

ECBOT: Arquitectura Abierta para Robots Móviles

Carlos Iván Camargo – Universidad Nacional de Colombia
cicamargoba@unal.edu.co

Abstract—Este artículo presenta el trabajo realizado en el diseño de una plataforma abierta para robótica móvil, enfatizando sus puntos más importantes: la arquitectura Software y Hardware. Esta plataforma permite la ejecución de Linux de forma nativa, lo que posibilita la ejecución de un servidor Player y la utilización de simuladores como Stage(2D) / Gazebo (3D). La última parte de este artículo presenta algunas aplicaciones de la plataforma y los planes de utilización a futuro.

Index Terms—Sistemas Embebidos, Robótica Móvil, Linux, Player.

I. INTRODUCCIÓN

La robótica Móvil representa un campo de investigación que crece rápidamente. En la actualidad existe un gran número de grupos de investigación que trabajan en el desarrollo de técnicas de auto-organización para sistemas Multi-Robot, entre los que se encuentran: El proyecto Swarm-bots [1], [2], Interaction Labs (USC) [3] [4], Autonomous System Lab (EPFL), MIT Computer Science and Artificial Intelligence Laboratory [5]. De las experiencias obtenidas de estos proyectos, se deduce, que para desarrollar algoritmos de inteligencia artificial, es necesario contar con plataformas Hardware y Software que permitan validar los algoritmos y modelos computacionales propuestos. La característica común de los proyectos ya mencionados es la implementación física de los algoritmos sobre robots reales, lo cual separa estos trabajos de investigaciones similares en el área de los Sistemas Multi-Agente, los cuales son desarrollados en plataformas Software.

ECBOT fue desarrollado en el Departamento de Ingeniería Eléctrica y Electrónica de la Universidad Nacional de Colombia, con el objetivo de proporcionar una plataforma **abierta** Hardware y Software de bajo costo que permita el desarrollo de aplicaciones en Sistemas Multi-Robot (MRS). El eje central del desarrollo Hardware es el sistema operativo (OS) Linux, ya que este impone requerimientos que de cierta forma definen la arquitectura del componente Hardware del dispositivo. Por otro lado, la columna vertebral del sistema Software es el proyecto *Player/Stage* [6], el cual, suministra funciones de alto nivel que encapsulan el manejo tedioso de bajo nivel de los sensores y actuadores, haciendo posible que la programación del Robot se realice en una variedad de lenguajes de programación como Java, Lisp, Python, C/C++, etc.

Se presenta una plataforma robotica comparable en capacidades Software y Hardware a las plataformas comerciales más utilizadas en el área. A continuación se listan dos de las plataformas más populares, resaltando sus recursos Software y Hardware así como su costo.

• Khepera III

- Procesador: DsPIC y Xscale @ 400MHz

- Movimiento: 2 Servo motores DC
- Sensores, Entrada/Salida:
 - * 9 sensores Infrarojos de proximidad y luz de ambiente
 - * 2 sensores Infrarojos para seguimiento de línea
 - * 5 Sensores de Ultrasonido
 - * 16 I/O Digitales, 8 Analógicas
 - * 2 Leds Programables
- Comunicación:
 - * Puerto serie Standard
 - * Comunicación USB
 - * Ethernet Inalámbrico
- Simuladores: WEBOTS¹
- Herramientas de Desarrollo: GNU TOOLS
- Costo: 2950 USD

• EPFL e-puck²

- Procesador: DsPIC
- Movimiento: 2 Servo motores DC
- Sensores, Entrada/Salida:
 - * 8 sensores Infrarojos de proximidad y luz de ambiente
 - * 1 Acelerómetro
 - * 3 micrófonos
 - * 1 cámara de color.
 - * 9 Leds Programables
- Comunicación:
 - * Puerto serie Standard
 - * Puerto Infrarojo
 - * Enlace Bluetooth
- Simuladores: WEBOTS
- Herramientas de Desarrollo: GNU TOOLS
- Costo: 1100 USD

II. ARQUITECTURA GLOBAL DE ECBOT

ECBOT está formado por la unión de dos grandes componentes: El Componente Hardware y el Componente Software, el primero constituye el dispositivo físico mediante el cual se interactúa con el entorno y sobre el cual el componente Software implementa un determinado algoritmo. Para mayor comprensión primero se realizará la presentación de la Arquitectura Software.

