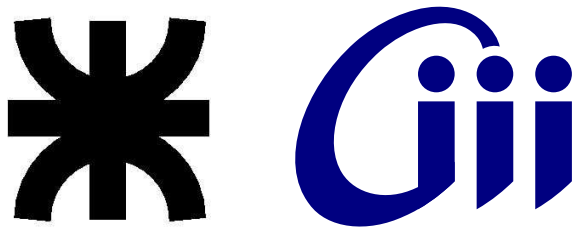
Entorno de desarrollo de robots Player/Stage/Gazebo

Gonzalo F. Perez Paina

Centro de Investigación en Informática para la Ingeniería
Universidad Tecnológica Nacional, F.R.C.
http://twiki.frc.utn.edu.ar/CIII



Diciembre 2009

Contenido

Contenido

Programación de robots

El desarrollo de programas de robots es muy diferente al de las demás aplicaciones de software, puesto que son sistemas complejos, y

- están conectados directamente a la realidad física a través de sensores y actuadores
- deben responder a varias fuentes de actividad y objetivos simultáneamente
- se enfrentan cada vez mas a una mayor heterogeneidad respecto al hardware utilizado

Además, para mayor flexibilidad se necesitan aplicaciones distribuidas y una interfaz gráfica de usuario (GUI) para facilitar la depuración de los programas

Programación de robots, simuladores

Objetivos

Depurar los algoritmos en un entorno virtual controlado que simule las observaciones de los sensores y las ordenes a los actuadores

Capacidades

Los simuladores actuales son capaces de simular sensores tan complejos como la visión, en un entorno de dos dimensiones o aun mas realistas en tres dimensiones con comportamiento dinámico de los elementos del entorno

Ejemplos

Dos de los simuladores mas utilizados son `SRIsim (MobileSim)` para los robots ActivMedia y los simuladores `Stage` y `Gazebo` de software libre

Entornos de desarrollo de robots (RDE)

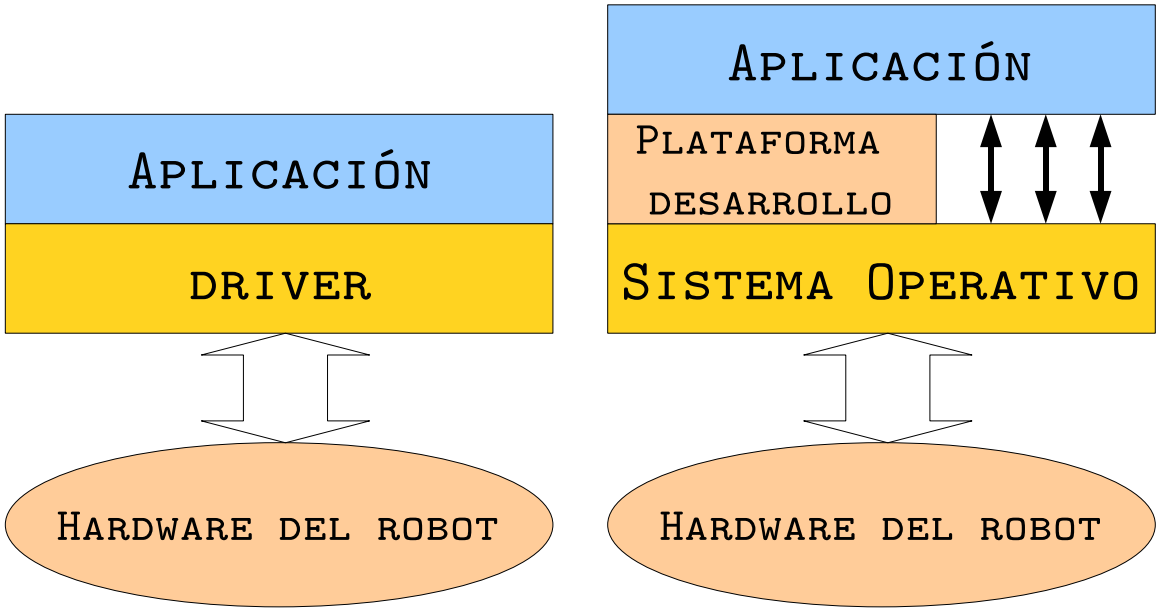
Antiguamente

Los robot eran desarrollos únicos que no se producían en serie → uso directo los *drivers* para acceder a los dispositivos sensoriales y de actuación

Actualmente

Gracias a los fabricantes de robots y el trabajo de muchos grupos de investigación → aparecieron plataformas de desarrollo de robots

- simplifican la programación de aplicaciones robóticas
- incluyen un modelo de programación de software
- permiten manejar la creciente complejidad del código cuando se incrementa la funcionalidad del robot



Los entornos de desarrollo de robots ofrecen

- acceso abstracto y simple a sensores y actuadores
- funcionalidades de uso común como algoritmos de control, localización, navegación segura, construcción de mapas, etc

Algunas plataformas de desarrollo

Hoy en día los fabricantes más avanzados incluyen plataformas de desarrollo

- ActivMedia: `ARIA` para sus robots Pioneer, PeopleBot, etc.
- iRobot: `Mobility` para sus B14 y B21
- Evolution Robotics: `ERSP`
- Sony: `OPEN-R` para sus Aibo

Además de los fabricantes, muchos grupos de investigación han creado sus propias plataformas de desarrollo.

