Sistema configurable de simulación 3D para robótica móvil. Plataforma de desarrollo K++

Recibido 20 de septiembre de 2006, aprobado 30 de noviembre de 2006.

Juan Bautista Martínez Suárez

Ingeniero Electrónico. Gerente unidad de negocios Desarrollo en Internet, GoalNet Ltda. Miembro Grupo de investigación en manejo eficiente de la Energía Eléctrica GIMEL (Categoría A, Colciencias), Universidad de Antioquia, línea de Robótica y Mecatrónica (GIRAA). Medellín, Colombia.

juanbm@epm.net.co

Nelson David Muñoz Ceballos

Ingeniero Electrónico, M.Sc. en Ingeniería. Depto. de Ingeniería de Sistemas. Miembro del Grupo de investigación en manejo eficiente de la Energía Eléctrica GIMEL (Categoría A, Colciencias), Universidad de Antioquia, línea de Robótica y Mecatrónica (GIRAA). Medellín, Colombia.

nmunoz@udea.edu.co

Nelson Londoño Ospina

Ingeniero Electrónico, candidato a Ph.D. Universidad del Valle. Profesor Asociado, Departamento de Ingeniería Eléctrica. Miembro del Grupo de investigación en manejo eficiente de la Energía Eléctrica GIMEL (Categoría A, Colciencias), Universidad de Antioquia, en la línea de Robótica y Mecatrónica (GIRAA). Medellín, Colombia.

nlondono@udea.edu.co

RESUMEN Se presenta un sistema software que permite simular y controlar robots móviles, parametrizar características hardware, visualizar el estado de sensores y accionar actuadores (simulados o reales). K++ utiliza un lenguaje de programación propio. Se describen los resultados obtenidos en la programación de un algoritmo de control.

PALABRAS CLAVE

Robótica móvil, Simulación, ActiveX, DirectX, Sockets.

Configurable 3D Simulation System for Mobile Robotics. K++ Development Platform

ABSTRACT A software system for simulation and control of mobile robots is presented, which allows hardware characteristic parametrization, sensors state visualization and to drive actuators (simulated or real). K++ uses an own programming language. Results from the programming of control algorithm are described.

KEYWORDS

Mobile robotics, Simulation, ActiveX, DirectX, Sockets.

INTRODUCCIÓN

La mayoría de los sistemas de desarrollo o los simuladores en robótica móvil están definidos para un robot específico y generalmente no son compatibles con otros tipos de robots, por ejemplo: Kiks [1], kMatlab [2] y Amorsim [3], entre otros. No obstante, son muchos los robots que no cuentan con sistemas de desarrollo estructurado ni posibilitan herramientas de simulación y validación. Existe la necesidad de contar con una plataforma software versátil y reconfigurable para el estudio y validación de diferentes tipos de robots móviles, especialmente, plataformas robóticas de investigación. Para suplir esta necesidad, se diseñó un sistema de desarrollo para robótica móvil denominado K++, compuesto por: un simulador y un lenguaje de programación que permite desarrollar aplicaciones de control, ligado a un compilador que ofrece las posibilidades de convertir en ejecutable cualquier desarrollo implementado; dicha programación es de tipo estructurada teniendo como premisa el fácil manejo de la sintaxis. Adicionalmente, el simulador permite el levantamiento de mapas de entorno (navegación 2D en mapas de bits y 3D con DirectX). Como sistema base, se tomó ejemplo de las características del robot Khepera [4]. Se implementó un sistema de módulos funcionales o primitivas de software que describe las características y comportamientos del robot y permite simular motores, sensores, actuadores, etc. Dichos módulos fueron construidos en componentes ActiveX desarrollados en Visual Basic. Se usó la eficiencia del desarrollo RAD (Rapid Application Development), la reutilización de los componentes ActiveX [5] y la potencialidad de creación de escenarios 3D con DirectX [6].

Para la investigación y el desarrollo de software para robots autónomos, es muy importante contar con un entorno de simulación lo más real posible, así como con un software de alto nivel que permita sobrellevar las problemáticas de bajo nivel de los dispositivos robóticos. Sin embargo, muchas veces es necesario también poder modificar características de bajo nivel para, por ejemplo, añadir nuevos dispositivos o modificarlos. Es por ello fundamental contar con un sistema de control y simulación abierto, portable, escalable y reusable.

