótica, los threads son normalmente utilizados para monitorear un sensor. Debido a que existen únicamente tres entradas para sensores un programa sencillo necesitará usar 3 a 5 threads, sin embargo el límite impuesto por lejOS es de 255 threads.

Arrays

lejOS permite el manejo de arrays incluyendo arrays multidimensionales. Debido a que dependiendo del tipo de dato los arrays ocupan mucho espacio, existe un límite de 512 elementos para un array.

Modelo de eventos

lejOs es capaz de utilizar el modelo de eventos de Java, el cual incluye listeners y fuentes de eventos. En la API estándar de lejOS existen sólo tres objetos que utilizan listeners: Timer, Button y Sensor.

Excepciones

Java utiliza un esquema de manejo de errores para su tratamiento en el momento que estos se producen. La ventaja del manejo de excepciones es que permite que el código de chequeo de errores esté separado de la lógica del programa. Esta funcionalidad fue incluída en lejOS.

Recursividad

La técnica de recursividad permite que un método se llame a sí mismo. lejOS está limitado en el número de veces que un bloque de código puede estar anidado por la recursividad. Actualmente el límite en lejOS es de diez niveles de profundidad.

Garbage Collection

Los lenguajes orientados a objetos están basados en la creación de nuevos objetos. Cuándo un objeto no es utilizado más en un programa, el mismo debería ser removido de memoria por medio de un proceso llamado garbage collection. La mayoría de las implementaciones de Java poseen este mecanismo aunque lejOS actualmente no lo soporta. Esto requerirá que el programador tome todos los recaudos necesarios para optimizar la utilización de variables en el código lejOS.

Extensiones para la plataforma RCX

La API de lejOS contiene el paquete *josx.platform.rcx*. El prefijo *josx* en el paquete quiere significar Java Operating System extensión, siendo estas las extensiones necesarias a los paquetes estándar de Java para acceder y controlar el RCX. A continuación se presentará con mayor nivel de detalle este paquete pero sin entrar en el detalle de nombres de clases y métodos de la API.

Output

Existen tres puertos de salida (A, B y C) que pueden ser controlados por medio de lejOS. Los motores pueden ser movidos hacia delante o hacia detrás. Existen funcionalidades para hacer la reversa como para también detenerlos de distintas formas. El nivel de energía del motor también puede ser controlado.

Input

La lectura de datos desde los puertos de entrada es un poco más compleja que el control de puertos de salida. Existen dos seteos que deben ser especificados para cada sensor: tipo y modo. El tipo identifica la clase de sensor: luz, tacto, rotación y temperatura. Este seteo se debe a la necesidad de asociar a cada puerto el algoritmo adecuado para leer sus datos. El modo especifica que tipo de colección de datos realizará el sensor. Existen básicamente ocho modos a utilizar para este seteo.

RCX Buttons

Todos los botones excepto el de prendido/apagado pueden ser reprogramados por medio de lejOS. El modelo de eventos puede ser utilizado para capturar las presiones del botón y reaccionar acordemente cuando es activado. Esto

permite separar el manejo de la interfase de usuario del código principal con un estilo orientado a objetos.

System Time

El tiempo se mantiene dentro del RCX como un número en milisegundos desde que fue encendido. Esto es útil para registrar tiempos al momento que se suceden determinados eventos.

Battery Power

lejOs es capaz de chequear el estado de las baterías.

Programas Múltiples

lejOS permite hasta ocho programas almacenados simultáneamente. Una vez cargados, estos programas pueden ser ejecutados por medio del botón View.

LCD

Este sistema operativo puede controlar todas las funciones asociadas al LCD, pudiéndolo utilizar para presentar números y letras. Normalmente el LCD es utilizado para depurar los programas.

Speaker

Se puede hacer que el RCX toque varios sonidos predefinidos por medio del speaker. lejOS permite un control completo por medio de la frecuencia y duración de las notas. Estos sonidos pueden ser incorporados dentro de los programas como así también para la depuración de programas.

Comunicaciones por Infrarrojos

Las comunicaciones se pueden presentar en tres escenarios distintos:

- Entre el RCX y la torre de infrarrojos
- Entre dos RCXs
- Entre el RCX y el control remoto LEGO

La comunicación es muy sencilla y consiste en el envío y recepción de valores del tipo byte. La clase de comunicaciones puede también cambiar el rango de alcance de la torre permitiendo estar en contacto con el RCX a grandes distancias.

Timers

El RCX tiene un timer interno para seguir la pista de eventos. La diferencia entre el timer y el system time es que el timer permitirá la notificación a

todos los listeners luego de un delay por medio del modelo de eventos. El timer puede ser accedido por medio de la clase *josx.util.Timer*.

API de Java

La versión de lejOS de la API de Java puede ser pensada como una versión reducida de la API de Java original. Siempre que fue posible se mantuvo nombres de métodos y tipos de retorno idénticos a los originales de la API de Java. Actualmente existen sólo tres paquetes en lejOS que reflejan la funcionalidad de la API estándar de Java. Estos paquetes son:

- Java.lang
- Java.util
- Java.io

Programación de robótica

Debido a que el RCX es mayormente utilizado para programación de robótica tiene sentido incluir clases y métodos genéricos que aceleren el tiempo de desarrollo y aumenten la compatibilidad de programas. Estas clases fueron creadas sabiendo que los robots tienen a menudo distintos tamaños de ruedas, radios de giro e incluso velocidades de motor. Estos tipos de clases están almacenados en el paquete *josx.robotics*.

Clases de control de comportamiento

Uno de los temas más populares en programación de robótica es la programación de control de comportamiento. Este tipo de programación fue iniciado por Rodney Brooks del MIT y posteriormente refinada para la utilización con robots simples. lejOS lo incorporó con una serie de clases e interfases para hacer que el diseño de la arquitectura de control de comportamiento sea más sencilla y rápida.

Clases de navegación

El paquete de navegación provee clases que aceleran el desarrollo cuándo es necesario tareas de navegación en robots. Estas clases permiten el control del movimiento de manera tal que se le pueda indicar al robot para que apunte hacia cierta dirección, se mueva por una cierta distancia o incluso a una coordenada específica. Estas clases incluso mantienen la pista de las coordenadas del robot relativas a su punto de partida, por lo tanto manteniendo continuamente la información de dónde está ubicado el robot.

Redes Neuronales

Introducción

Como una de las diferentes tecnologías utilizadas en Inteligencia Artificial (IA), las redes neuronales han surgido como una poderosa herramienta para

resolver una gran variedad de problemas. Sus características de procesamiento paralelo y representación de conocimiento único, hicieron que fueran capaces de manejar gran cantidad de datos no lineales, multidimensionales y complejos.

Hoy día, están siendo utilizadas en aplicaciones de negocios tales como pronósticos financieros y detección de fraudes, como así también en el área de ingeniería, diseño y manufactura.

Arquitectura básica

En esta sección presentaremos algunos de los conceptos principales en el campo de las redes neuronales y definiremos un gran número de términos comúnmente utilizados. Aunque las definiciones dadas más adelante se relacionan con el tipo de redes neuronales denominadas "multi layer perceptron", también se aplican en otras arquitecturas de redes neuronales.

Una red neuronal consiste de un conjunto de unidades de procesamiento simples que se comunican enviándose señales entre sí a través de un gran número de conexiones a las cuales se le asignan pesos. Muchas de las características propias del modelo de procesamiento distribuido pueden ser distinguidas en las redes neuronales:

- Un conjunto de unidades de procesamiento denominados nodos o neuronas.
- Un estado de activación y_k para cada unidad que es equivalente a la salida de la unidad.
- Conexiones entre las unidades. Generalmente cada conexión esta definida por su peso w_{jk} que determina el efecto que tiene la señal de la unidad j sobre la unidad k.
- Una regla de propagación, que determina la entrada efectiva s_k de una unidad de sus entradas externas.
- Una función de activación F_k, que determina el nuevo nivel de activación basada en la entrada efectiva $s_k(t)$ y la activación actual $y_k(t)$.
- Una entrada externa, bias, θ_k para cada unidad.
- Un método para recolectar información (la regla de aprendizaje).

Dentro de un sistema neuronal es útil distinguir tres tipos de unidades o nodos, las unidades de entrada que reciben datos del exterior de la red neuronal, las unidades de salida que envían datos fuera de la red y las unidades ocultas o internas cuyas señales de entrada y salida permanecen dentro de la red.