- `CARMEN` de Carnegie Mellon University
- `Orocos`
- `Player/Stage/Gazebo`
- `JDE`

Análisis de RDES

El trabajo realizado por James Krammer y Scheutz [1] analiza 9 *RDEs* de código abierto y realiza una evaluación y comparación objetiva desde varios puntos de vista

- TeamBots
- Advanced Robotics Interface for Applications (ARIA)
- Player/Stage
- Python Robotics (Pyro)
- Carnegie Mellon Robot Navigation Toolkit (CARMEN)
- MissionLab
- APOC Development Environment (ADE)
- Middleware for Robots (Miro)
- Mobile and Autonomous Robotics Integration Environment (MARIE)

Alta puntuación $\rightarrow$ `Player/Stage`

[1] *"Development Environments for Autonomous Mobile Robots: A Suervey"*, James Kramer and Mtthias Scheutz, Artificial Intelligence and Robotics Laboratory, University of Notre Dame, 2007

Contenido

Player/Stage/Gazebo

Tres piezas de software originalmente desarrollado en el laboratorio de investigación de robótica de la *University of Southern California* (USC, Robotics Research Lab) por Brian P. Gerkey y Richard T. Vaughan. Ahora es un proyecto activo de sourceforge.net [2] usado por un gran número de investigadores alrededor del mundo.

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

Player

Servidor de dispositivos utilizados en robótica

- Basado en sockets lo 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

Consta de dos partes

- Servidor de red para el control de robots y corre abordo del robot. Hace de HAL (hardware abstraction layer) para dispositivos robóticos
- Librerías clientes que brindan acceso a los dispositivos remotos. El proyecto oficial cuenta con librerías clientes en C, C++ y Python; además de otras creadas por terceros como Java, MATLAB, GNUOctave, etc

Player, ventaja de los sockets

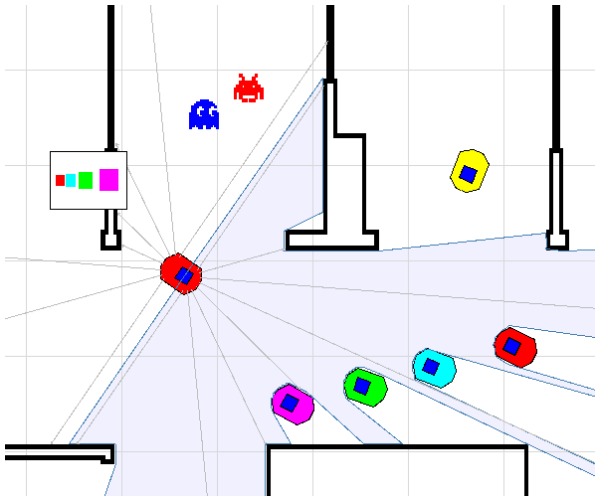
- Distribución: los programas clientes tienen acceso a los sensores y actuadores en cualquier lugar sobre la red
- Independencia: los programas clientes se pueden escribir en cualquier lenguaje de programación lo que da flexibilidad en la plataforma de desarrollo
- Abstracción: el servidor proporciona interfaces de abstracción unificada de los dispositivos conectados al mismo

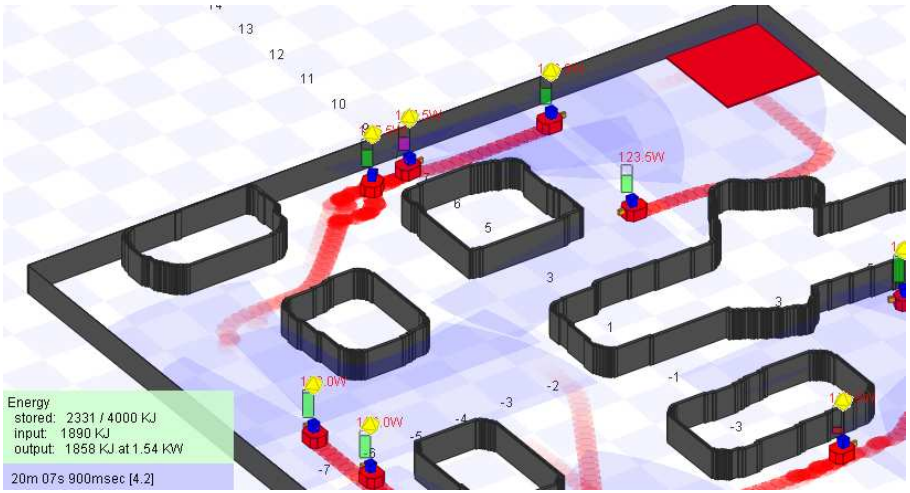
Stage

Simulador de múltiples (cientos) robots que simula una población de robots, sensores y objetos en un entorno bitmap 2D.