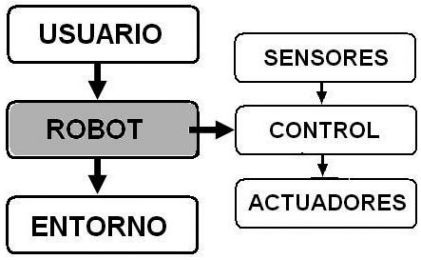

Figura 1. Diagrama de bloques de una arquitectura básica de un robot.

ARQUITECTURA TÍPICA DE UN SISTEMA ROBOT

Una breve descripción de la configuración típica de un robot se presenta en la Figura 1 [7]; este esquema básico es el utilizado en la arquitectura de software propuesta en el simulador de K++.

• Usuario: define la tarea que debe realizar el robot y la supervisa.

• Robot: conforma básicamente el hardware y software del sistema. A su vez consta de:

Sensores: Elementos que perciben señales del ambiente y la convierten en información.

Algoritmos de Control: Estructura computacional formada por diversos módulos interconectados y cuya función es garantizar el cumplimiento de las tareas.

Actuadores: Elementos eléctro-mecánicos que generan el desplazamiento del robot.

•Entorno: lo constituye todo sistema externo que interactúa con el robot, puede ser estático o dinámico.

SISTEMA K++

El K++ está compuesto por diferentes elementos (Ver Figura 3):

EL SIMULADOR K++

Es interactivo y visual. En la Figura 2 se observa parte del aspecto del simulador, que se implementó aplicando técnicas de manejo en:

- Mapas de bits para detección de bordes, posicionamiento, navegación y levantamiento de mapas.

- DirectX para la creación y navegación de espacios en 3D en base a imágenes en 2D.

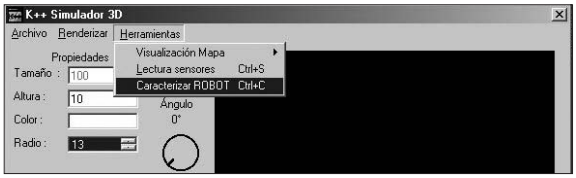

Figura 2. Acceso a caracterizar un robot.

- Matrices de transformación homogéneas para la rotación y translación de objetos en 2D y 3D.

- Sockets para la comunicación entre el programa de control y el simulador.

- Funciones trigonométricas para el cálculo de la dirección de los sensores durante la navegación.

El simulador permite, mediante menús, realizar un conjunto de funciones entre las que se destacan:

- *Parametrización virtual del robot:* Uno de los aspectos más importantes del sistema K++ es poder definir un robot móvil que cumpla con una serie de características preestablecidas, que pueden ser modificables dependiendo del tipo de robot que se desee simular. Este simulador permite definir los sensores (definición polinomial de la respuesta del sensor), los actuadores (definición de la cinemática) y los comandos de control asociados a funciones primitivas (definición de la velocidad de cada motor, respuesta de los conversores A/D asociados a los sensores, etc.). Por ejemplo: la opción de parametrización de un robot está disponible en el simulador mediante menú de Herramientas, Caracterizar Robot, como se

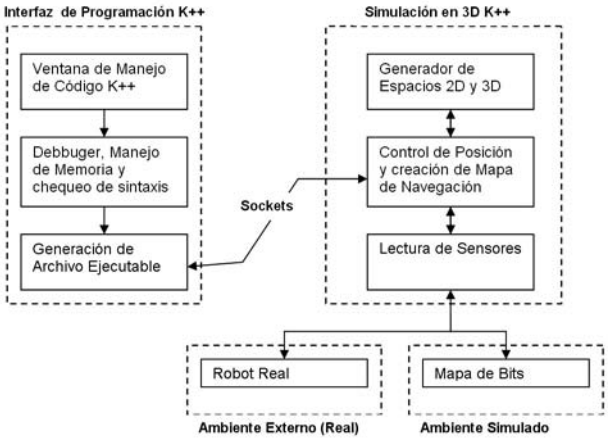

Figura 3. Diagrama de Bloques del Compilador/Simulador K++.