A. Arquitectura Software

La columna vertebral del componente Software es el proyecto *Player/Stage*, el cual, es el resultado de un proyecto de investigación del Grupo de Investigación en Robótica de la University of Southern California. Este proyecto está dividido en tres partes:

- 1) **Player** Es un servidor que proporciona una interfaz flexible a una gran variedad de sensores y actuadores. Como puede verse en la Figura 1 utiliza un modelo cliente/servidor basado en sockets TCP, su principal característica es que permite el manejo de los sensores y actuadores a través de una interfaz de programación de alto nivel, lo cual permite que los programas de control del robot sean escritos en cualquier lenguaje y puedan ser ejecutados en cualquier plataforma³ (cliente) que posea una

¹<http://www.k-team.com>

²<http://www.e-puck.org>

³También es posible que el cliente y el servidor se ejecuten en el mismo robot.

conexión de red con el robot (servidor). Además, Player soporta múltiples conexiones concurrentes de clientes, lo cual es muy útil en estudios de control descentralizado.

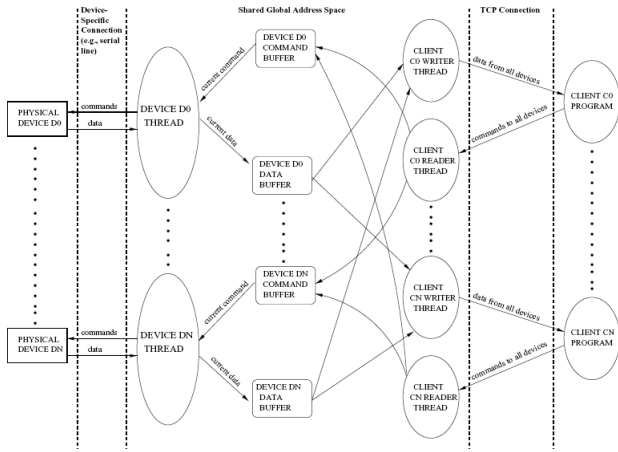


Fig. 1. Arquitectura de Player. [7]

- 2) **Stage** Es un simulador escalable; que simula robots que se mueven y realizan operaciones de sentido en un entorno de dos dimensiones, estos robots son controlados por Player. Stage proporciona robots virtuales los cuales interactúan con dispositivos simulados. En la Figura 2 se puede observar una simulación multi-robot utilizando las librerías de Stage; una de las características más atractivas del proyecto Player-Stage es que permite la creación de sensores y actuadores que simulan el comportamiento de los dispositivos reales.
- 3) **Gazebo** Es un simulador multi-robot 3D. A diferencia de Stage que fué diseñado para simular grupos numerosos de robots, gazebo simula el comportamiento de poblaciones pequeñas de robots (menos de 10).

1) **Arquitectura Hardware:** La mayoría de los robots que utilizan Player utilizan el puerto serie para conectarse con el robot ⁴ y es necesario que un dispositivo externo ejecute el servidor de Player y realice las tareas control por este medio, esto obliga a

⁴Roomba: <http://www.irobot.com>, Khepera: <http://www.k-team.com>, Pioneer: <http://www.activrobots.com/>

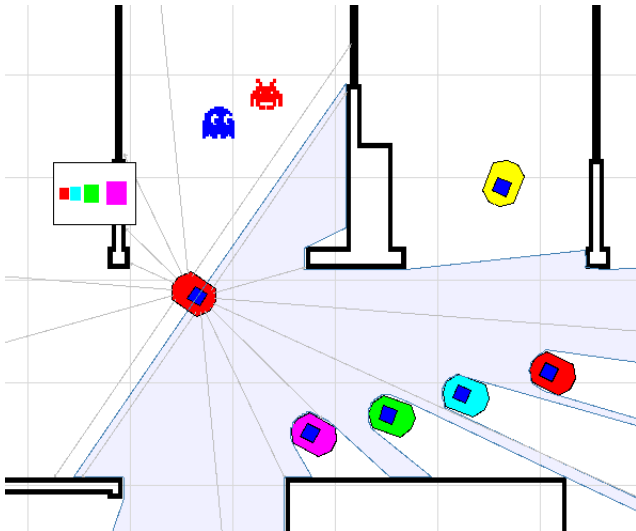


Fig. 2. Captura del simulador Stage.

