Programación y simulación en robótica móvil utilizando Player/Stage

Gonzalo F. Péred Paina   David A. Gaydou
Centro de Investigación en Informática para la Ingeniería, CIII
Universidad Tecnológica Nacional, Facultad Regional Córdoba, UTN-FRC
gperez@scdt.frc.utn.edu.ar

Resumen— **En el área de investigación de robótica móvil es de gran importancia contar con herramientas flexibles tanto para la programación como simulación. En el presente trabajo se describen la utilización del entorno de desarrollo libre Player/Stage, y la manera de incluir un robot móvil de diseño propio en dicho entorno; para lo cual es necesario generar el driver correspondiente de Player y el modelo de simulación de Stage. Además, se presentan varios ejemplos de aplicación práctica de dicho entorno en la robótica móvil, utilizando diferentes sensores.**

I. Introducción

En el área de investigación en robótica móvil es de gran importancia contar con herramientas flexibles tanto para la programación como simulación, que permita simular el robot junto con sus sensores, actuadores y entorno controlado facilitando el desarrollo y validación teórica de algoritmos de navegación autónoma. En la actualidad los principales fabricantes de robots incluyen plataformas de desarrollo para los usuarios de sus productos, por ejemplo ActivMedia ofrece la plataforma ARIA, para sus robots Pioneer, PeopleBot, etc.; Evolution Robotics vende su plataforma ERSP, y Sony ofrece OPEN-R para sus Aibo. Además, muchos centros de investigación han creado sus propias plataformas de desarrollo como la suite de navegación CARMEN de Carnegie Mellon University, Player/Stage, OROCOS, etc. Estos entornos de desarrollo de robots ofrecen un acceso más abstracto y simple a sensores y actuadores, además de funcionalidades de uso común como algoritmos de control, localización, navegación segura, construcción de mapas, etc. Los entornos de desarrollo creados por universidades tienen la principal ventaja de que permiten incorporar hardware no soportado.

El presente trabajo describe la experiencia en la utilización del entorno de desarrollo Player/Stage en el robot `RoMAA` [1] no soportado originalmente por dicho entorno. El entorno de desarrollo Player/Stage fue seleccionado considerando el balance final del trabajo presentado en [2], donde se analizan nueve Entornos de Desarrollo de Robótica de código abierto que, además de describir sus ventajas y desventajas, propone un método objetivo de evaluación incluyendo aspectos como la abstracción de hardware, lenguaje de programación, impacto en la comunidad robótica, etc.

La sección II describe brevemente el robot `RoMAA` y el controlador embebido. En la sección III se introduce el entorno de desarrollo Player/Stage, describiendo en detalle los pasos necesarios para ser utilizado en la programación del robot `RoMAA` en la sección IV. La sección V muestra la flexibilidad en la utilización de diferentes dispositivos de robótica para diversos experimentos al utilizar un entorno de desarrollo. Y por último, la sección VI presenta las conclusiones del trabajo.

II. El robot RoMAA

El robot `RoMAA` es un robot móvil de tracción diferencial tipo uniciclo con dos ruedas de tracción y una rueda castor de apoyo detrás (Fig. 1). Los motores de tracción se realimentan a partir de las lecturas de los encoders ópticos incrementales acoplados a cada rueda.

El sistema embebido a bordo implementa los lazos de control en velocidad para cada uno de los motores de tracción, cálculo de odometría del robot a partir de las lecturas de los encoders ópticos incrementales, y la comunicación con la PC de control de alto nivel a bordo del vehículo. Dispone de tres modos de funcionamiento; el primer modo permite controlar al robot mediante comandos de velocidad lineal y angular agregando un lazo de control externo conocido como cross-coupling [3]; el segundo modo evita este lazo y permite acceder directamente y de forma independiente a cada uno de los lazos de control de los motores de tracción mediante comandos de velocidad, y el tercer modo permite controlar los motores directamente con comandos de señal de PWM, funcionando en lazo abierto. Además, el controlador permite el acceso a variables internas del lazo como las lecturas de los encoders, ya sea en forma de cantidad de pulsos, velocidad angular, desplazamiento lineal recorrido; ajustar los valores de las constantes de los controladores, etc.