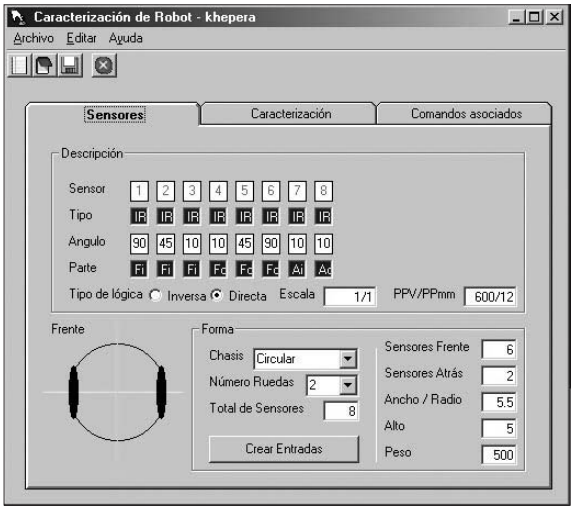

Figura 4. Vista inicial del Parametrizador.

observa en la Figura 2, y las opciones del parametrizador, que se indican en la Figura 4, así:

• *Sensores:* Cantidad de bit, de conversión lógica de uso, introducción de funciones de transferencia (gráficas, tabulada o por ecuaciones)

• *Actuadores:* Cantidad, propiedades físicas, eléctricas, característica cinemática y dinámica, etc.

• *Comandos Asociados* (subsección del parametrizador): Se implementó debido a la necesidad de cambiar el tipo de algoritmo de control de los actuadores y la información enviada al robot real vía RS-232.

En el simulador es posible realizar varias operaciones, entre las que se resaltan:

- Cambio del ángulo de inicio en la navegación.

- Guardado automático de la imagen del entorno recorrido (mapa de navegación), para evitar que por olvido, por falla de energía o por bloqueo del PC, se pierda el mapa generado.

- Cambio del entorno de navegación.

- Posibilidad de cambiar la curva característica de los sensores (respuesta discretizada del sensor al material de las paredes del entorno).

- Ver las coordenadas del mapa y el ángulo de navegación respecto a la referencia absoluta.

- Visualizar el espacio de trabajo en 3D (Figura 5).

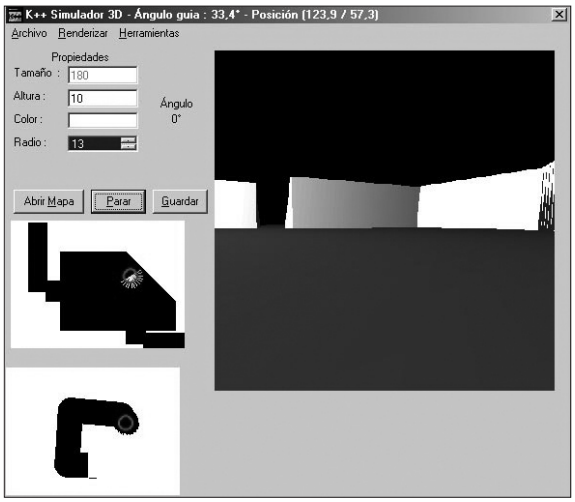

Figura 5. Vista del Simulador K++.

- Cambio del tipo de renderización (re-dibujado de polígonos) del mapa en 3D. Los tipos de renderizado disponibles son: por puntos, por líneas, por sólidos.

- Detener temporalmente la simulación, cambiar el ángulo de navegación, salvar el mapa generado.

- Facilitar que el usuario pueda establecer una posición inicial para el robot simulado.

- Permite ver el entorno de simulación por el que se navega y el mapa de entorno generado por los sensores del robot simulado, como se observa en la Figura 5.

- Visualizar y almacenar mapas de entorno mientras se efectuaba la simulación o la navegación del robot real, por lo que se optó por hacerlo parte implícita del sistema; en este caso se pueden generar 2 tipos de mapas:

• Por detección de borde: El mapa se genera cuando uno de los sensores simulados detecta un borde, en esa coordenada se dibuja un punto.

• Por espacio recorrido: Se genera dejando un rastro del espacio ocupado por el robot, es decir, mientras los sensores permitan el desplazamiento del robot, éste genera una marca por donde se movió.

EL COMPILADOR K++

Un segundo componente desarrollado en el proyecto es el compilador, dada la necesidad de cambiar la estructura de control sin tener que cambiar el programa de simulación en sí. Corresponde a la interfaz donde se van a programar todos los algoritmos de control. La Figura 6 ilustra el ambiente de trabajo del compilador. La necesidad de estructurar, por medio de un lenguaje de programación específico, los algoritmos de control, llevó a la implementación de técnicas de programación de alto nivel y condujo a la creación de un lenguaje, denominado K++.