Dispone de robots virtuales y de varios modelos de sensores incluyendo sonares, sensores láser rangefinder, cámaras pan-tilt-zoom con detección de blobs de color y odometria.

Simulación 2.5D a partir de la versión 3.0

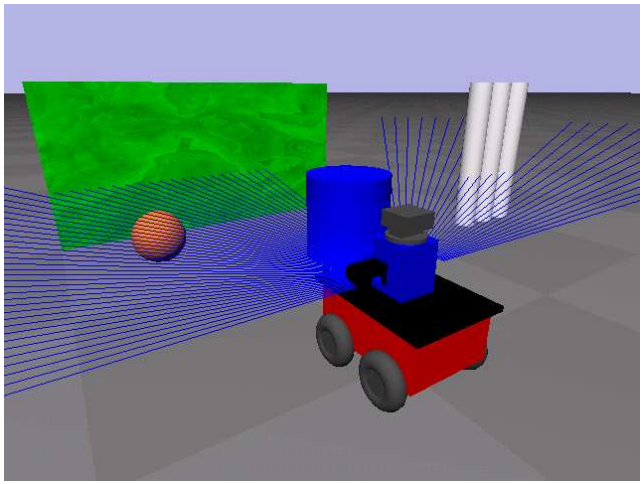


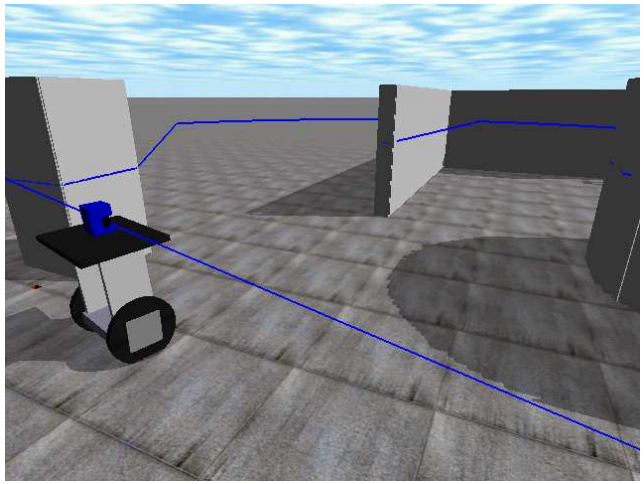


Gazebo

Simulador de múltiples robots en entornos indoor y outdoor, simula una población de robots en un entorno 3D.

Genera realimentación realista de sensores incluyendo simulación precisa de la física de cuerpos rígidos con su dinámica y detección de colisiones.