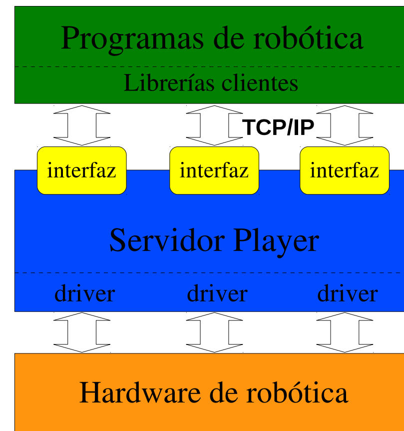
III. Entorno de desarrollo Player/Stage

Player y Stage fueron originalmente desarrollado en el laboratorio de investigación de robótica de la *University of Southern California* -USC, Robotics Research Lab- [4], y actualmente es un proyecto activo de SourceForge [1]. Player es un servidor de dispositivos utilizados en robótica basado en sockets que proporciona una interfaz simple a sensores y actuadores en redes TCP/IP, la abstracción de los sockets posibilita la independencia del lenguaje de programación y de la plataforma de trabajo.

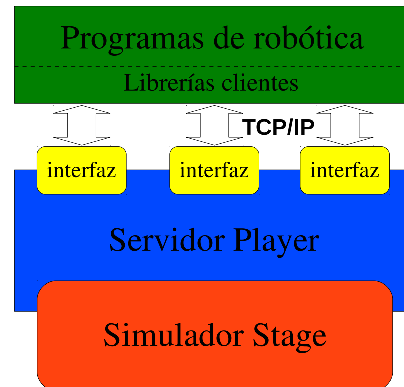
Player consta de dos partes, una parte funciona como servidor de red para el control de robots y corre abordo del

[1] http://playerstage.sourceforge.net/

Fig. 1: Robot Móvil de Arquitectura Abierta - RoMAA



(a)Player con robot real



(b)Player con el simulador Stage

Fig. 2: Diagrama de Player con robot real y con el simulador Stage

robot. Actúa como una capa de abstracción de hardware (HAL) para dispositivos robóticos en el caso de robots reales -Fig. 2(a)- , o bien sirve de interfaz al simulador Stage -Fig. 2(b)-. La otra parte son las librerías clientes que brindan acceso a los dispositivos remotos (reales o simulados) en el lado del servidor.

Los programas clientes, a través de interfaces predefinidas en Player, se comunican con el servidor para leer/escribir datos desde/hacia los dispositivos reales o simulados. En el caso de dispositivos reales, se necesitan drivers para adecuar los comandos de bajo nivel del dispositivo a las interfaces predefinidas en Player. Estas interfaces especifican cómo interactuar con cierta clase de sensores, actuadores o algoritmos de robots, definiendo la sintaxis y semántica de los mensajes que se pueden intercambiar con entidades de la misma clase de dispositivos. Algunas de las interfaces predefinidas en Player son `position1d` para actuadores con un grado de libertad, `position2d` para dispositivos que se mueven en el plano, `camera` para cámaras digitales, etc. Así, cualquier robot móvil utiliza la interfaz `position2d` independientemente de la marca o modelo, y puede ser comandado de forma transparente para el programa cliente. Además de los driver necesarios para interactuar con dispositivos reales, Player dispone de algoritmos estándares de navegación, como localización, planificación de trayectoria, evasión de obstáculos, generación de mapas, etc., programados como drivers que no interactúan con dispositivos físicos (drivers abstractos) sino que procesan información de otros drivers para realizar acciones o proveer los resultados del proceso. Algunas de las interfaces utilizada para proveer información por estos drivers son `localize`, `map`, `planner`, etc.