tener una conexión física entre el dispositivo en el que se está ejecutando el servidor de player y el robot, lo cual resulta poco práctico en algunas aplicaciones. Algunos fabricantes de robots⁵ solucionaron este problema adicionando dispositivos inalámbricos (BlueTooth, ZigBee) para permitir la creación de “puertos seriales inalámbricos”; sin embargo, esto no elimina la necesidad de un dispositivo externo que se encargue de ejecutar el servidor de Player. Por esta razón, uno de los requerimientos es que ECBOT sea capaz de ejecutar de forma nativa a Player. Para esto, es necesario que ECBOT pueda correr aplicaciones Linux, o lo que es lo mismo que soporte el sistema operativo Linux. Por esta razón el primer paso en el desarrollo de ECBOT fue el estudio de plataformas que soportaran linux Embebido, como resultado de este estudio se diseñó la plataforma abierta: ECB_AT91 [8], la cual fué la base del desarrollo que dio como resultado ECBOT. En la figura 3 se muestra el Diagrama de Bloques de ECBOT,

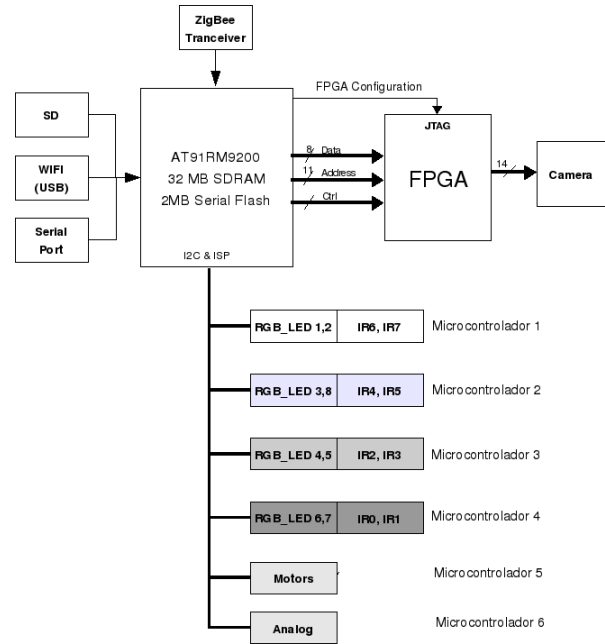


Fig. 3. Diagrama de Bloques del componente Hardware de ECBOT.

B. Especificaciones de ECBOT

1) **Sensores y Actuadores:** Como puede verse en La Figura 3 ECBOT cuenta con un bus I2C encargado de realizar la interfaz con el mundo análogo (ya que el procesador central AT91RM9200 de ATmel no posee conversores A/D), seis microcontroladores de 8 bits (AVR de ATmel) permiten el manejo de los sensores y actuadores a través de este bus. Cuatro microcontroladores de 8 bits, manejan cada uno de ellos: dos sensores infrarojos de proximidad y luz de ambiente, utilizados en tareas de evasión de obstáculos y 6 LEDs (2-rojos, 2-verdes, 2-azules) que representan el estado interno del Robot; el quinto microcontrolador de 8 bits se encarga de manejar la velocidad y posición de 2 motores DC, para reducir el costo de los componentes ECBOT implementa un control de posición y velocidad de motores utilizando un método de medición de la velocidad basado en la fuerza electromotriz [9], [10]. El bus I2C se encuentra disponible y puede ser utilizado para adicionar cualquier sensor que cumpla con su protocolo.