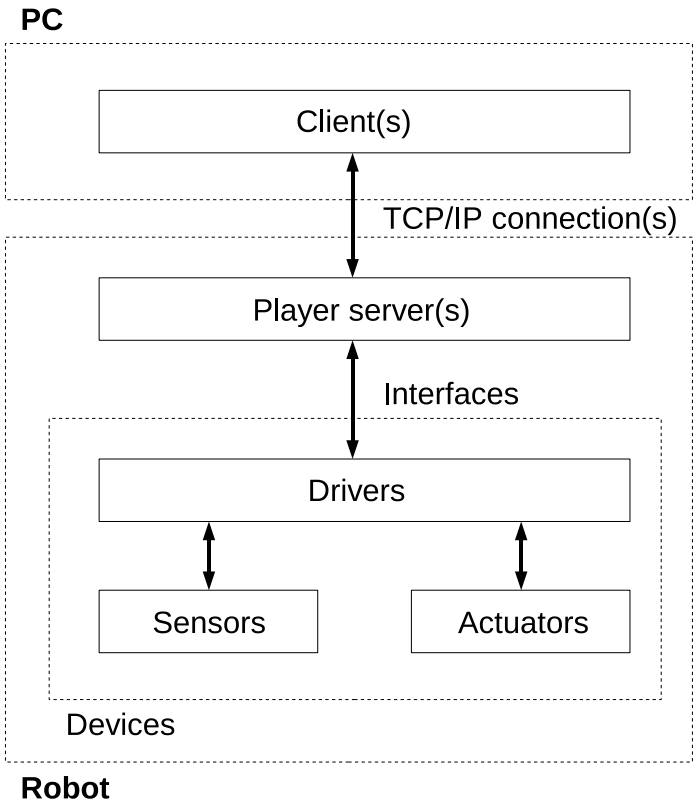


Contenido

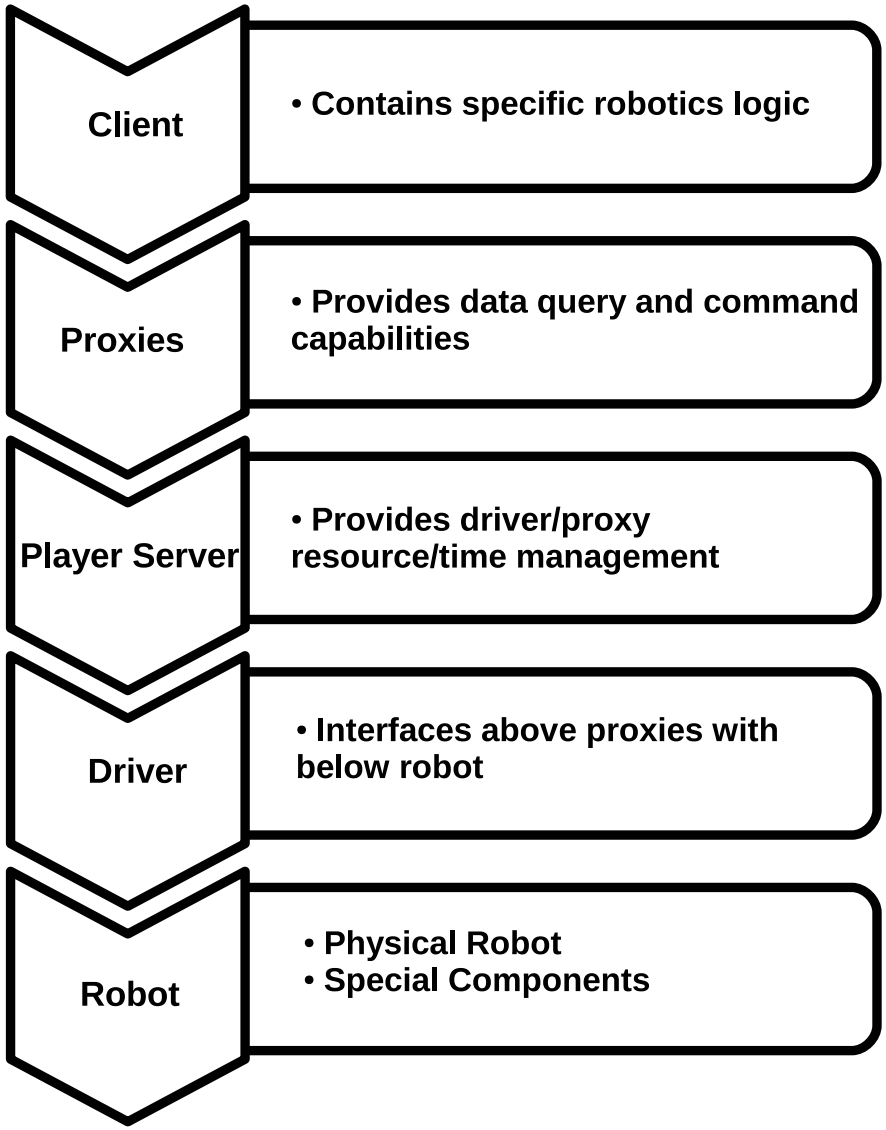
Interfaz, Drivers y Dispositivos

Tres conceptos claves de *Player*

- **Interfaz** Especifica como interactuar con cierta clase de sensores, actuadores o algoritmos
- **Drivers** Pieza de software que traduce las entradas y salidas de los sensores, actuadores o algoritmos para adaptarse a una *interfaz*
- **Dispositivo** Entidad sobre la cual interactúan los mensajes a través de las *interfaces*



Niveles de abstracción de Player



Interfaces incluidas en Player

- **position2d** Planar mobile robot
- **laser** Laser range-finder
- **camera** Camera imagery
- **ranger** A range sensor
- **actarray** An array of actuators
- **gripper** Gripper interface
- **localize** Multi-hypothesis planar localization system
- **planner** A planar path-planner
- **blobfinder** A visual blob-detection system
- **fiducial** Fiducial (marker) detection
- **joystick** Joystick control
- **ptz** Pan-tilt-zoom unit
- **log** Log read/write control
- **imu** Inertial measurement unit
- **gps** Global positioning system
- **map** Access maps

Dispositivos incluidos en Player

- **Robots**
  - Acroname Garcia
  - iRobot Roomba
  - Segway
  - MobileRobots (ActivMedia) Pioneer
- **Hardware**
  - SICK LMS200 laser
  - Hokuyo URG laser
  - Sony EVID30/EVID100 pan-tilt-zoom camera
  - CMUcam2
  - IEEE1394 (Firewire) cameras
  - Camera supported by Video4Linux
  - DirectedPerception PTU-D46 pan-tilt unit
- **Algorithms**
  - Vector Field Histogram (VFH+), goal-seeking obstacle avoidance algorithm (Ulrich & Borenstein)
  - Adaptive Monte Carlo Localization (AMCL) (Fox)
  - Wavefront propagation planner (Latombe)

Servidor Player - Drivers y archivos de configuración

Como se mencionó Player es una capa de abstracción de hardware que conecta el código de la aplicación con el hardware del robot funcionando como una aplicación Cliente/Servidor. Los archivos de configuración le indican al servidor Player cuales drivers utilizar y que interfaces usan estos drivers

```
$> player arch_conf.cfg
```