Stage es un simulador de múltiples robots que simula una población de robots, sensores y objetos en un entorno 2D; dispone de robots virtuales de modo que Player interactúe con el entorno simulado en lugar de los dispositivos físicos, además de varios modelos de sensores incluyendo sonares, sensores láser rangefinder, cámaras pan-tilt-zoom, odometría, etc. Stage es adecuado para investigaciones de sistemas autónomos multi-agente, debido a que se basa en un modelo simple de bajo requerimiento computacional de múltiples dispositivos en lugar de emular cada dispositivo con gran fidelidad. Stage se puede utilizar también como un simulador independiente de Player, mediante la programación de controladores que se asocian a un modelo del entorno simulado, el cual se carga en tiempo de ejecución [5].

A partir de la versión 2.0 de Player [6] donde se realizaron mejoras en la estructura interna del programa permitiendo mayor flexibilidad y simplicidad, y de la versión 3.0 de Stage que agrega simulación 2.5D [7]; el conjunto de herramientas Player/Stage se han convertido en un estándar en la comunidad robótica "open source" [8].

A. *Servidor Player*

Como se mencionó, Player es una capa de abstracción que conecta el código de la aplicación (programa cliente) con los dispositivos reales de robótica o bien con dispositivos simulados mediante el simulador Stage. Al ejecutar Player se pasa como parámetro un archivo de configuración (.cfg) de texto plano indicando los dispositivos a utilizar. En el caso de dispositivos reales, el servidor Player se ejecuta en el computador conectado físicamente a los dispositivos del robot y el archivo de configuración indica qué drivers de dispositivo cargar y qué interfaces utili-

zar para estos drivers. Para interactuar con el simulador Stage, el archivo de configuración indica cargar el simulador, el entorno a simular, y las interfaces utilizadas para comunicarse con los dispositivos simulados.

B. Cliente Player

Las librerías clientes están disponibles en varios lenguajes para facilitar el desarrollo de programas clientes TCP. Estas librerías se utilizan para desarrollar los programas de control de robot con el servidor Player ya sea con hardware real o simulado. Las librerías disponibles oficiales del proyecto son en lenguaje C (`libplayerc`), C++ (`libplayerc++`) y python (`libplayerc_py`). Existen además otras librerías de contribución que incluyen `MATLAB`, `Smalltalk`, `Java`, `GNU/Octave`, etc.

C. Archivo de configuración

Los archivos de configuración le indican al servidor Player cuáles drivers utilizar y qué interfaces usan estos drivers. Para cada modelo en la simulación o dispositivo real con el que se necesita interactuar se debe especificar un driver en el archivo de configuración, mediante una sección de la siguiente manera

```
driver
(
  name "driver_name"
  provides [device_address]
  # other parameters...
)
```

Los parámetros `name` y `provides` son obligatorios, sin los cuales Player no sabe qué driver utilizar (dado por `name`) y qué tipo de información envía/recibe el driver usado (dado por `provides`). El parámetro `name` tiene que ser un nombre de los drivers incluidos en Player, o un driver propio en el caso de hardware no soportado oficialmente por Player. El parámetro `provides` le indica a Player que interfaz utilizar para poder interpretar la información dada por el driver, esta información se utiliza en el código de la aplicación de robótica. Además, cada driver en particular tiene parámetros de configuración extra. La dirección de dispositivo (`device_address`) se indica de la siguiente manera

```
provides [ "host:robot:interface:index"
           "host:robot:interface:index"
           "host:robot:interface:index"
           ...]
```

El campo `host` es la dirección IP donde se ejecuta el servidor de Player, `robot` es el puerto TCP, `inteface` es alguna de las interfaces predefinidas en Player (por ejemplo `position2d`, `camera`, `laser`, `imu`), e `index` permite tener varias interfaces del mismo tipo (por ejemplo `camera:0`, `camera:1`).

IV. Player/Stage en el robot RoMAA

A. Drivers de Player

Para poder controlar el robot móvil de tracción diferencial `RoMAA` es necesario desarrollar un driver de Player que adecua los comandos de bajo nivel del controlador embebido del robot con alguna de las interfaces predefinidas en Player. Los drivers de Player pueden ser de dos tipos, los drivers normales que vienen compilados con el servidor y los driver plugin que se cargan una vez ejecutado el servidor. Player incluye drivers para diferentes dispositivos comerciales de robótica (sensores, actuadores, robots, etc.). Player incluye librerías para la programación de los drivers plugin, que facilita generar un nuevo driver para un dispositivo no soportado. Una vez compilado el driver plugin se genera una librería compartida (archivo `libdrivername.so`) que se debe indicar en el archivo de configuración del servidor de Player.