Adicionalmente ECBOT cuenta con una cámara digital que permite la implementación de algoritmos de seguimiento de color, una FPGA se encarga del manejo del sensor de imagen y de la

⁵Garcia: <http://www.acroname.com>

comunicación con el procesador. Gracias a que ECBOT cuenta con 32MB de memoria RAM es posible capturar una imagen completa de 640x480 pixels. Adicionalmente la FPGA proporciona 10 señales de propósito general que pueden ser utilizadas para el manejo de servos o para el manejo de servo motores.

Los microcontroladores de 8 bits y la FPGA son programados por el procesador central a través de pines de Entrada/Salida de propósito general, los archivos de programación/configuración son almacenados en la plataforma y pueden ser modificados en cualquier momento; esto hace que ECBOT sea una plataforma independiente y que pueda ser programada “en caliente”.

2) *Comunicaciones*: ECBOT proporciona 3 formas de comunicación:

- Módulo WiFi: ECBOT permite la conexión de un adaptador USB 802.11, en la actualidad solo los adaptadores basados en el chip ZD1211 están soportados.
- Módulo 802.15.4: ECBOT cuenta con el transceiver de Texas Instruments CC2420.
- Puerto Serie: En las primeras etapas del desarrollo es necesario utilizar el puerto serie para cargar las imágenes del loader, bootloader y kernel.

3) *Memorias y Dispositivos de Almacenamiento*: Se cuenta con una memoria Flash serial de 2 Mbytes donde se almacenan las imágenes del loader, bootloader y kernel, esta memoria puede ser modificada utilizando un loader que se ejecuta al inicializar la plataforma, o puede modificarse desde linux a través de un dispositivo MTD (Memory Technology Device). ECBOT permite la utilización de una memoria SDRAM de hasta 64 MBytes, la cual es utilizada como memoria de propósito general para las aplicaciones que corren bajo Linux.

ECBOT utiliza la distribución de linux *Buildroot*, la cual esta basada en la librería de C *uClibc*, concebida para trabajar con sistemas embebidos. Buildroot es altamente configurable y permite seleccionar los paquetes que serán parte del sistema de archivos, lo cual optimiza el espacio requerido por este. Adicionalmente, permite la adicionar fácilmente nuevos paquetes, en nuestro caso se agregaron 3 nuevos: El servidor Player, el programador de microcontroladores para AVR: *uisp*⁶ y el programador para FPGAs: *xc3sprog*⁷. El sistema de archivos generado por Buildroot y las aplicaciones necesarias para el funcionamiento de ECBOT son almacenadas en una memoria SD.

4) *Unidad Central de Procesamiento*: El cerebro de ECBOT es un procesador de 32 Bits de la familia ARM de ATMEL el AT91RM9200, que corre a 180 MHz. Este procesador goza de gran popularidad dentro del grupo de desarrolladores de drivers para Linux, por lo que casi la totalidad de sus periféricos están soportados. Esto facilita la creación del soporte necesario para la board; la información necesaria sobre el *port* de Linux a la plataforma se puede encontrar en [11].

5) *Fotografías de la tarjeta principal de ECBOT*: En las Figuras 4 y 5 se muestra el lado de componentes y el lado de soldadura de la tarjeta principal de ECBOT; en ellas podemos observar la localización de los componentes, la cual está fuertemente influenciada por el robot *e-puck* [12] del EPFL.

III. SERVIDOR PLAYER PARA LA PLATAFORMA ECBOT

Una de las características más atractivas del proyecto Player/Stage es que es de carácter libre y su código fuente está disponible, gracias a esto, fué posible escribir un driver que soportara a ECBOT. Este driver permite controlar:

- 1) El dispositivo de seguimiento de color implementado en la FPGA.
- 2) La velocidad y posición de 2 motores.
- 3) El color de los LEDs.
- 4) La activación y la lectura de los sensores InfraRojos de proximidad.