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

Objetivo del driver

Crear un vínculo entre los dispositivos/algoritmos y la interfaz predefinida de *Player* que mejor lo representa

Tipos de drivers

- driver normal
- driver plugin

Otros parámetros pueden ser `requires` y `plugin`

Servidor Player - Dirección de dispositivo

Le indica a Player donde presentar o recibir la información del driver y que interfaz se utiliza para acceder a dicha información.
Se especifica mediante una cadena de la forma

`key:host:robot:interface:index`

- `key`: permite soportar múltiples interfaces del mismo tipo desde diferentes dispositivos
- `host`: dirección de la computadora host donde se encuentra el dispositivo (dirección IP)
- `robot`: puerto TCP en el que Player espera recibir datos desde una interfaz
- `interface`: interfaz utilizada para interactuar con los datos
- `index`: permite múltiples dispositivos del mismo tipo (misma interfaz), por ejemplo dos cámaras podrían ser `camera:0` y `camera:1`. Es diferente al campo `key` puesto que "tener un driver que soporta varias interfaces del mismo tipo" NO es lo mismo que "tener múltiples dispositivos que usan la misma interfaz"

Servidor Player - Ejemplos de archivos de configuración

```
driver
(
  name      "p2os"
  provides [ "odometry:::position2d:0"
             "compass:::position2d:1"
             "gyro:::position2d:2"
             "sonar:0"
             "aio:0"
             "dio:0"
             "power:0"
             "bumper:0"
             "gripper:::gripper:0"
             "blobfinder:0"
             "sound:0" ]
  port     "/dev/ttyS0"
)
```

```
driver
(
  name      "sicklms200"
  provides [ "laser:0" ]
  port      "/dev/ttyS2"
)
```

```
driver
(
  name       "linuxjoystick"
  provides   [ "joystick:0" ]
  requires   [ "position2d:0" ]
  max_speed  [1 0 30]
  axes       [1 -1 2]
  port       "/dev/input/js0"
  alwayson   1
)
```

```
driver
(
  name      "urglaser"
  provides ["laser:0"]
  port      "/dev/ttyS1"
  min_angle  -115.0
  max_angle  115.0
  baud       115200
)
```

Servidor Player - más ejemplos

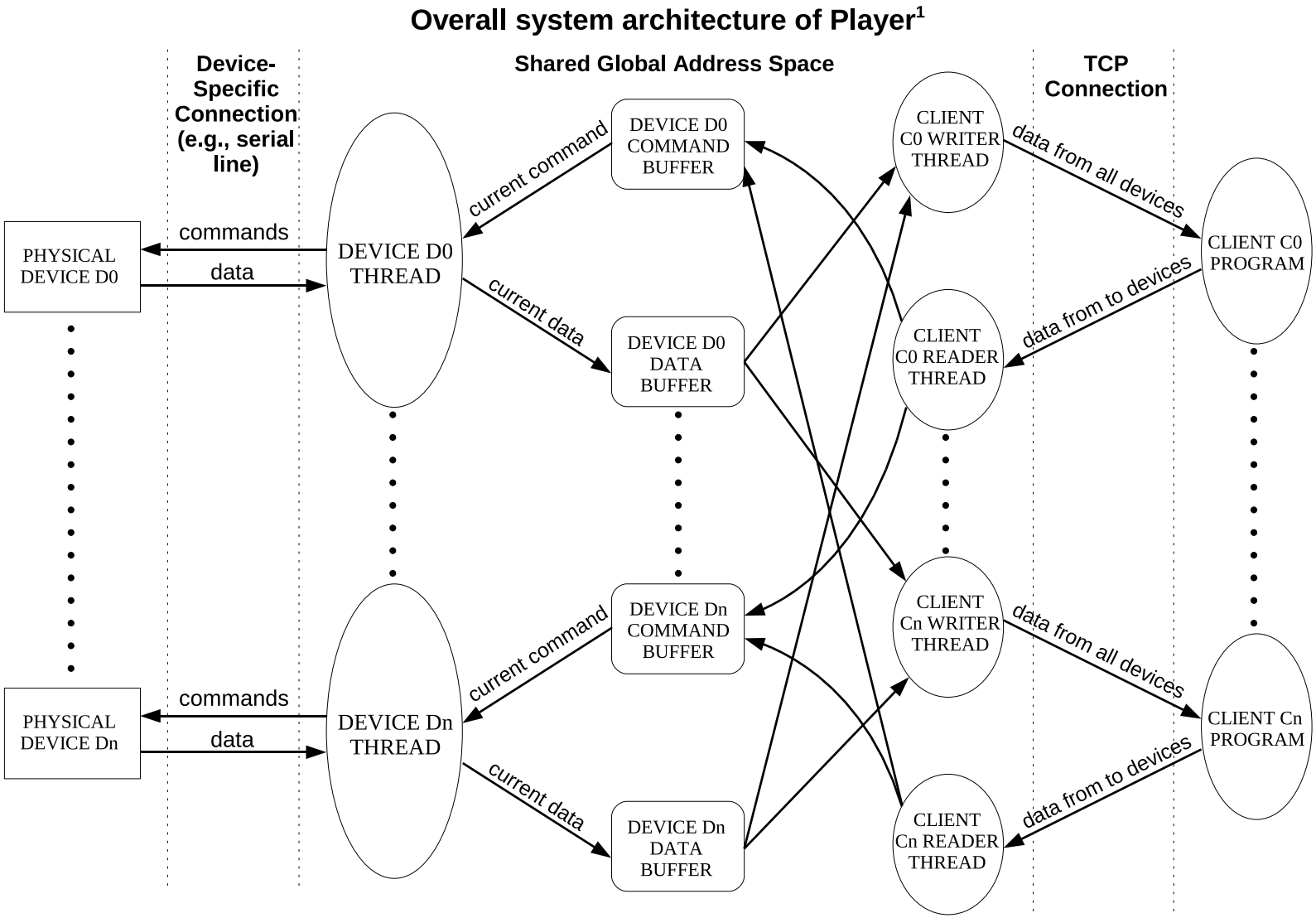
```
driver
(
  name          "romaadriver"
  plugin        "libromaadriver"
  provides      [ "position2d:0" ]
  port          "/dev/ttyUSB0"
  baudrate      38400
  motor_pid_kp 7
  motor_pid_ki 1
  motor_pid_kd 0
  vw_pid_ki     1
  vmultiplier   100
  wmultiplier   100
)
```

```
driver
(
  name        "camera1394"
  framerate   50
  alwayson    1
  provides    [ "camera:0" ]
)
```

```
driver
(
  name                  "writelog"
  log_directory         "/home/myuser/logs"
  timestamp_directory   1
  basename              "romaalaser''
  requires              [ "laser:0"
                          "position2d:0" ]
  provides              [ "log:0" ]
  alwayson              0
  autorecord            0
)
```

```
driver
(
 name        "readlog"
  filename    "logfile.log"
  provides    [ "position2d:0"
                "laser:0"
                "log:0" ]
  speed       2.0
)
```