Para interactuar con el `RoMAA` se escribió un driver plugin que utiliza la interfaz `position2d`, la que permite controlar mecanismos con tres grados de libertad (dos de posición y uno de orientación). Mediante esta interfaz se pueden enviar comandos de velocidad y consultar la odometría entre otras cosas.

B. Implementación del driver

El driver plugin se programa como una clase de C++ heredada de la clase `ThreadedDriver`, cuya declaración puede verse en el Listado 1

Esta clase tiene los siguientes métodos virtuales

- `Main()`: Es la función principal del thread del dispositivo, donde se deben incluir todas las interacciones con el mismo
- `MainSetup()`: Aquí se deben implementar las funciones de inicialización específicas del dispositivo por ej. abrir y configurar un puerto de comunicación serie. `MainSetup` ejecuta el thread del driver
- `MainQuit()`: Es el opuesto al método `MainSetup`, aquí se debe realizar la terminación de dispositivos como por ej. cerrar un puerto de comunicación; también se detiene el thread del driver
- `ProcessMessage()`: Este método se invoca con cada mensaje entrante al servidor y ofrece la posibilidad de enviar una respuesta publicando un mensaje utilizando la función miembro pública `Publish`

Listado 1: Declaración de la clase driver para el robot RoMAA

```
#include <libplayercore/playercore.h>

class Romaa : public ThreadedDriver
{
  public:
    // Constructor
    Romaa(ConfigFile* cf, int section);

    // This method will be invoked on each
    // incoming message
    virtual int ProcessMessage(
        QueuePointer &resp_queue,
        player_msghdr * hdr,
        void * data);

  private:
    // Main function for device thread.
    virtual void Main();
    virtual int MainSetup();
    virtual void MainQuit();
};
```

La inicialización del driver por parte del servidor Player utiliza un conjunto de funciones, algunas de ellas se describen en el Listado 2

El servidor Player inicia el driver plugin llamando a `player_driver_init` (función en C), esta función llama a `Romaa_Register`, la cual agrega el driver a una tabla de drivers del servidor Player mediante el método `AddDriver`, y ésta llama a `Romaa_Init`. `Romaa_Init` crea un nuevo objeto `Romaa` y devuelve un puntero al driver. Una vez creado el objeto `Romaa` se ejecuta el constructor. En el constructor se deben cagar los parámetros propios del driver indicados en el archivo de configuración. Luego se ejecuta `MainSetup` y el método principal `Main`.

Listado 2: Inicialización del driver

```
/* need the extern to avoid C++
   name-mangling  */
extern "C" {
  int player_driver_init(DriverTable* table)
  {
    puts("romaa driver initializing");
    Romaa_Register(table);
    puts("romaa driver done");
    return(0);
  }
}

void Romaa_Register(DriverTable* table)
{
  table->AddDriver("romaa", Romaa_Init);
}

Driver* Romaa_Init(ConfigFile* cf,
                   int section)
{
  // Create and return a new instance of
  // this driver
  return ((Driver*)(new Romaa(cf, section)));
}
```

El método `Main` es el núcleo del driver plugin y se ejecuta en un thread lo que significa que corre en paralelo con otros drivers. La mayor parte del método `Main` está contenido en un loop `for(;;)`. El método `Main` (Listado 3) debe llamar unas pocas funciones específicas

- `pthread_textcancel()`: Verifica si el thread fue terminado (killed). La función saldrá del loop y ejecutara el método `MainQuit`
- `ProcessMessages()`: Le pasa el control a Player que llamará al método `ProcessMessage` del plugin con cada mensaje esperando en la cola de mensajes
- `usleep()`: Es un comando de sleep para liberar los recursos utilizados por el driver

Listado 3: Método Main del driver de RoMAA

```
Romaa::Main(
{
  // The main loop; interact with the
  // device here
  for(;;)
  {
    // test if we are supposed to cancel
    pthread_testcancel();

    // Process incoming messages
    ProcessMessages();

    // Interact with the device, and push
    // out the resulting data, using
    // Driver::Publish()

    // Sleep (you might, for example,
    // block on a read() instead)
    usleep(100000);
  }
}
```

Procesamiento de mensajes

Las diferentes interfaces en Player interactúan unas con otras enviando y recibiendo mensajes a través de Player.