LENGUAJE DE PROGRAMACIÓN

Trabajar con algoritmos de control para robots, exige aplicar lenguajes de programación de alto nivel y, por tanto, comandos más específicos para aplicaciones en robótica. K++ cuenta con archivos fuente propios que contienen básicamente:

- Una estructura algorítmica regida por un estilo de programación imperativo, funcional, modular y orientado a objetos.

- Los archivos fuentes con extensión .kpp o .dpp (Archivos Backup).

- Las líneas de código terminarán con punto y coma.

- Protocolo de declaración, especificación, inicialización de variables y de funciones.

- El compilador con depuración de errores.

- Menú ayuda.

- Conversión a un programa ejecutable.

- La comunicación que realiza el programa es mediante sockets, no se requiere configurar.

- Incluye comandos propios para cada tarea, como por ejemplo WSC que permite escribir el dato de control en el puerto para ser enviado al robot real o para el robot simulado.

El siguiente es un listado de algunas de las variables, comandos y métodos utilizados por el lenguaje:

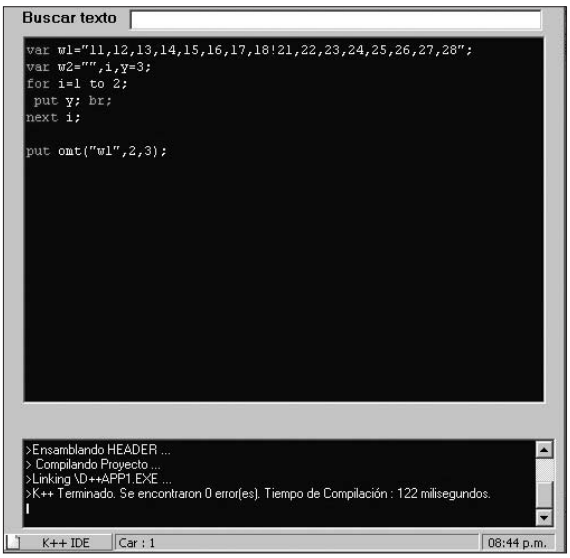

Figura 6. Vista del compilador K++.

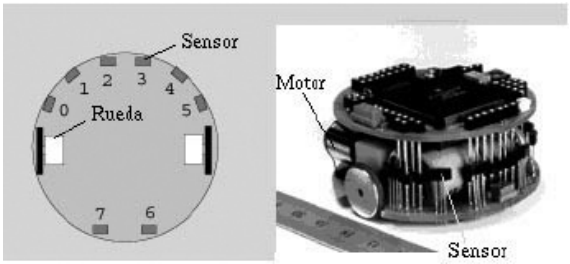

Figura 7. Algunas características del khepera.

put [expresión];	Muestra texto, todo al tiempo.
inp [variable];	Entrada de datos, no acepta nulos.
pass [variable];	Entrada de datos encriptada(*).
br;	Retorno de carro (siguiente línea).
title [expresión];	Título de la aplicación; se muestra en la barra de título.
delete [expresión];	Borra un archivo específico.
box [expresión],[título];	Crea un cuadro de diálogo, [título] es opcional.
pause [expresión];	Pausa el programa un tiempo específico.
open [expresión];	Abre un archivo especifico.
web [expresión];	Enlaza a una URL.
wav [expresión];	Ejecuta un archivo WAV (audio).
var [variable];	Declara una nueva variable.
var [variable]=[valor]	Declara una variable y la asigna.
var [var], [var], [var];	Declara más de una variable.
[variable] = [expresión];	Asignación de variable.
hide;	Oculta la aplicación.
show;	Muestra la aplicación.
show_controls;	Muestra los controles Minimizar & Cerrar de la ventana.
hide_controls;	Oculta los controles Minimizar & Cerrar de la ventana.
enable_cad;	Habilita CTL_ALT_DEL.
disable_cad;	Deshabilita CTL_ALT_DEL.
end;	Finaliza el programa en esa posición.
finish;	Finaliza ejecución del programa.
'	Comodín para comentarios.