Servidor Player - Estructura



Cliente Player - Librerías

Disponibles por el proyecto

- `libplayerc` Librerías clientes C
- `libplayer_py` Librerías clientes python
- `libplayerc++` Librerías clientes C++

Contribuciones

Para lenguajes como `MATLAB`, `Smalltalk`, `Java`, `GNUOctave`, etc.

Los programas clientes utilizan objetos proxy definidos en las librerías clientes para leer/escribir datos desde/hacia los dispositivos.

Algunos objetos Proxies definidos en C++ son: `Position2dProxy`, `LaserProxy`, `BlobfinderProxy`, `LocalizeProxy`, `MapProxy`, `PtzProxy`, etc

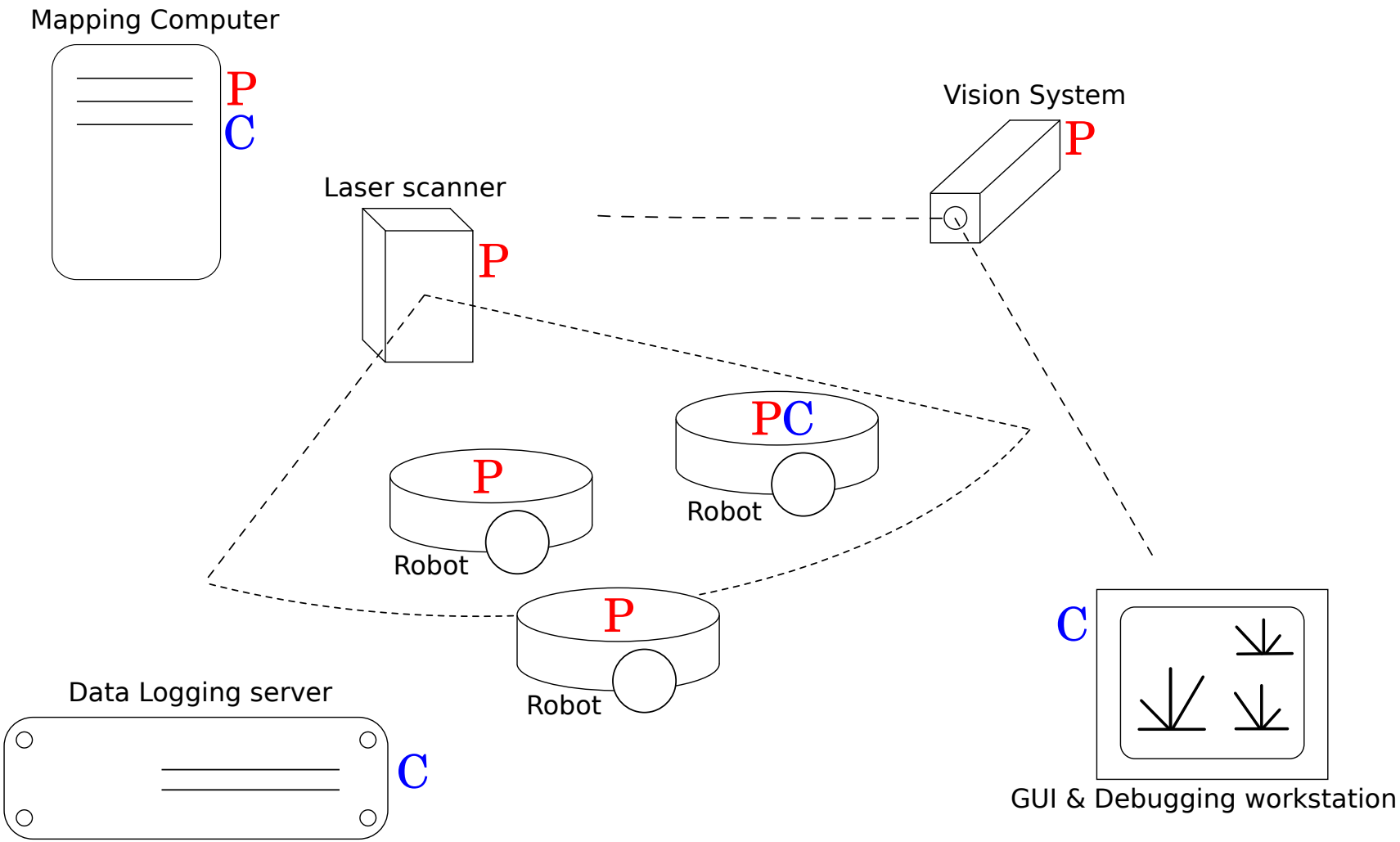
Cliente Player - Programa ejemplo 1

```
#include <iostream>
#include <libplayerc++/playerc++.h>

int main( int argc, char* argv[] )
{
  // Connect to the local player process on port 6665
  PlayerCc::PlayerClient romaa_robot( "localhost", 6665 );
  // Create a position2d proxy
  PlayerCc::Position2dProxy romaa_pos2d( &romaa_robot, 0 );

  romaa_pos2d.ResetOdometry( );
  romaa_pos2d.SetSpeed( 1, 0 );

  for( int i = 0; i < 200; i++ )
  {
   romaa_robot.Read( );

   std::cout << "i: " << i << " ---> ";
   std::cout << "px: " << romaa_pos2d.GetXPos() << " - ";
   std::cout << "py: " << romaa_pos2d.GetYPos() << " - ";
   std::cout << "pa: " << romaa_pos2d.GetYaw() << std::endl;

   usleep(50000);
  }
  romaa_pos2d.SetSpeed( 0, 0 );
  return 0;
 }
```

Cliente Player - Programa ejemplo 2

```
#include <iostream>
#include <libplayerc++/playerc++.h>

int main( int argc, char* argv[] )
{
  // Connect to the local player process on port 6665
  PlayerCc::PlayerClient romaa_robot( "localhost", 6665 );
  // Create a laser proxy
  PlayerCc::LaserProxy romaa_laser( &romaa_robot, 0 );

  romaa_robot.Read( );

  std::cout << "Laser data..." << std::endl;
  for( int i = 0; i < romaa_laser.GetCount( ); i++ )
  {
    std::cout << romaa_laser.GetRange( i ) << " ";
  }
  std::cout << std::endl;

  std::cout << "Laser bearing..." << std::endl;
  for( int i = 0; i < romaa_laser.GetCount( ); i++ )
  {
    std::cout << romaa_laser.GetBearing( i ) << " ";
  }
  std::cout << std::endl;
  return 0;
}
```