⁶<http://www.nongnu.org/uisp/>

⁷<http://www.rogerstech.co.uk/xc3sprog/>

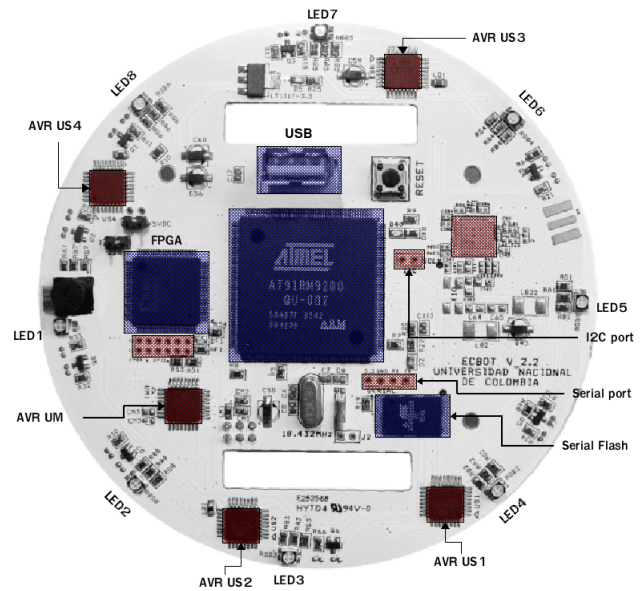


Fig. 4. Lado de Componentes de la tarjeta principal de ECBOT.

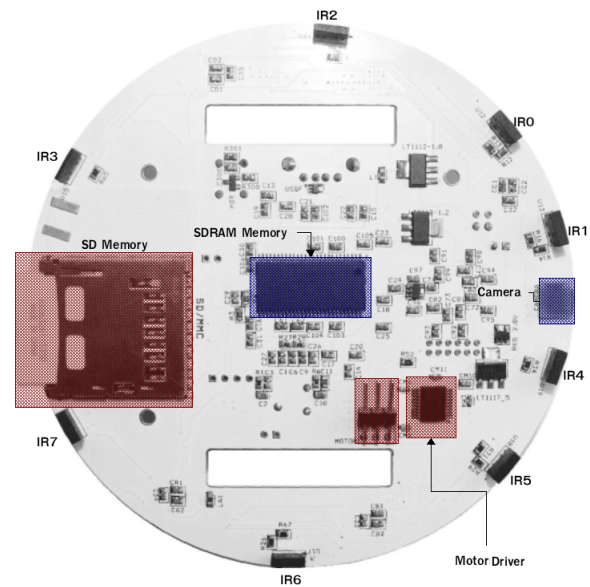


Fig. 5. Lado de soldadura de la tarjeta principal de ECBOT.

La figura 6 muestra gráficamente la arquitectura del servidor Player de ECBOT, como puede verse, el servidor está encargado de comunicarse con los sensores (cámara y sensores InfraRojos), procesar esta información y enviarla al cliente, donde el algoritmo de control por intermedio de un generador de comandos del servidor controla los actuadores (LEDs y motores). En esta misma figura se muestra una base de bajo costo para la plataforma ECBOT.

A manera de ejemplo a continuación se muestran dos archivos típicos al utilizar Player/Stage. El primer archivo, muestra la configuración para el cliente de Player:

```
driver
(
  name "ecbot"
  provides ["position2d:0" "ir:0" "blinkenlight:0"]
)
```

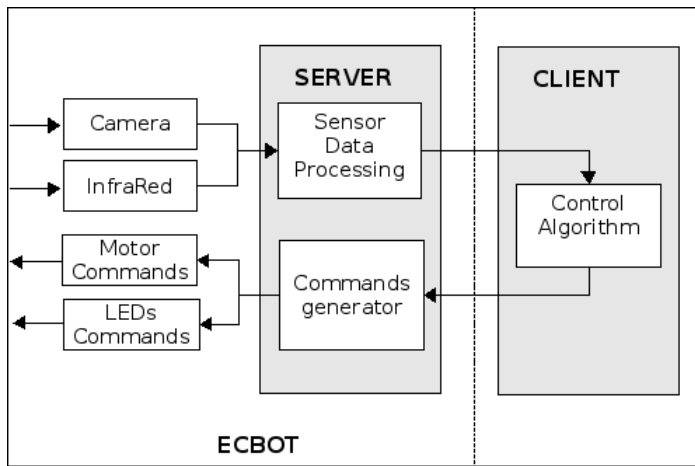


Fig. 6. Servidor Player de ECBOT.