VALIDACIÓN DE RESULTADOS

Para validar los elementos del K++ se propuso una estrategia de control de la navegación del robot móvil khepera (Figura 7), basada en la aplicación de una técnica de inteligencia computacional:

NAVEGACIÓN Y EVASIÓN DE OBSTÁCULOS BASADA EN UNA RED NEURONAL ARTIFICIAL (RNA)

Varias investigaciones sobre la aplicación de las redes neuronales en el control de robots móviles se describen en [8]. En este caso, se implementó una red neuronal perceptron multicapa con algoritmo de aprendizaje backpropagation, función de activación tangente hiperbólica, con 8 entradas, una capa oculta de 6 neuronas y una capa de salida con 2 neuronas. La arquitectura de la red se observa en la Figura 8 y la estructura de cada neurona se observa en la Figura 9. Los patrones de entrenamiento representaron diferentes estados robot-obstaculos [9]. La red neuronal se entrenó mediante Matlab. Después de 1200 iteraciones, el error disminuye notablemente y luego de 2000 iteraciones se obtuvo un error de 10^{-10}. Las características de la red considerada son:

- Matriz de entrada [8 x #], 8 sensores y # casos.*

- Matriz de salida [2 x #], 2 motores (actuadores) y # acciones esperadas.*

- Función de activación, tangente hiperbólica.

- Algoritmo backpropagation rápido con factor adaptable de aprendizaje y momentum.

- Rata de aprendizaje 0.5.

- Error 0.1.

*Donde, # = 7 en simulación y 40 en la implementación real en el robot.

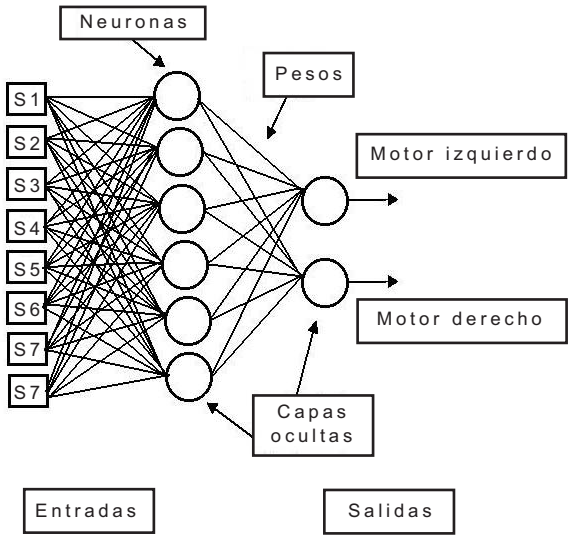

Figura 8. Representación de la RNA

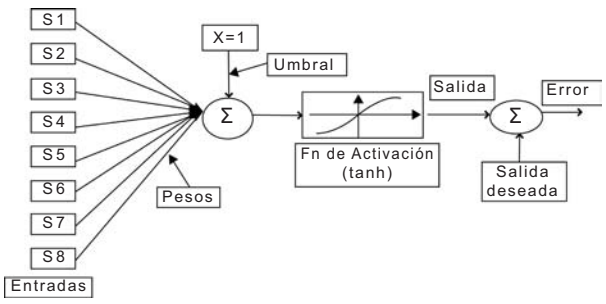

Figura 9. Estructura de cada neurona

CÓDIGO DE LA RED NEURONAL EN LENGUAJE DE PROGRAMACIÓN K++

Con la red entrenada, se programó la arquitectura de la figura 9, utilizando el lenguaje de programación K++ y el compilador. A continuación se observa parte del algoritmo codificado:

```
inc "ej\robot";
function main( )
{
        title "Red Neuronal";
        // matriz de pesos primera capa oculta
        var w1 ="-3.1329,0.9322,-4.3843,1.4555,1.1492,-
                2.0926,0.2629,4.8811, -0.8426,0.3324,
                0.6651,2.1470,-1.2618,2.1959, 0.7445,
                3.2244, 2.0572,-.0133,1.8638,0.5167,
                -2.2901,3.6034,1.3471,1.2931,1.4219,
                0.6584,-1.0265,0.9059,0.1803,5.4044,
                0.8321,- 0.9517,0.1575,1.6699,2.5773,
                3.1493";
        // matriz de pesos segunda capa oculta
        var w2="-0.8023, 1.5231 ,-0.2285,-4.6099,-3.1600,-
                2.3097!7 .5366,-4.5653,-2.2480,
                -2.1292,-2.1529,5.5818";
        // bias de la primera capa oculta
        var b1 ="0.0739,-2.6911,5.4271,-1.4515,-3.3440,-
                0.5035";
        // bias de la segunda capa oculta
        var b2="0.9832,2.2533";
        var gn1="0,0,0,0,0,0";
        var gn2="0,0";
        var i,j,tmp,x,z,y =3,Vel_nom=50;
        Call arranca( );
}
Function arranca( ) //sólo se ejecuta una vez, para un control
continuo debe estar en un ciclo infinito
{
        for i=1 to 6;
                tmp=0;
                for j= 1 to 8;
                    tmp=tmp+( omt("senx", 1 j)*omt("w1",i,j));
                next j;
                tmp=tmp+omt("b1",1,i);
                tmp=tanh(tmp);
```

```
            tmp=imt("gn1",1,i,tmp);
        next i;
        for i=1 to 2;
            tmp=0;
            for j=1 to6;
                tmp=tmp+(omt("gn1",1,j)*omt("w2",i,j));
            next j;
            tmp=tmp+omt("b2",1,i);
            tmp=imt("gn2",1,i,tmp);
        next i;
        Wsc "D," & omt("gn2", 1,1) & "," & omt("gn2", 1 ,2),0;
}
```