Tipos de mensajes

- **Commands**: Se utilizan para darle instrucciones al driver cuando no se requiere una respuesta
- **Request**: Son mensajes desde otros drivers para acceder a datos que no son publicados regularmente o enviar comandos que requieren algún tipo de respuesta
- **Data**: Los mensajes de datos son publicados en cada iteración del loop Main del driver

Archivo de configuración del robot RoMAA real

El Listado 4 muestra el archivo de configuración de Player para la utilización de las interfaces `position2d`, donde se ve el nombre del driver a cargar `romaa` el cual es un driver del tipo plugin, indicando mediante el parámetro `plugin` el nombre de la librería compartida generada al compilar el driver, `libromaa` en este caso. El campo obligatorio `provides` indica que dicho driver utiliza una interfaz del tipo `position2d`.

El parámetro `wheel_control` permite (cuando vale `1`) controlar directamente las velocidades de las ruedas o bien comandos de velocidad lineal y angular (cuando vale `0`). Los demás parámetros como `port` y `baudrate` indican el nombre del puerto serie y la velocidad de transferencia para comunicar el controlador embebido del robot con el computador a bordo que corre el servidor Player; y `motor_pid`, `t_odometry`, etc. son particulares de la implementación y sirven para configurar el controlador embebido del robot.

Listado 4: romaa.cfg

```
driver
(
  name           "romaa"
  plugin         "libromaa"
  provides       [ "position2d:0" ]
  port           "/dev/ttyUSB0"
  baudrate       115200
  motor_pid      [ 20.0 10.0 0.0 ]
  vw_pid         [ 8.0 4.0 0.0 ]
  wheel_control  0
  t_odometry     25
  t_loop         20
)
```

C. Simulación en Stage

Las aplicaciones robóticas desarrolladas mediante programas clientes de Player se puede simular en Stage ejecutando el servidor Player con un archivo de configuración que en lugar de cargar drivers de dispositivos reales ejecuta el simulador. Un ejemplo de archivo de configuración se muestra en el Listado 5.

Archivo de configuración para la simulación del robot RoMAA

En el caso de una simulación, el archivo de configuración de Player indica en una sección `driver` que cargue el simulador Stage, el cual esta programado como un driver plugin cuyo nombre es `stage` y librería compartida `libstageplugin.so`. El parámetro adicional del simulador Stage `worldfile` indica un archivo de texto (worldfile.world) donde se describe el entorno a simular. El archivo de configuración debe contener además una sección driver

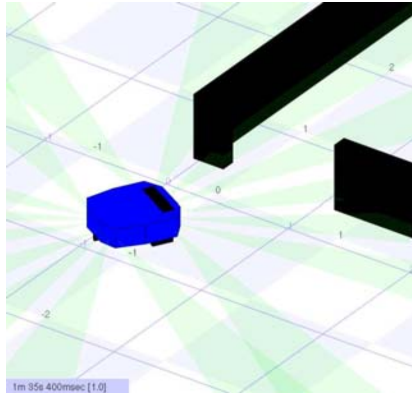


Fig. 3: Modelo de simulación del robot RoMAA para Stage con sensores de ultrasonido

adicional para cada dispositivo simulado utilizando como interfaz algunas de las disponible en Player. En el ejemplo se simula el robot `RoMAA` con interfaz `position2d` y el parámetro adicional `model` se refiere al modelo del robot descripto dentro del archivo `.world`. La Fig. 3 muestra el modelo de simulación del robot `RoMAA`.

En [9] se explica detalladamente como crear los archivos de configuración de Player y de descripción del entorno (archivos `.world`) de Stage.