Y el segundo el código para un cliente que controla la velocidad de los motores, el color de los leds y lee el estado de los sensores InfraRojos:

```

1 int main(int argc, char *argv[])
2 {
3     using namespace PlayerCc;
4     using namespace std;
5
6     int preferredTurn = 1;
7
8     std::cout << "connecting to robot" << std::endl;
9     PlayerClient robot("ecbot");
10
11     std::cout << "connecting to ir proxy" << std::endl;
12     IrProxy ir(&robot, 0);
13
14     std::cout << "connecting to pos proxy" << std::endl;
15     Position2dProxy pp(&robot, 0);
16
17     std::cout << "connecting to LED proxy" << std::endl;
18     BlinkenLightProxy bp(&robot, 0);
19
20
21     pp.SetMotorEnable(true);
22
23     bp.SetColor(0xf0, 0x01, 0x01); // LED color RED
24
25     for(;;)
26     {
27         char turnrate = 0x40;
28         char speed = 0x40;
29         unsigned char IR[8];
30
31         robot.Read();
32         ++samples;
33
34         bp.SetColor(0x01, 0xf0, 0x01); // LED color GREEN
35         pp.SetSpeed(speed, turnrate); // Forward
36         for(int i=0; i<8; i++){
37             IR[i] = ir.Voltages[i]; // Read IRs
38         }
39     }
40 }

```

En la línea 9 se realiza la conexión con el servidor de Player del robot *ecbot*⁸; en 12, 15 y 18 se definen los *proxies* IR, Position2d y BlinkenLight, los cuales permiten la interacción con los sensores y actuadores del robot. En

⁸*ecbot* es la dirección IP del robot dentro de la red

las líneas 21 y 34 se cambia el color de los LEDs a Rojo y verde respectivamente; en la línea 35 se fija la velocidad de los motores de tal forma que el robot avanza; finalmente, en la línea 37 se lee el valor de los sensores de proximidad. Como puede verse de estos dos ejemplos, la utilización del proyecto Player/Stage facilita la implementación de algoritmos de control en robótica.

IV. CONCLUSIONES Y TRABAJO FUTURO

La contribución de este trabajo es el desarrollo de una plataforma **abierto** Hardware y Software para aplicaciones de robótica móvil. A continuación se listan las tareas que se realizaron para la creación de esta plataforma:

- Adecuar Linux para que se ejecute en un procesador ARM de 32 bits: Para esto fue necesario, la modificación de drivers existentes y la definición de una nueva plataforma.
- Diseño de las placas de Circuito Impreso: Para llegar al prototipo final, se implementaron tres tarjetas que permitieron probar los diferentes módulos de ECBOT. La primera se diseñó para probar el procesador y para realizar la implementación de Linux; en la segunda tarjeta se probó la interfaz del procesador y una FPGA, y la comunicación I2C con microcontroladores de 8 bits para permitir la lectura de señales analógicas. La tercera es la que se muestra en la Figura 7 y corresponde a la plataforma final.
- Creación o modificación de software para programación de los microcontroladores de 8 bits y configuración de la FPGA.
- Escritura de las rutinas de control de los sensores y actuadores seleccionados: Motores, sensores infrarrojos, y LEDs.
- Adaptación del software Player/Stage para dar soporte a la plataforma robótica.
- Creación de los módulos de interfaz entre el procesador y la FPGA, e implementación del procesamiento de imagen en la FPGA.

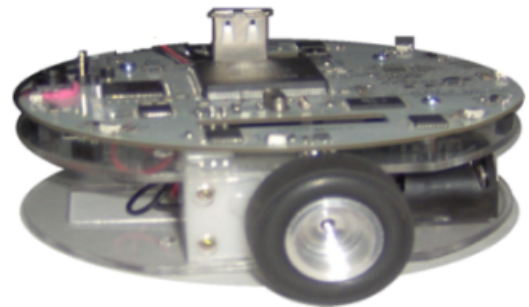


Fig. 7. Servidor Player de ECBOT.