Nodos

Una red neuronal esta compuesta por un gran número de elementos de procesamiento interconectados denominados nodos. Cada nodo tiene un número de entradas y asociado a cada entrada existe un número, denominado

peso. Conceptualmente, los nodos son ordenados en capas donde pueden operar en paralelo.

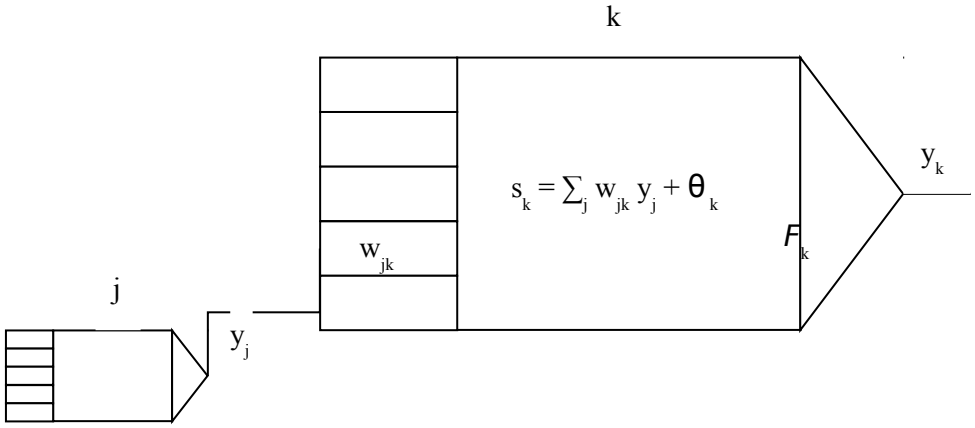

Figura 5 - Componentes básicos de una red neuronal

El cálculo computacional realizado por el nodo consiste en multiplicar cada una de sus entradas por su peso asociado y luego sumar los resultados para producir un único valor, como se muestra en la figura 5. Generalmente, este valor es procesado por una función de activación o transformación para determinar la salida de la unidad. La salida de la unidad puede luego alimentar a las entradas de otras unidades en la red neuronal o utilizada como una salida de la red.

Además de este procesamiento, una segunda tarea es el ajuste de los pesos. El sistema es inherentemente paralelo en el sentido de que muchas unidades pueden llevar a cabo su procesamiento al mismo tiempo. Durante la operación, las unidades pueden actualizarse sincrónicamente o asincrónicamente. Con la actualización sincrónica, todas las unidades actualizan su activación simultáneamente, con la actualización asincrónica cada unidad tiene una probabilidad de actualizar su activación en el tiempo t y generalmente solo una unidad podrá realizarlo.

Conexiones entre unidades

Los pesos asociados a cada entrada, junto con la función de activación, transforman la entrada de la red neuronal en una salida. El algoritmo particular utilizado para calcular estos pesos varía ampliamente según el tipo de red neuronal. Debido a que es difícil calcular los pesos directamente, esto es realizado por un proceso repetitivo denominado entrenamiento de la red neuronal.

En la mayoría de los casos asumimos que cada unidad provee una contribución aditiva a la entrada de la unidad a la cual esta conectada. El total de la entrada de la unidad k es simplemente la suma pesada de las salidas de cada una de las unidades conectadas a esta unidad mas un desvio o bias denominado θ_k.

$$s_k = \sum_j w_{jk}(t)\, y_j(t) \quad + \quad \theta_k(t)$$

Figura 6 – Regla de propagación

La contribución de valores positivos de w_{jk} se considera como una excitación y cuando se trata de valores negativos como una inhibición. En algunos casos, se aplican reglas más complejas para combinar las entradas, en las que se hacen una distinción entre entradas excitadas e inhibidas. Se denominan unidades sigma a las unidades con regla de propagación como la de la figura 6.

Funciones de activación

La función de activación, F_k transforma la entrada de un nodo s_k en un valor de salida. Generalmente la función de activación es una función no decreciente sobre la entrada total.

$$y_k(t+1) = F_k(s_k(t)) = F_k(\sum_j w_{jk}(t)\, y_j(t) + \theta_k(t))$$

Figura 7 – Función de activación

Un ejemplo simple de una función de activación es la función umbral. Cuando el total de la entrada de un nodo excede el valor umbral, el nodo produce una determinada salida. En la práctica, las funciones de activación de este tipo son poco utilizadas. En cambio, la mayoría de las redes neuronales utilizan funciones de activación no lineales, que le aportan a la red más flexibilidad. Con funciones no lineales, la red neuronal puede implementar transformaciones más complejas, permitiéndole resolver una amplia variedad de problemas. Por ejemplo, una función del estilo sigmoide como la que se muestra en la figura 8 es generalmente utilizada:

$$y_k = F_k(s_k) = 1 \,/\, (1 + e^{-sk})$$

Figura 8 – Función de activación sigmoide

Estructura de la red neuronal

Existen diferentes arquitecturas de redes neuronales, que consideran básicamente las conexiones entre los nodos y la propagación de los datos podemos distinguir dos tipos de redes:

- **Redes feed-forward:** Como se describe en la figura 9, en esta red, los nodos están organizados en capas conectados de manera tal que el flujo de datos va estrictamente desde la entrada hacia la salida. Cada nodo en la primera capa tiene una única entrada. La salida de las unidades de la primera capa alimenta a todas las unidades de la capa intermedia, cuya salida alimentan a la última capa. Las salidas de las unidades en la capa de salida forman la salida de la red neuronal. Ejemplos clásicos de estas redes son: Perceptron propuesta por Ronsenblatt y Adaline propuesta por Widrow y Hoff.

- **Redes recurrentes:** Estas redes contienen conexiones que retroalimentan a la red. Contrariamente a las redes feed-forward, las propiedades dinámicas de la red son importantes. En algunos casos, los valores de activación de las unidades experimentan un proceso de relajación de manera tal que la red evoluciona hacia un estado estable, en el cual estos valores de activación no cambian más. Ejemplos clásicos de estas redes son: redes de Anderson, redes de Kohonen y redes de Hopfield.

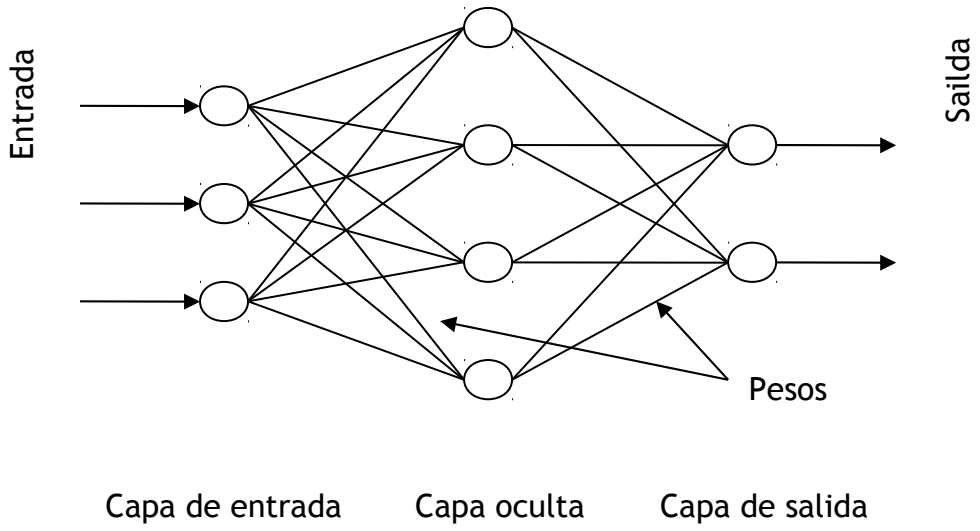

Figura 9 – Topología de una red neuronal multi-layer perceptron

Entrenamiento de la red neuronal

A diferencia de las técnicas tradicionales para el procesamiento de datos, que requieren una programación compleja, las redes neuronales desarrollan

soluciones a los problemas sin la necesidad de instrucciones explícitas. En efecto, las redes neuronales son entrenadas más que programadas.

Durante la fase de entrenamiento, a las redes neuronales se les presentan datos ejemplos con sus respectivos pesos hasta que la salida de la red es obtenida. El método para ajustar los pesos es conocido como algoritmo de entrenamiento o aprendizaje. Hay un sinnúmero de algoritmos, muchos de los cuales son específicos del tipo de la red neuronal.