Listado 5: Archivo de configuración para simulación en Stage

```
# Desc: Player configuration file for
# controlling RoMAA Stage simulator
# Author: Gonzalo F Perez Paina
# Date: 1 July 2009

# load the Stage plugin simulation driver
driver
(
  name       "stage"
  provides   [ "simulation:0" ]
  plugin     "libstageplugin"
  # load the named file into the simulator
  worldfile "empty.world"
)

# Create a Stage driver and attach
# position2d to the model "romaarobot"
driver
(
  name      "stage"
  provides [ "position2d:0" ]
  model     "romaarobot"
)
```

V. Flexibilidad de Player

El entorno de desarrollo Player/Stage ha sido utilizado en diversos experimentos incluyendo dispositivos como cámaras digitales, escáner láser, unidades pan&tilt, etc. A continuación se describen, a modo de ejemplo, los archivos de configuración utilizados en algunos casos concretos.

Listado 6: firewire.cfg

```
driver
(
  name       "camera1394"
  provides   [ "camera:0" ]
  framerate 15
  mode       "640x480_mono"
)
```

Para realizar experimentos de navegación de robótica móvil utilizando visión por computadoras se necesita obtener imágenes de una cámara digital a bordo del robot, las cuales pueden ser procesadas on-line o almacenadas para su procesamiento off-line, a medida que el robot realiza una trayectoria dada. El Listado 6 muestra una sección driver a utilizar en un archivo de configuración de Player que permite, utilizando el driver incluido en Player para cámaras FireWire, obtener las imágenes a través de la interfaz `camera`. Los datos provistos por la interfaz `camera` se pueden procesar en un programa cliente o bien ser de utilidad para otro driver de Player. En algunos casos es de gran utilidad poder almacenar estas imágenes para su procesamiento off-line. Player dispone de un driver dedicado al log de datos de nombre `writelog` que permite registrar datos disponibles en las interfaces de Player. El Listado 7 muestra cómo se carga el driver `writelog` para almacenar las imágenes a disco.

Además, esta cámara puede estar montada en una unidad pan&tilt para modificar su orientación hacia una zona de interés del entorno próximo del robot. Player incluye el driver para la unidad pan&tilt DirectedPerception PTU-D46 llamado `ptu46` que puede ser controlada mediante la interfaz `ptz` para dispositivos tipo pan-tilt-zoom; el listado 8 muestra cómo utilizar el driver para la PTU-D46.

En experimentos donde se requieren varios de estos driver se genera un archivo de configuración que contenga las secciones drivers necesarias. Por ejemplo para utilizar el robot `RoMAA` con una cámara FireWire montada sobre una unidad pan&tilt, se genera un archivo de configuración con los listados 4, 6 y 8.

Listado 7: writelogcam.cfg

```
driver
(
  name                 "writelog"
  log_directory        "/home/user/logdir"
  timestamp_directory 1
  basename             "image"
  requires             [ "camera:0" ]
  provides             [ "log:0" ]
  camera_save_images   1
  camera_log_images    0
)
```

Listado 8: pantilt.cfg

```
driver
(
  name      "ptu46"
  port      "/dev/ttyUSB0"
  provides [ "ptz:0" ]
)
```

Listado 9: writeloglaser.cfg

```
driver
(
  name                 "writelog"
  log_directory        "/home/user/logdir"
  timestamp_directory 1
  basename             "romaalaser"
  requires             [ "laser:0"
                         "position2d:0" ]
  provides             [ "log:0" ]
)
```

Player incluye driver para diferentes sensores láser utilizados en robótica como los de marca SICK y Hokuyo. Un

ejemplo de archivo de configuración del sensor láser SICK LMS200 se muestra en el Listado 10. El Listado 9 muestra un archivo de configuración para realizar el log de datos del sensor láser y de la odometría del robot a medida que navega en un entorno; estos datos se pueden reproducir luego para ajustar diversos algoritmos de robótica basados en esta información como el mapeo, SLAM, etc.