Todo el software, hardware y archivos necesarios para la fabricación de ECBOT se encuentran disponibles, su distribución es libre, esto es lo que da a la plataforma la categoría **abierto**.

El componente Hardware de esta plataforma permite la ejecución del servidor Player de forma nativa, lo cual simplifica la implementación de algoritmos de control. Por otro lado, se presenta una plataforma abierta, es decir, los archivos necesarios para la fabricación de la tarjeta de circuito impreso, los esquemáticos, el firmware de los microcontroladores y su respectivo código fuente, los cambios requeridos al kernel de Linux para realizar estudios sobre el sistema operativo Linux en sistemas Embebidos.

Este trabajo es el punto de partida en el estudio de técnicas de auto-coordinación en sistemas Multi-Robot, ECBOT fué diseñada de tal forma que permita la realización de experimentos clásicos en este estudio. El siguiente paso consiste en la fabricación de un grupo de 10 Robots y en la implementación de algoritmos que permitan a este grupo realizar un trabajo determinado de forma autónoma.

V.

REFERENCES

- [1] F. Mondada, G. C. Pettinaro, I. Kwee, A. Guignard, L. Gambardella, D. Floreano, S. Nolfi, J.-L. Deneubourg, and M. Dorigo. SWARM-BOT: A swarm of autonomous mobile robots with self-assembling capabilities. In C.K. Hemelrijk and E. Bonabeau, editors, *Proceedings of the International Workshop on Self-organisation and Evolution of Social Behaviour*, pages 307–312, Monte Verità, Ascona, Switzerland, September 8–13, 2002. University of Zurich.
- [2] G. Baldassarre, S. Nolfi, and D. Parisi. Evolving mobile robots able to display collective behaviours. In C.K. Hemelrijk and E. Bonabeau, editors, *Proceedings of the International Workshop on Self-Organisation and Evolution of Social Behaviour*, pages 11–22, Monte Verità, Ascona, Switzerland, September 8–13, 2002. University of Zurich.
- [3] C. Jones and M. Mataric. Adaptive Division of Labor in Large-Scale Minimalist Multi-Robot Systems. *Proceedings of the IEEE/RSJ International Conference on Robotics and Intelligent Systems (IROS)*. Las Vegas, Nevada, 2003.
- [4] Jones and Maja J Mataric. *Autonomous Mobile Robots: Sensing, Control, Decision-Making, and Applications*, chapter Behavior-Based Coordination in Multi-Robot Systems. Marcel Dekker, Inc, 2005.
- [5] J. McLurkin. Speaking Swarmish. *AAAI Spring Symposium*, 2006.
- [6] B.P. Gerkey, R. T. Vaughan, and A. Howard. The Player/Stage project: Tools for Multi-robot and Distributed Sensor Systems,. *Proceedings of the International Conference on Advanced Robotics (ICAR)*, pages 317–323, 2003.
- [7] B. Gerkey, K. Stoy, and R. T. Vaughan. Player Robot Server. Technical report, USC Robotics Labs, 22 November 2000.
- [8] C. Camargo. First Colombian Linux SBC runs Debian. <http://www.linuxdevices.com/news/NS6698241512.html>, 24 April 2006.
- [9] Abu Zaharin and Mohd Nasir. A Study On The DC Motor Speed Control By Using Back-EMF Voltage. *ASIAN CONFERENCE ON SENSORS*, July 2003.
- [10] R. LeGrand. Closed-Loop Motion Control for Mobile Robotics. *Circuit Cellar Ink*, August 2004.
- [11] C. Camargo, N. Castillo, and A. Calderón. Free ECB_AT91. <http://wiki.emqbit.com/wiki>.
- [12] EPFL. e-puck EPFL Education robot. <http://www.e-puck.org/>.



Carlos Camargo es Ingeniero Electricista de la Universidad Nacional de Colombia, Magister en Ingeniería Eléctrica de la Universidad de los Andes y Candidato a Doctor en Ingeniería Eléctrica en la Universidad Nacional de Colombia. Es

profesor del Departamento de Ingeniería Eléctrica y Electrónica de la Universidad Nacional de Colombia en el área de los sistemas Digitales.