Los algoritmos de aprendizaje están divididos en dos clases:

- **Aprendizaje supervisado:** Con estos algoritmos, la red neuronal aprende ajustando sus pesos de manera tal que la salida coincide con un objetivo en particular. Las redes neuronales que utilizan el aprendizaje supervisado confían en un conjunto de salidas "objetivo" para deducir los valores de los pesos de las conexiones. Como primer paso se realiza una comparación entre la salida producida por la red y el valor deseado. El algoritmo luego ajusta los pesos para reducir las diferencias entre la salida y el objetivo. Este tipo de aprendizaje se utiliza en sistemas de reconocimiento de patrones, donde la red es entrenada con un conjunto de patrones que han sido identificados de antemano. Las etiquetas que identifican a los patrones son utilizadas como la salida objetivo que la red utiliza para evaluar su "aprendizaje" de los datos. La red en efecto aprende mediante ejemplos.

- **Aprendizaje no supervisado:** A las redes que utilizan estos algoritmos, no se les suministra un objetivo. En cambio, aprenden identificando patrones en los datos de muestreo. A diferencia del aprendizaje supervisado, las redes que utilizan aprendizaje no supervisado determinan sus pesos independientemente de cualquier criterio de perfomance. En otras palabras, estas redes utilizan algoritmos que manipulan los datos de entrada y ajustan sus pesos, pero no realizan conclusiones acerca de que dato debe indicar. El aprendizaje no supervisado es útil para buscar patrones que pueden existir en datos complejos. Bajo este paradigma, el sistema descubre estadísticamente características salientes de la población de entrada.

Una vez entrenada, los pesos de la red neuronal son ajustados y la red puede ser utilizada para predecir la solución, dado un conjunto de datos aun no conocido por la red. Esta fase se denomina etapa de clasificación.

Un punto importante a tener en cuenta cuando se utiliza una red neuronal es distinguir dos características importantes que influyen en la performance del sistema.

- **Poder representativo:** Se refiere a la habilidad de la red neuronal de poder representar la función deseada. Debido a que la red esta construida sobre un conjunto de funciones estándar, en la mayoría de

los casos la red solo aproximará al valor deseado y aún para un conjunto de pesos óptimo, el error de aproximación nunca será cero.

- **Algoritmo de procesamiento:** Dado que existe un conjunto de pesos óptimos en la red, debemos hacernos la siguiente pregunta ¿Existe un procedimiento iterativo finito que encuentre este conjunto adecuado de pesos?

Por el teorema de la convergencia, podemos afirmar que en el algoritmo de aprendizaje que hemos utilizado en nuestro trabajo, existe un número finito de pasos después de los cuales se encontrará una solución, es decir el valor óptimo de los pesos de las conexiones independiente del valor inicial de los mismos.

Procesamiento de datos

Antes que los datos sean suministrados a la red, los mismos deben convertidos a un "formato" adecuado mediante algún tipo de preprocesamiento. Los datos que son utilizados como entrada de la red neuronal, se denominan vector de datos o patrón de datos de entrada. Otro de los objetivos del preprocesamiento es seleccionar los datos mas relevantes, de manera tal que el número de entradas de la red no sea más grande que lo necesario. Esto da como resultado una red simple con bajo tiempo de procesamiento.

La transformación de los datos puede realizarse de distintas maneras, dependiendo del tipo de datos y de la aplicación. Algunos ejemplos son:

- Agrupamiento de los datos continuos en rangos discretos.
- Redimensionamiento de los datos para dar una resolución uniforme (imágenes)
- Adaptar la curva de la función a un conjunto de datos para permitir el cálculo sobre un número de puntos deseados.

La selección de datos relevantes puede realizarse de diferentes maneras:

- Utilizar un filtro para limpiar los ruidos de los datos.
- Agrupar tipos de datos relacionados para constituir clases de datos.
- Utilizar medidas estadísticas para resumir los datos numéricos.

El postprocesamiento de los datos involucra la transformación de los datos de salida de la red neuronal en un formato más fácil de comprender por el usuario. Esto se lleva a cabo asignando etiquetas o identificadores a los datos de salida, para otorgarle una interpretación mas natural a los resultados.

Generalización

La habilidad de generalizar, representa una de las mayores ventajas de las redes neuronales sobre los programas convencionales. La generalización es la

habilidad de la red neuronal para encontrar la respuesta correcta o clasificación para un conjunto de datos que aun no han sido presentados a la red. Por ejemplo, una red neuronal diseñada para distinguir fotografías de caras masculinas y femeninas no puede ser entrenada con imágenes de cada mujer y hombre que exista en el planeta. A pesar de esto, la red es capaz de clasificar correctamente basándose en generalizaciones hechas durante el entrenamiento. Es importante sin embargo que la red no sobre generalice, esto es, que desdibuje distinciones importantes entre los tipos de datos.

Redes back-propagation (BPN)

Las redes multi-layer perceptron, reavivaron el interés sobre las redes neuronales a pesar del documento presentado por Minsky y Papert en 1969, donde demuestran que los perceptrones tienen muchas restricciones con respecto a lo que pueden llegar a representar, por ejemplo, no pueden representar la función o-exclusiva o xor (Minsky and Papert's Perceptrons). Introducida por Rumelhart a mediados de 1980, estas redes neuronales ofrecen una solución a los problemas de clasificación de datos que no son linealmente separables.

El término back-propagation describe el tipo de algoritmo de aprendizaje supervisado que utilizan las redes multi-layer perceptron. Además de las capas de entrada y salida, estas redes contienen una o más capas ocultas de nodos, totalmente interconectadas. El número apropiado de capas ocultas dependen del tipo y la complejidad del problema de clasificación a resolver, aunque algunas investigaciones han determinado que un máximo de cuatro capas es suficiente. Los valores de entrada son pasados desde las capas de entrada a través de las conexiones hasta la primera capa oculta. La función de activación de los nodos de la capa oculta es una sigmoide pero también pueden utilizarse otras funciones como por ejemplo la función tangente hiperbólica. En la capa de salida, los valores de error de la red son calculados basados en los valores actuales de los pesos y las entradas. Este error es luego propagado hacia atrás en la red reduciendo o aumentando los pesos para poder reducir el valor del error. Este ciclo de cálculo hacia adelante y luego hacia atrás es repetido hasta que el error tienda a un mínimo, después del cual la fase de entrenamiento finaliza. Por el teorema de la convergencia, podemos afirmar que existe un número finito de pasos después de los cuales se encontrará una solución, es decir el valor óptimo de los pesos de las conexiones independiente del valor inicial de los mismos.

En una red que utiliza el algoritmo back-propagation, el error es proporcional a la suma de los cuadrados de la diferencia entre la salida deseada y la actual. El algoritmo denominado regla de gradiente descendente, es utilizado para calcular las magnitudes por las cuales los pesos de la red deberían ser ajustados para minimizar el error. Buscar los mejores valores para los parámetros de las ecuaciones de esta regla de aprendizaje es un proceso de "prueba y error" con el mínimo número de iteraciones.

Aunque las redes back-propagation pueden ser utilizadas con cualquier número de capas ocultas, se ha demostrado que para redes con unidades

binarias, una única capa es suficiente para aproximar cualquier función con muchas discontinuidades finitas con una precisión arbitraria, dado que las funciones de activación de las unidades de la capa oculta son no-lineales. En la mayoría de las redes feed-forward con una única capa oculta se utiliza una función de activación sigmode en cada una de sus unidades.

La aplicación típica de las redes back-propagation o feed-forward, es la clasificación de patrones. Estas redes se adaptan más fácilmente que otros tipos de redes para clasificar datos. También pueden ser utilizadas para predecir cuando se utilizan series de datos temporales en el vector de entrada.

Otra de las aplicaciones es la de aproximar una función por ejemplos. Una de las debilidades de estas redes es que clasifican arbitrariamente el espacio de entrada. Otra es que para grandes redes, el tiempo requerido para entrenamiento puede ser prohibitivamente largo.

Enfoque de la solución

En las próximas secciones describiremos cuál es el enfoque de nuestra solución al problema establecido en el capítulo de introducción de nuestra tesis. Entendemos que la mejor forma de describirla es primero detallar cuál es el enfoque funcional de la solución, o sea, “el que” vamos a implementar y a continuación describir cuál es el enfoque técnico de la misma, o sea, “el como” la implementaremos.

Enfoque funcional

Nuestra solución al problema de clasificación de patrones será modelada por medio de la utilización de un robot que será quien posea la inteligencia necesaria para llevar a cabo esta tarea. A continuación describiremos cuáles son los componentes principales de la misma.

Por un lado, tenemos una PC que tendrá conectados una webcam que será la encargada de reconocer y codificar los patrones a clasificar y una torre infrarrojos que transmitirá los patrones codificados al robot.