Los resultados obtenidos en la simulación presentaron una respuesta aceptable; no se observan colisiones, pese a que en la red sólo se especificaron 7 patrones de entrenamiento. Sin embargo, fue necesario reentrenar la red con más patrones de entradas – salidas (un total de 40 patrones), para que se desempeñara aceptablemente en el entorno real (aún se presentaron algunas colisiones), dado que aspectos de la dinámica del sistema como la inercia, la fricción, entre otros, no fueron contemplados en la simulación. Además, se nota una gran influencia del ruido ambiente en los sensores infrarrojos de distancia, tanto por la luz de día, como por la iluminación artificial de las lámparas fluorescentes del laboratorio.

CONCLUSIONES

Se diseñó e implemento una herramienta de amplia aplicación en el estudio, desarrollo y validación de sistemas robotizados, que facilita el diseño de algoritmos de control para robots móviles, incluyendo la caracterización de diferentes dispositivos electrómecánicos, sensores, actuadores, entre otros.

El compilador, el parametrizador y el simulador permiten definir un robot con base en una serie de características preestablecidas, que pueden ser modificables y acondicionadas a diferentes tipos de robots móviles. La posibilidad de creación de mapas de entorno derivados de la detección de borde o seguimiento de ruta, es uno de los más importantes logros de este proyecto.

Los resultados obtenidos en la implementación del algoritmo de control y navegación dan pie a que el estudio de esquemas más complejos puedan ser implementados y validados mediante esta herramienta, lo cual facilita el trabajo a estudiantes e investigadores relacionados con el área de robótica móvil y permite desarrollar la aplicación de diferentes tipos de control en robots móviles. Como trabajo futuro, se abre la posibilidad de hacer simulaciones remotas ya que la comunicación entre el programa ejecutable (generado por el compilador) y el simulador se realiza a través de sockets de comunicación; éste sería un proyecto complementario.

REFERENCIAS

[1] T. Nilsson.
Kiks User Guide. 2001.

[2] Y. Piguet, S. Legon.
kMatlab -- Matlab commands for Khepera. K-Team SA. Acceso septiembre 20 de 2006. http://ftp.k-team.com/khepera/matlab/readme.txt

[3] T. Petrinić, E. Ivanjko, I. Petrović.
"AMORsim – A Mobile Robot Simulator for Matlab" En *Proceedings of 15th International Workshop on Robotic*. Balatonfüred, Hungary, 2006.

[4] Khepera user manual.
K-Team SA. Acceso enero 15 de 2006. http://www.k-team.com/

[5] K. Moore.
ActiveX Control Tutorial.
Developer.com. Acceso Mayo 15 de 2005. http://www.developer.com/net/vb/article.php/1539541

[6] B. Correas Suárez.
Fundamentos Gráficos por Computador. 2003.

[7] N. Londoño Ospina.
Análisis y diseño de una arquitectura para robots, desde una concepción metodológica. Medellín, 2002.

[8] A. M. Zalzala, y A. S. Morris.
Neural networks for robotic control. ED. Ellis Horwood, 1996.

[9] G. Pezzotti, N. Londoño, J. Valencia.
"Redes neuronales para evitación de obstáculos en robótica móvil". En *Memorias IX Congreso Latinoamericano de Control Automático*. Santiago de Cali, noviembre de 2000.