Listado 10: sicklms200.cfg

```
# Config file para laser sick lms200
driver
(
  name            "sicklms200"
  provides        [ "laser:0" ]
  port            "/dev/ttyUSB1"
  resolution      100
  serial_high_speed_mode        1
  serial_high_speed_baudremap  230400
  connect_rate    [ 9600 500000 38400 ]
  transfer_rate  38400
  alwayson        1
  delay           35
)
```

Los datos almacenados mediante el driver `writelog` se puede reproducir con Player utilizando el driver `readlog` a fin de poder acceder a estos datos almacenados simulando su procedencia de sensores reales.

Listado 11: readlog.cfg

```
# Play back odometry and laser data
# at twice real-time
driver
(
  name       "readlog"
  filename   "/home/user/logfile.log"
  provides   ["position2d:0"
              "laser:0"
              "log:0"]
  speed      2.0
)
```

VI. Conclusiones

En el presente trabajo se describió cómo incorporar un robot móvil al entorno de desarrollo de robótica Player/Stage, no soportado oficialmente por el proyecto, lo cual incluye la programación del driver para el servidor Player en el caso del robot real, y la generación de un modelo de simulación para el robot simulado en Stage.

La utilización de un entorno de desarrollo de robots como Player permite generar aplicaciones para robótica de una manera flexible y ágil, gracias a la abstracción del hardware basada en sockets para comunicarse con los dispositivos, además permite la reutilización de software. La arquitectura cliente/servidor brinda independencia del lenguaje de programación y la posibilidad de generar aplicaciones distribuidas sin imponer una modelo de software particular. Los simuladores como Stage permiten depurar el código de aplicación en un entorno virtual controlado antes de llevarlo a la realidad física.

Los experimentos realizados utilizando Player/Stage que estas herramientas son de gran utilidad y flexibilidad, permitiendo adecuar el conjunto de componentes acorde al experimento necesario de manera fácil, mediante la reutilización de herramientas o código previamente desarrollado y probado. Esto permite generar ciclos de desarrollos, simulación y experimentación más ágiles en un campo de constante evolución como es el de la robótica móvil.

Agradecimientos

Los autores se financian bajo el programa de Becas de Formación de Doctores en Áreas Tecnológicas Prioritarias. Ministerio de Ciencia, Tecnología e Innovación Productiva, Agencia Nacional de Promoción Científica y Tecnológica - FONCyT IP-PRH 2007 - Resolución C.S. Nº 649/08

El presente trabajo se enmarca bajo el Proyecto U.T.N. PID 1151 “Robot móvil de aquitectura abierta RoMAA-II”, homologado por disposición SCyT-UTN Nº34/10

Referencias

[1] D. A. Gaydou, G. F. Pérez Paina, G. M. Steiner, and J. Salomone. Plataforma móvil de arquitectura abierta. In *Actas de las V Jornadas Argentinas de Robótica 2008*. Ediuns, 2008, November 2008.

[2] James Kramer and Matthias Scheutz. Development environments for autonomous mobile robots: A survey. *Auton. Robots*, 22(2):101–132, 2007.

[3] Koren Y. Borenstein J. Feng, L. Cross-coupling motion controller for mobile robots. *Control Systems Magazine, IEEE*, 13(6):35 –43, Dec 1993.

[4] Brian P. Gerkey, Richard T. Vaughan, Gaurav S. Sukhatme, Kasper Stoy, Andrew Howard, and Maja J. Mataric. Most valuable player: A robot device server for distributed control, 2001.

[5] Richard Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2-4):189–208, 2008.

[6] Toby H. J. Collett, Bruce A. Macdonald, and Brian P. Gerkey. Player 2.0: Toward a practical robot programming framework. In *Proc. of the Australasian Conf. on Robotics and Automation (ACRA)*, Sydney, Australia, 2005.

[7] Collet T.H.J Wong N., Peng Hsu J.C. Improving the 2.5d stage robotic simulator. In *Proc. of the Australasian Conf. on Robotics and Automation (ACRA)*, Canberra, Australia, December 2008.

[8] Richard T. Vaughan, Brian P. Gerkey, and Andrew Howard. On device abstractions for portable, reusable robot code. In *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2421–2427, 2003.

[9] Jennifer Owen. *How to Use Player/Stage*, July 2009.