Por otro lado, tenemos el robot que estará construido con piezas de Lego y contendrá el RCX que será el cerebro del mismo y que permitirá recibir los patrones transmitidos por su lector infrarrojos, clasificarlos, y por medio de distintos sensores y motores conectados al mismo, accionar un mecanismo que permita depositar los patrones clasificados en diferentes cestos.

La inteligencia del robot estará distribuida entre la PC y el RCX. En la PC se encontrará la lógica necesaria para leer los patrones, codificarlos y transmitirlos por el puerto infrarrojo, mientras que en el RCX se encontrará la lógica necesaria para leer los patrones codificados por el infrarrojo, clasificarlos por medio de una red neuronal con aprendizaje por back-propagation y luego accionar sensores y motores del robot.

La utilización de la red neuronal necesitará de la ejecución de dos procesos:

Proceso 1: Entrenamiento de la red.
En este paso se alimenta la red neuronal con los patrones a reconocer, esto se hace por medio de un archivo plano que contiene la codificación binaria de los mismos. Los patrones definidos en este archivo son leídos cíclicamente hasta que el grado de error de aprendizaje de la red esté por debajo de un parámetro determinado, en nuestro caso será del 1 %. Una vez logrado este porcentaje, los pesos de las conexiones de la red se encuentran ajustados como para que la red pueda reconocer estos patrones. Este proceso se encuentra en la PC que es la que se encargará de calcular los pesos asociados a la red. A continuación presentamos el pseudo código de este proceso:

- **Paso 1 - Inicializar los pesos y los bias:** Asignar a los pesos y los bias de cada nodo con valores aleatorios pequeños.

- **Paso 2 - Presentar la entrada y la salida deseada:** Presentar un vector de valores $x_0, x_1, \ldots, x_{n-1}$ y especificar la salida deseada $d_0, d_1, \ldots, d_{m-1}$. Si la red es utilizada como una clasificadora, luego todas las salidas deseadas deberían ser igual a 0, excepto para la salida que corresponde a la clase de la entrada, a la cual se le asigna el valor 1. La entrada podría ser distinta en cada intento o la muestra para el conjunto de entrenamiento podría ser presentada cíclicamente hasta que los pesos se estabilicen.

- **Paso 3 - Calcular el valor de la salida:** Utilizar la siguiente fórmula para calcular la salida $O_1, O_2, \ldots, O_{m-1}$ para cada unidad en la red:

$$O_i = f(\Sigma x_i w_i + b_i)$$
$$f(y) = 1 / (1 + e^{-\lambda * y})$$

  donde x es el vector de entrada, w el vector de pesos, b el vector de bias y λ el parámetro de squashing. En nuestro caso utilizaremos un valor neutro, es decir 1.0.

- **Paso 4 - Adaptar los pesos:** Utilizar un algoritmo recursivo comenzando en la salida y volviendo hacia atrás hasta la primera capa oculta de la red. Esto puede contener varios pasos:

  - **Paso 4.a** – Calcular la suma de los cuadrados del error de la red.

$$\textbf{Error} = \tfrac{1}{2} \sum_{i \in \text{outputs}} (d_i - O_i)^2$$

- **Paso 4.b** – Calcular el error de cada neurona en la capa de salida.

$$\delta_i = O_i (1 - O_i) (d_i - O_i)$$

- **Paso 4.c** – Calcular el error de cada neurona en la capa oculta.

$$\delta_i = O_i (1 - O_i) \sum_j \delta_j w_{ij}$$

  donde j, es el índice de los nodos de la capa siguiente a la que las señales de i se conectan.

- **Paso 4.d** – Calcular los pesos delta. η es la tasa de aprendizaje. Un bajo valor de la tasa de aprendizaje puede asegurar una convergencia más estable. En cambio una alta tasa de aprendizaje puede acelerar el proceso.

$$\Delta w_{ki} = \eta \, \delta_k \, x_{ki}$$

  donde w_{ki} es el peso del nodo k en la capa oculta al nodo i.

- **Paso 4.e** – Sumar los pesos delta a cada uno de los pesos.

$$w_{ki}(t+1) = w_{ki}(t) + \Delta w_{ki}$$

  donde t denota el número de iteración.

- **Paso 5 - Volver al paso 2.**

Proceso 2: Puesta en ejecución de la red con un patrón.
Una vez entrenada la red por medio del proceso anterior, la misma ya puede ser utilizada para el reconocimiento de patrones. Este código se encuentra en el RCX quien es el que se encarga de ejecutar las acciones necesarias luego de clasificar el patrón.

Enfoque Técnico

A continuación presentaremos el enfoque técnico de nuestra solución. En primer término describiremos como fue diseñada la misma presentando cuál fue la arquitectura de software utilizada.

En segundo término describiremos como fueron implementados los distintos componentes de esta arquitectura de software.

Arquitectura Técnica

Desde el punto de vista de arquitectura técnica de nuestra solución, la misma puede ser vista como compuesta de tres capas principales.

En la capa identificada como hardware encontramos básicamente la PC y el RCX como los componentes principales.

En la capa identificada como kernel encontramos los componentes principales que dan servicio a la aplicación.

En la capa aplicación encontramos la aplicación propiamente dicha, es decir los componentes que permiten el aprendizaje y la clasificación de patrones.

En la siguiente figura puede observarse como se encuentran organizadas estas capas como así también, cuáles son los componentes principales dentro cada una de ellas. Las capas identificadas con verde son las implementadas para nuestro trabajo.

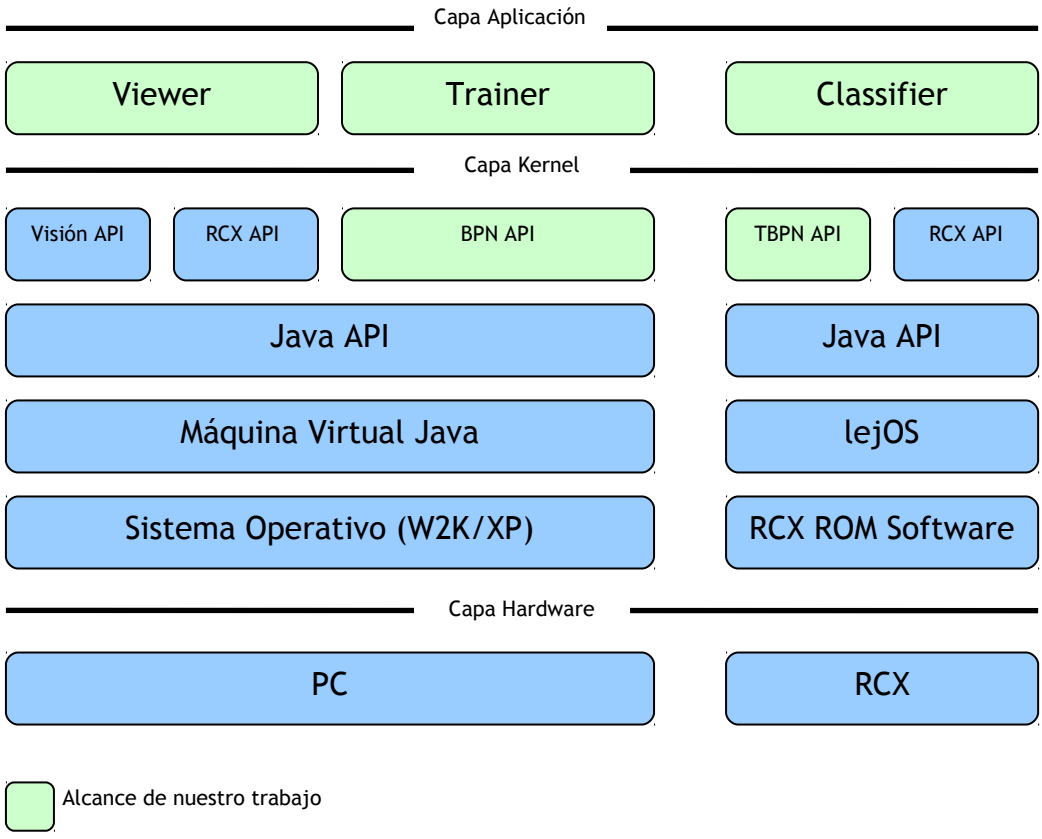

Figura 10 – Arquitectura de Técnica

Como se puede observar en el gráfico anterior la arquitectura considera los componentes tanto de la PC como del RCX.

Previamente a la descripción de esta arquitectura técnica, es importante aclarar dos decisiones de diseño que debieron ser tomadas para definir la misma. La primera de ellas tiene que ver con que la webcam siempre debe estar conectada a la PC, con lo que la lógica de visión debe residir también en ésta. La segunda tiene que ver con los recursos limitados del RCX. Debido a las importantes restricciones de memoria del mismo, se decidió implementar el proceso de aprendizaje, descrito en la sección anterior, en la PC y no en el RCX, dejando al RCX con la única responsabilidad de contener la red neuronal ya entrenada y la lógica de activación del robot.

A continuación veremos una breve descripción de las distintas capas de software de la arquitectura utilizada.

Comenzando de abajo hacia arriba, la primer capa que encontramos es la de hardware donde están la PC y al RCX.

A continuación de la capa de hardware encontramos la capa del kernel. Esta capa provee de servicio a la aplicación final y la encontramos en la PC como en el RCX. Primero veamos como se encuentra organizada la misma en la PC.

La primera capa que encontramos es la del sistema operativo, en este caso nosotros utilizamos Windows 2000, pero también se podría utilizar Windows XP, Windows 98 o inclusive Linux. Sobre esta capa se encuentra la Máquina Virtual Java cuya función es interpretar lenguaje java y traducirlo en el código de máquina de la plataforma en la que se ejecuta. En la próxima capa tenemos la Java API, que está compuesta por una serie de paquetes que proveen al desarrollador de las funcionalidades básicas del lenguaje java. Sobre esta capa se encuentran tres componentes: la API de Visión cuya funcionalidad es la de implementar una serie de métodos asociados con funciones de visión como captura de imágenes, interpretación de zonas, detección de movimientos, etc, la API del RCX que abstrae funcionalidades básicas del RCX, como las que se encargan de activar motores, leer el estado de los sensores y permitir la comunicación con otros componentes por medio de distintos protocolos propietarios y por último la BPN API que fuera desarrollada por nosotros y que modela el algoritmo de back-propagation.

Veamos ahora como está organizadas las capas del kernel pero en el RCX. En primer término se encuentra la capa del software del RCX residente en memoria ROM. A continuación se encuentra el sistema operativo lejOS. Luego la Java API y sobre ésta dos componentes: la RCX API y la TBPN API esta última desarrollada por nosotros y cuya función es la de utilizar la red neuronal entrenada para la clasificación de patrones.

Respecto a la última capa, la capa de aplicación, en la PC encontramos dos componentes desarrollados para este trabajo. En primer término el Viewer, cuya función es la de capturar por medio de la webcam los patrones y codificarlos y transmitirlos al RCX por medio del puerto infrarrojos. En segundo lugar encontramos el Trainer, cuya función es la de entrenar a la red neuronal para el reconocimiento de patrones. En el RCX encontramos el

componente Classifier cuya función es la de recibir los patrones por el infrarrojo, clasificarlo y activar el mecanismo del robot.

En la próxima sección veremos en mayor grado de detalle los aspectos de implementación de cada uno de los componentes desarrollados para este trabajo.

Implementación

A continuación presentamos un diagrama de clases general de nuestra solución. En el mismo aparecen los tres componentes principales que componen nuestra aplicación: el Viewer, el Trainer y el Classifier descriptos en la sección anterior. El siguiente diagrama muestra las clases relevantes involucradas y sus relaciones:

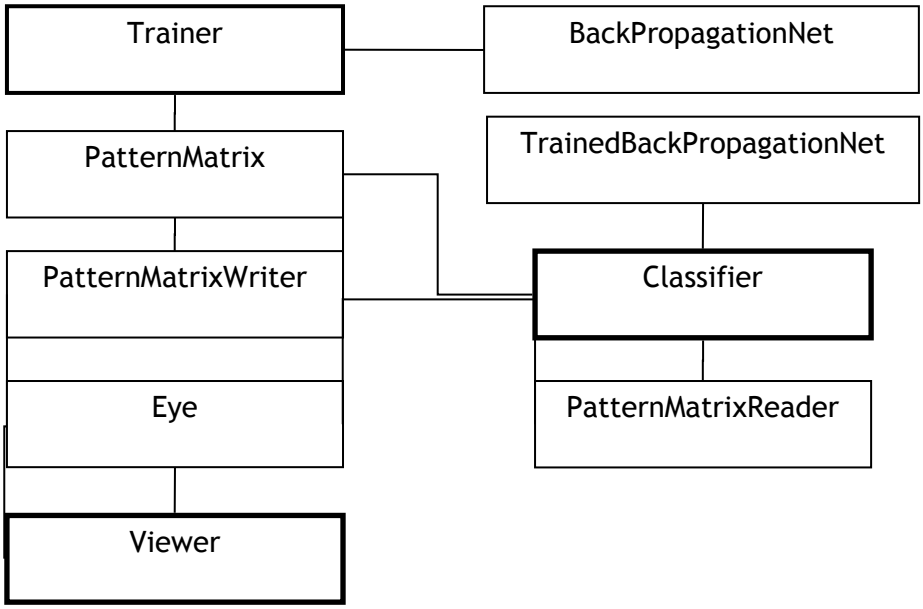

Figura 11 – Diagrama de clases

En la siguiente figura, el diagrama de implementación, mostramos los componentes de hardware y software de nuestra solución. En el podemos notar como están distribuidos y como se relacionan los módulos con el hardware. Como se puede observar, el programa de aplicación Viewer, se ejecuta en la PC, mientras que el programa Classifier se ejecuta en el RCX. La comunicación entre los mismos se realiza utilizando el enlace infrarrojo (IR).

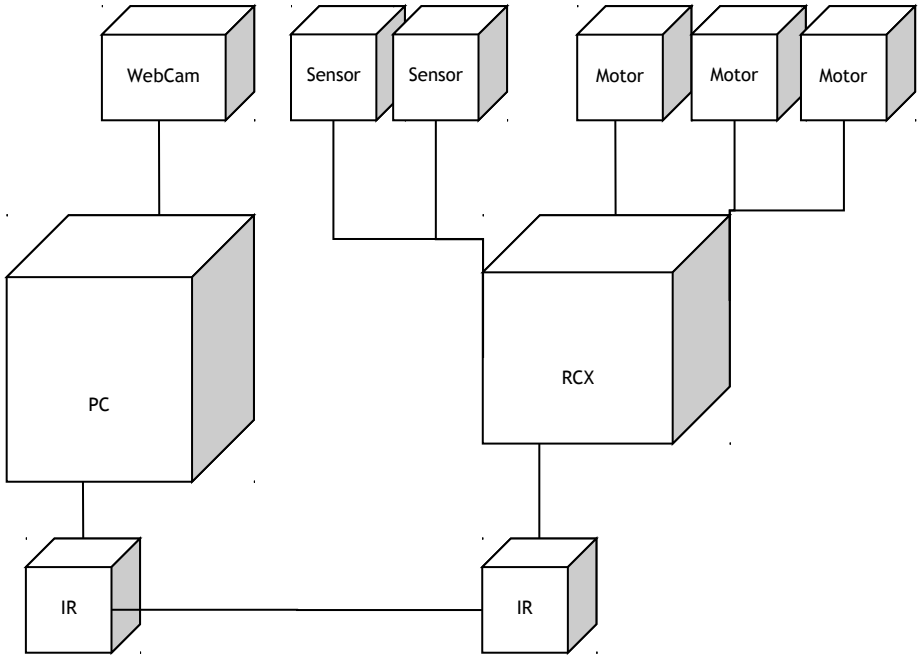

Figura 12 – Diagrama de implementación

PC

A continuación describiremos en detalle los componentes principales que se encuentran en la PC:

BPN API

Como mencionamos anteriormente, la solución que nos hemos propuesto desarrollar, consiste en implementar un conjunto de clases que nos permita reconocer patrones "de manera inteligente" dentro del entorno lejOS. En particular, hemos elegido modelar la red neuronal del tipo multi-layer perceptron que utiliza el algoritmo de aprendizaje back-propagation. El resultado de tal modelación dio origen a lo que denominamos BPN API, BackpropagationNet Application Program Interface.

Las clases que componen la BPN API son:

- **BackPropagationNet:** Esta clase modela una red neuronal del tipo multi-layer perceptron. La misma tiene como responsabilidad la de proveer los métodos y atributos necesarios que nos permita construir la infraestructura de la red neuronal (cantidad de niveles, cantidad de neuronas por nivel), asignar los valores de entrada y los deseados, asignar los parámetros del algoritmo de aprendizaje, como así también el método que nos permite ejecutar el mismo y dar inicio al proceso de aprendizaje.

- **WeightMatrix:** Esta clase da soporte a la clase BackPropagationNet y tiene como responsabilidad la de suministrar los métodos que le permite a la anterior asignar los pesos iniciales de las conexiones entre

las neuronas de niveles adyacentes, como así también la de calcular las magnitudes por las cuales los pesos de la red deberían ser ajustados para minimizar el error.

- **NeuronLayer:** Esta clase representa una colección de neuronas en un determinado nivel. Tiene como responsabilidad la de proveer los métodos para agrupar las neuronas en el nivel, asignar sus valores iniciales y calcular la salida de cada neurona que la compone.

- **Neuron:** Esta clase define la mínima unidad de procesamiento de una red neuronal. Sus métodos y atributos nos permite asignar sus valores iniciales, calcular la entrada, calcular la salida y el error cometido.

En el siguiente diagrama, mostramos las relaciones entre las clases mencionadas.

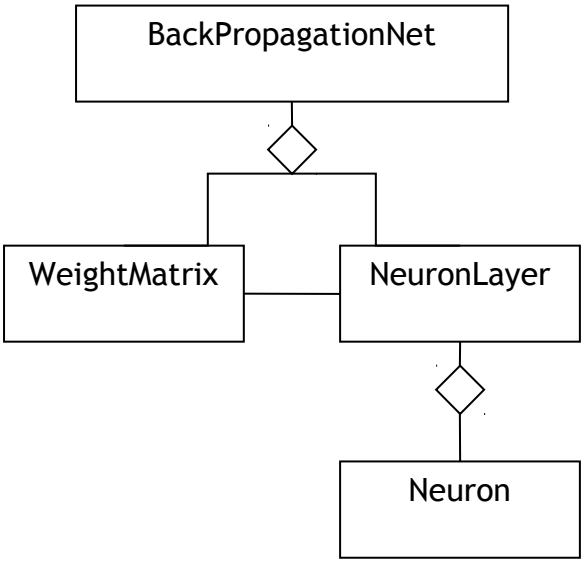

Figura 13 – Diagrama de clases BPN API

Trainer

El programa de aplicación que denominamos **Trainer**, utiliza la clase BackPropagationNet para entrenarla y generar dinámicamente la clase TrainedBackPropagationNet utilizada en el RCX. Para su entrenamiento, utiliza una lista de patrones, que lee de un archivo ASCII, cada uno de ellos representado por la clase PatternMatrix. El siguiente diagrama muestra las clases relevantes involucradas y sus relaciones:

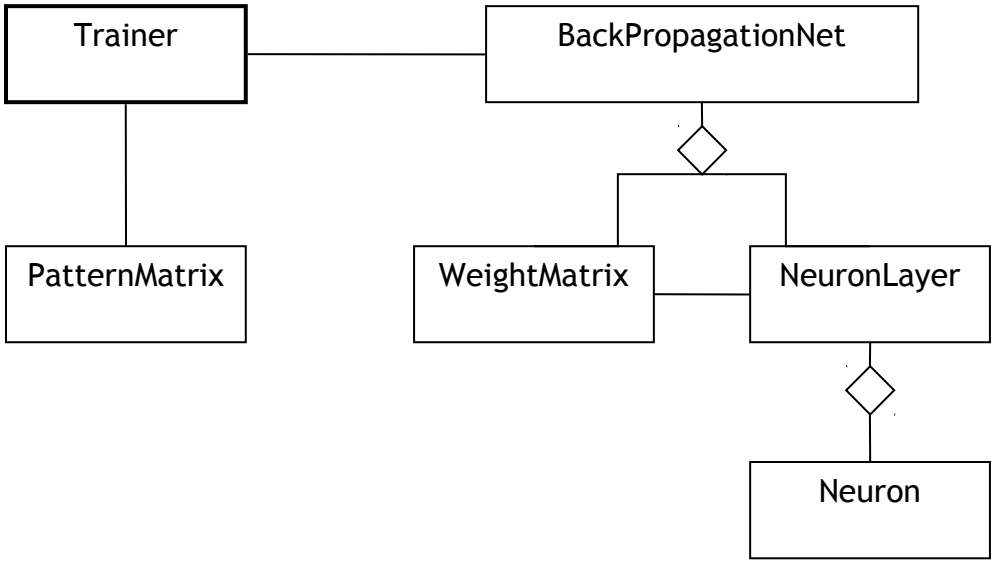

Figura 14 – Diagrama de clases del Trainer

En el siguiente diagrama de componentes, mostramos los módulos de software y sus relaciones del programa **Trainer**:

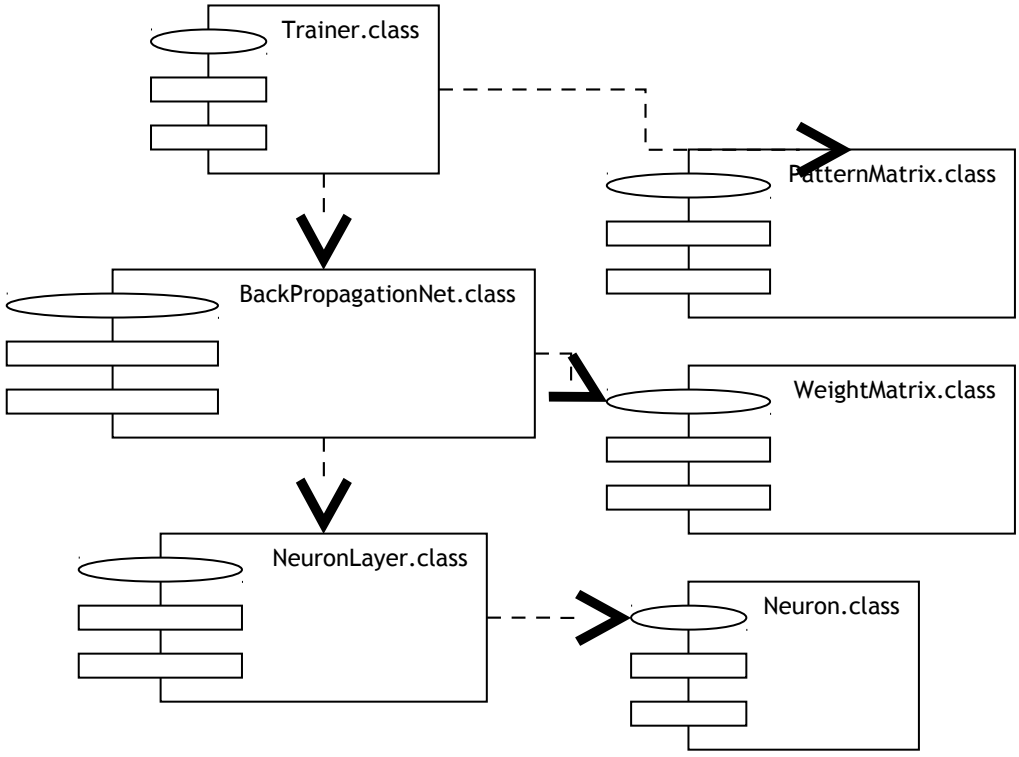

Figura 15 – Diagrama de componentes del Trainer

Viewer

El programa de aplicación que denominamos **Viewer**, utiliza las clases para la captura y manipulación de imágenes, toma una foto de lo que esta frente a la lente de la webcam, ante una petición del programa **Classifier**. Dicha petición es transmitida por el puerto infrarrojo. Al recibir la petición, el **Viewer** le solicita al objeto de la clase Eye, que tome una foto de la imagen que el mismo esta capturando en tiempo real. Este luego se encarga de codificarla y convertirla en un objeto de la clase PatternMatrix que es transmitido al **Classifier** utilizando nuevamente el enlance infrarrojo. Las clases que se encargan de transmitir y recibir objetos del tipo PatternMatrix son PatternMatrixReader y PatternMatrixWriter. El siguiente diagrama muestra las clases relevantes involucradas y sus relaciones:

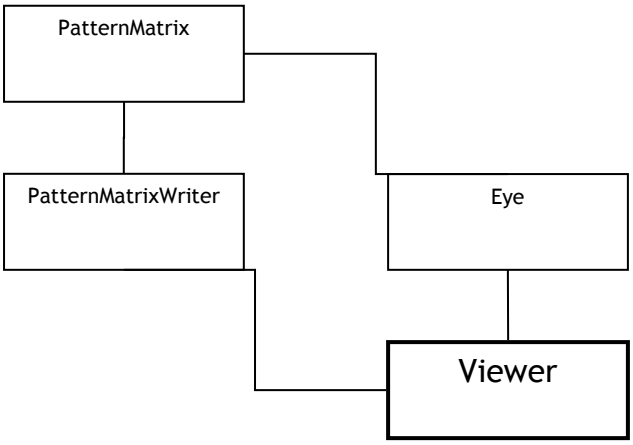

Figura 16 – Diagrama de clases del Viewer

En el siguiente diagrama, mostramos los módulos de software del programa de aplicación **Viewer**. En el mismo, se podrá observar los componentes más relevantes del programa de aplicación y la relación de dependencia entre los mismos:

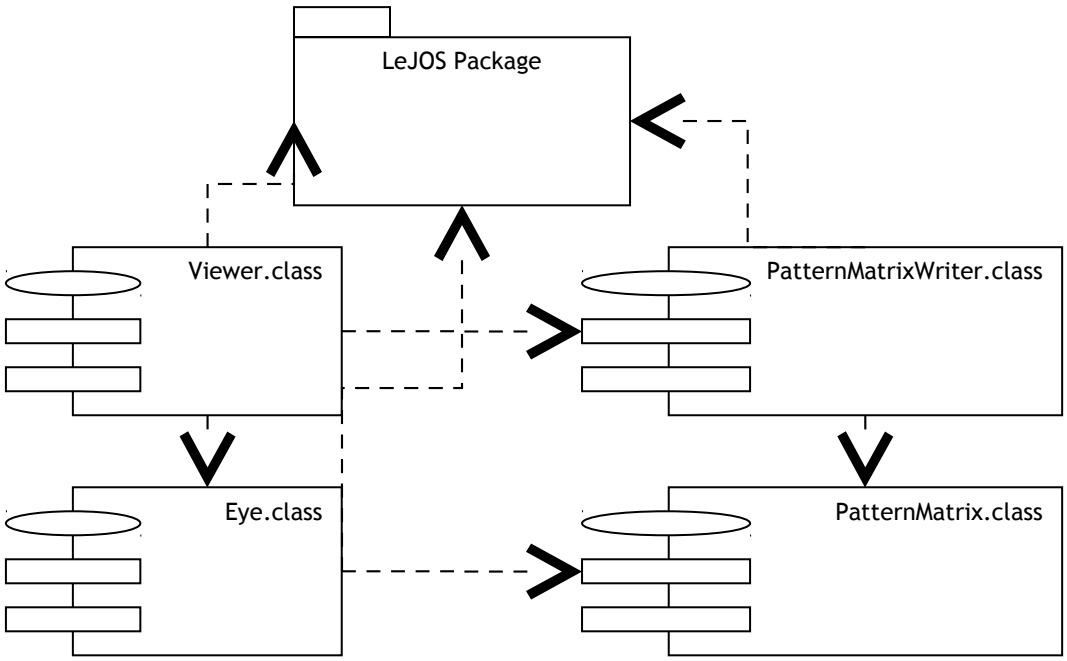

Figura 17 – Diagrama de componentes del Viewer

RCX

A continuación describiremos en detalle los componentes principales que se encuentran en el RCX:

TBPN API

Debido a las limitaciones que nos impone el hardware del RCX (escasa capacidad de procesamiento, poco espacio de memoria RAM disponible), hemos decidido implementar una segunda librería de clases que también implementa una red neuronal del tipo multi-layer perceptron, a la que hemos denominado Trained Backpropagation Network Application Program Interface (TBPN API).
Como su nombre lo indica, esta red ya esta entrenada y la utilizaremos en el proceso de clasificación que se ejecutará en el RCX.

La clase TrainedBackPropagationNet es generada dinámicamente al ejecutar el programa de entrenamiento Trainer que reside en la PC. Esta clase es más liviana que la clase BackPropagationNet, posee todos los métodos de esta última, excepto los métodos de aprendizaje. El resto de las clases que se relacionan con la misma, es decir WeightMatrix, NeuronLayer y Neuron, son idénticas a las anteriores. La relación entre estas clases se muestra en el siguiente diagrama:

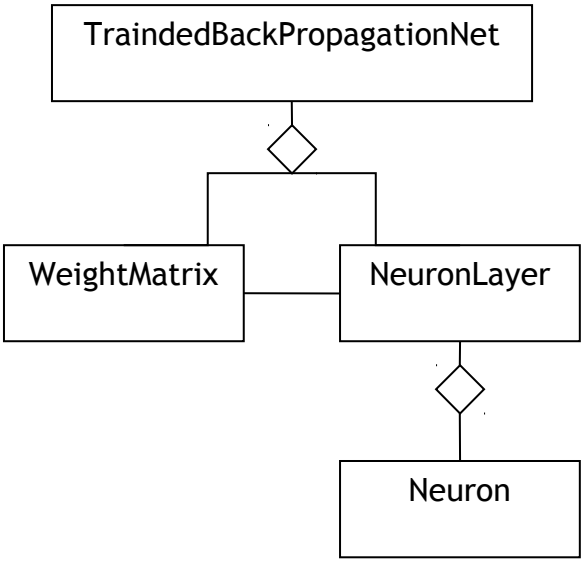

Figura 18 – Diagrama de clases TBPN API

Classifier

El programa de aplicación que denominamos **Classifier,** solicita un patrón a clasificar. Para ello, solicita al **Viewer** una imagen codificada que es transmitida mediante el puerto inflar ojo, desde la PC al RCX. El siguiente diagrama muestra las clases relevantes involucradas y sus relaciones:

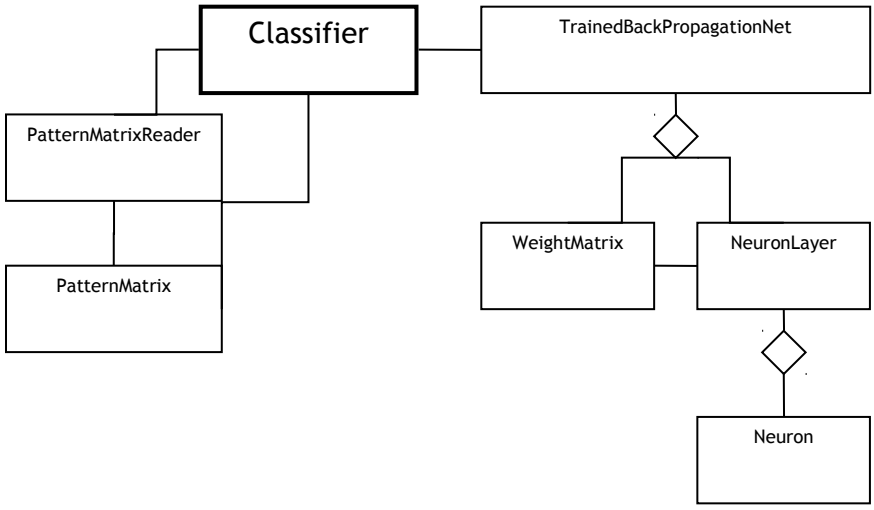

Figura 19 – Diagrama de clases del Classifier

Para finalizar, el siguiente diagrama de componentes, representa los módulos de software del programa de aplicación **Classifier.** Recordemos que este programa se ejecuta dentro del RCX y es el que le suministra inteligencia y motricidad al robot:

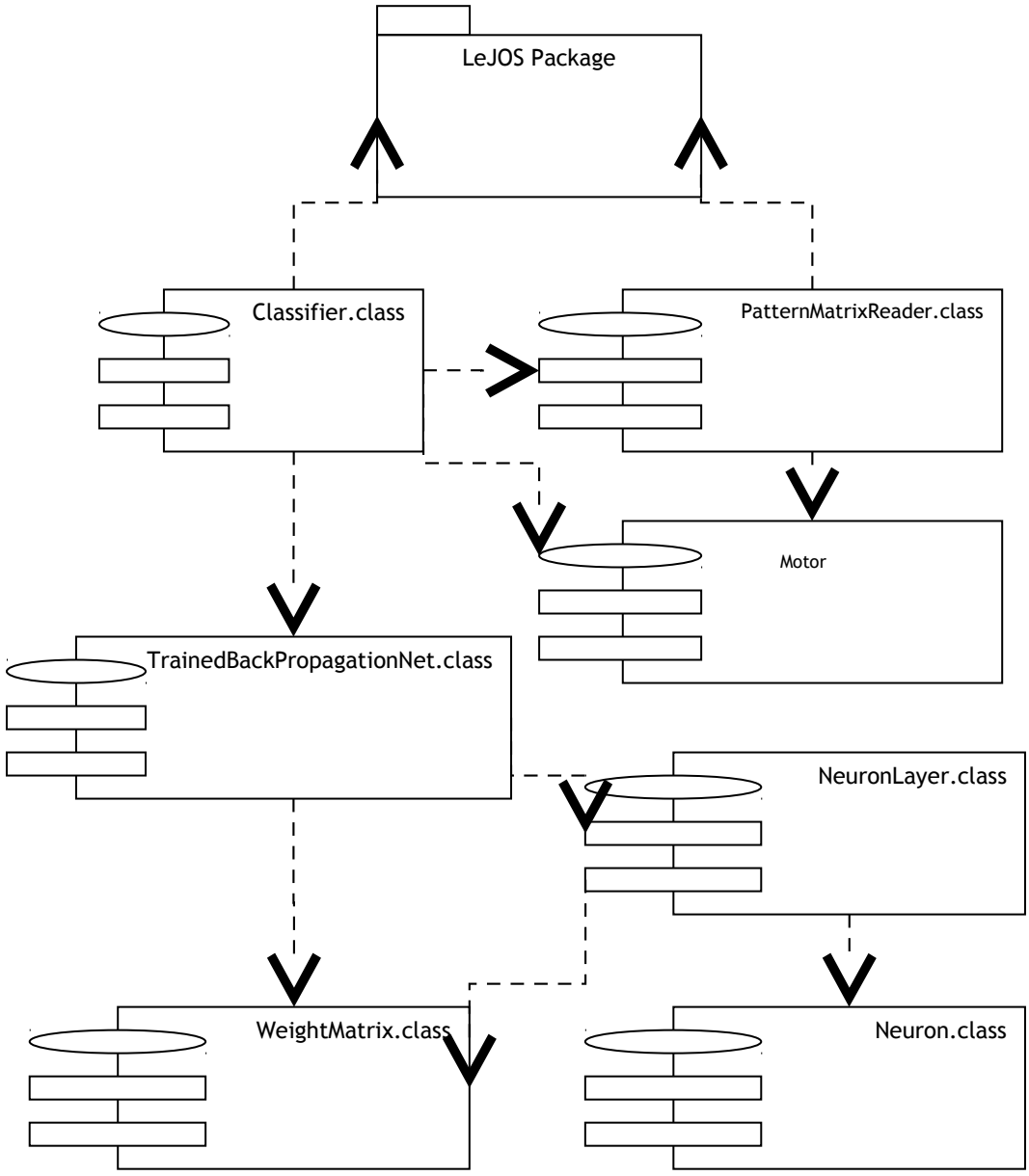

Figura 20 – Diagrama de componentes del Classifier

Comunicación PC – RCX

La comunicación entre la PC y el RCX es bidireccional y se realiza por medio del puerto infrarrojo en la PC y el lector/emisor de infrarrojos del RCX. Esta comunicación se establece por un lado desde el RCX para requerir una captura

de imágenes con la webcam en la PC, y por otro lado para enviar la imagen codificada desde la PC al RCX para su posterior clasificación.

Toda esta comunicación entre el RCX y la PC se encuentra implementada por medio de la utilización del protocolo LNP.

Demostración

Como corolario de nuestra tesis presentaremos a continuación un caso de ejemplo que demostrará las capacidades de la librería desarrollada.

Problema

En nuestro caso de ejemplo partimos con el problema de clasificar tres patrones diferentes. Estos patrones se encuentran codificados como matrices de cuatro pixels de ancho por cuatro pixels de alto, pudiendo cada uno de estos pixels estar coloreado con blanco o negro. Los patrones elegidos para este caso de ejemplo son los siguientes:

Figura 21 – Patrones de ejemplo

Solución

Veremos a continuación cuál fue el modelo de red y enfoque de entrenamiento utilizados para implementar nuestro caso de ejemplo.

Modelo de Red

Dado el universo de patrones a clasificar identificamos cuál sería la estructura de la red neuronal necesaria para resolver el problema de clasificación en cuestión. El modelo de la red elegida contendrá tres niveles. Un nivel de entrada, un nivel oculto y un nivel de salida. La estructura de la misma se presenta en la figura 22.

Entrenamiento

Uno de los puntos clave para llegar a una solución adecuada en la clasificación de patrones está íntimamente relacionado con la actividad de entrenamiento de la red neuronal. En nuestro caso entrenamos la red neuronal por medio de un archivo plano que contiene la representación binaria de los patrones a clasificar.

Figura 22 – Red Neuronal de ejemplo

Conclusiones y trabajos futuros

En este trabajo, hemos diseñado e implementado una librería de clases en Java para construir y utilizar una red neuronal del tipo multi-layer perceptron con algoritmo de aprendizaje back-propagation, en una plataforma con recursos limitados como lo es el RCX de Lego.

El trabajo terminó siendo desarrollado de una manera distinta a la pensada originalmente, debido a la escasa disponibilidad de recursos en el RCX. Esto nos obligó a pensar y experimentar en implementaciones alternativas, como la de distribuir la carga computacional en la PC y en el RCX, sin perder el objetivo que nos hemos propuesto.

A pesar de las limitaciones que nos enfrentamos, leJOS demostró ser una exitosa implementación de Java para ser utilizada en una plataforma especial como lo es el RCX. La filosofía "hacelo una vez, ejecutalo donde quieras" pudo ser comprobada y fue de gran utilidad en el desarrollo de nuestro trabajo. Esto nos facilitó programar y testear la aplicación a ejecutar en el RCX, sin inconvenientes en una plataforma distinta como la PC.

La solución que hemos propuesto fue evaluada exitosamente con distintos modelos de redes neuronales. Hemos experimentado el comportamiento del algoritmo de aprendizaje variando la cantidad de neuronas por nivel, el número de niveles, el tipo de patrón a clasificar y el margen de error aceptado.

Como trabajos futuros, proponemos las siguientes mejoras:

- Que el programa de aplicaciones **Trainer** genere automáticamente todo el código a ejecutar en el RCX y no solamente la clase TrainedBackPropagationNet.

- Extender la librería para poder utilizar otros algoritmos de aprendizaje diferentes al back-propagation.

- Generalizar algunas clases de nuestra librería, como por ejemplo la clase Neuron, de forma tal que facilite al desarrollador elegir su propia función de activación.

- Explotar la capacidad de representación del modelo de patrones que hemos elegido para nuestra solución. Por ejemplo, poder entrenar la red neuronal para que no solo reconozca patrones monocromáticos y planos sino también patrones con texturas, color y sombras.

Referencias

Bibliografía

[1] Brian Bagnall (2002), **Core Lego Mindstorms Programming**, Editorial Prentice Hall PTR

[2] Darío Laverde (2002), **Programming Lego Mindstorms with Java**, Editorial Syngress

[3] Stuart Russel (1997), **Inteligencia Artificial, un enfoque práctico**, Editorial Prentice Hall

[4] Kiran Hedge (2001), **AI Research Paper: Character Recognition using Backpropagation Neural Networks**

[5] Niels S. Andersen (2001), **Advanced Programming of the Lego RCX for Education.** Technical University of Denmark.

[6] 6836 Final Project (2002), **Evolution in the Micro-Sense: An Autonomous Learning Robot.** MIT.

[7] Ben Kröse (1996), **An Introduction to Neural Networks.** University of Amsterdam.

[8] Steven R. McVey (1996), **An Introduction to Neural Networks and Application.** Accenture.

[9] Hans-Erik Eriksson and Magnus Penker (1998), **UML Toolkit.** Editorial Wiley.

[10] Ian E. Darwin (2001), **Java Cookbook.** Editorial O'Reilly.

[11] Horstmann Cornell (1999), **Core Java 2 Fundamentals - Volumen I y II.** Editorial Prentice Hall / Sun Microsystems Press.

Internet

[1] Lego Mindstorms, www.legomindstorms.com

[2] lejOS, http://lejos.sourceforge.net

[3] IEEE Spectrum, http://www.spectrum.ieee.org/pubs/spectrum/0901/mind.html

[4] Foro Lugnet, Robotics, http://www.lugnet.com/robotics/

[5] The MIT Lego Robots Design Competition,
http://lcs.www.media.mit.edu/people/fredm/projects/6270/

[6] Generation 5, http://www.generation5.org

[7] The Java Tutorials,
http://developer.java.sun.com/developer/onlineTraining

[8] Lego Mindstorms Internals, http://www.crynwr.com/lego-robotics

[9] RCX Internals, http://graphics.standford.edu/~kekoa/rcx