Escenario de ejemplo

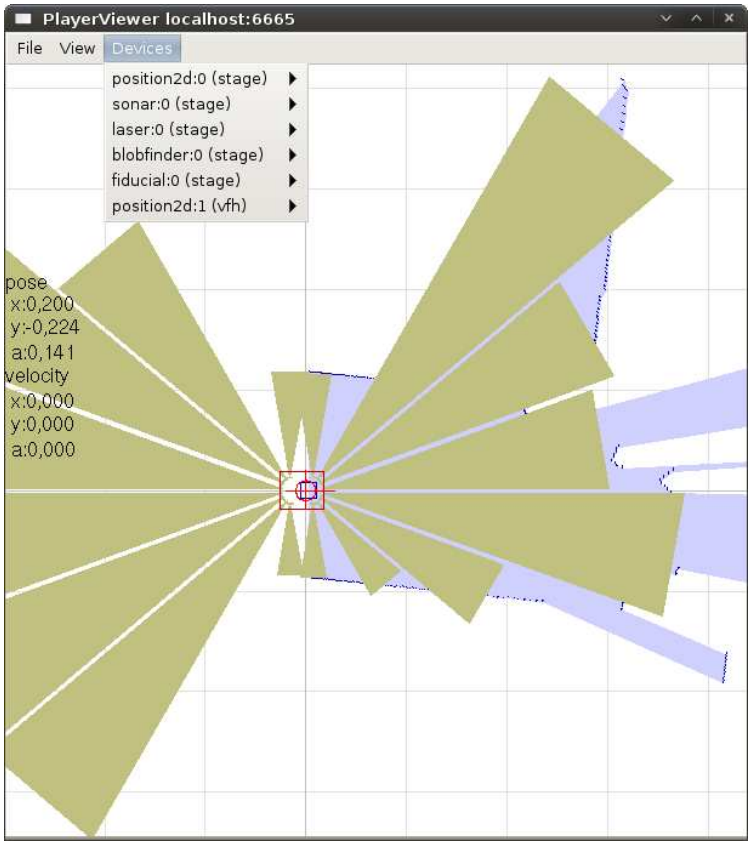


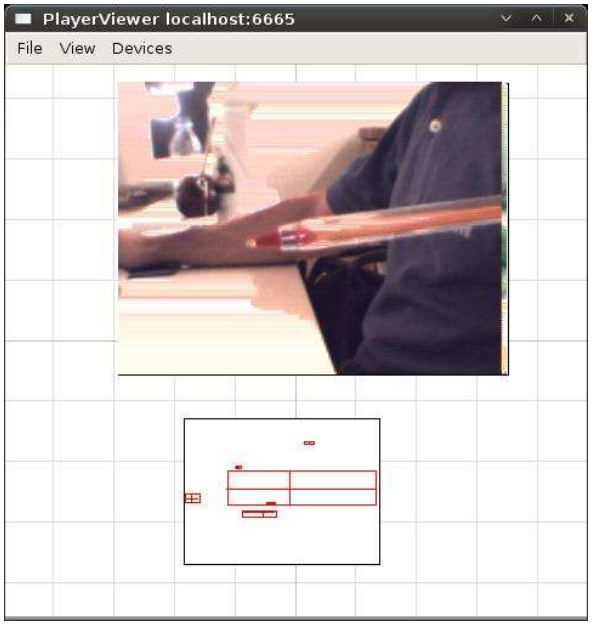
"Most Valuable Player: A Robot Device Server for Distributed Control". Brian P. Gerkey, Richard T. Vaughan, Kasper Støy, Andrew Howard, Gaurav S Sukhatme, and Maja J Mataric´. In Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS 2001), pages 1226-1231, Wailea,

Herramientas

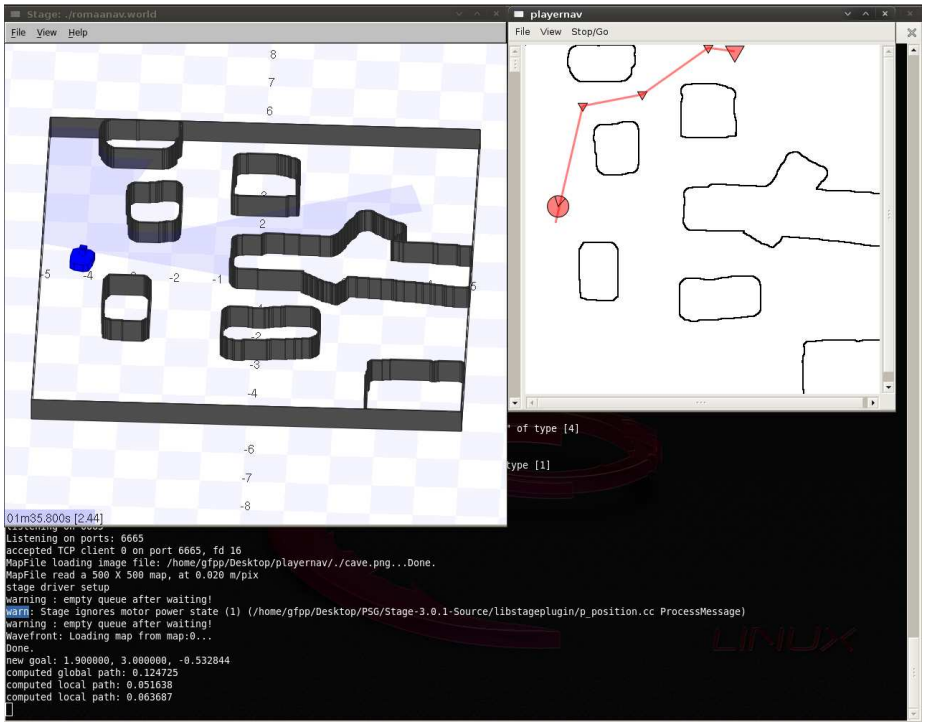
- **playerprint** Solicita e imprime datos de sensores a consola
- **playerv** Solicita y muestra gráficamente datos de sensores; también permite la teleoperación por medio de movimiento del mouse
- **playerjoy** Teleoperación por medio de joystick
- **playervcr** Control remoto de logging de datos y playback
- **playernav** Unidad de control gráfica que provee de control sobre múltiples robots para localización y planificación de trayectoria (path-planning)
- **playerwritemap** Solicita mapa de grilla y vector (por ej. para un driver de SLAM) y lo escribe a disco
- **playercam** Muestra imágen de video remotamente desde cámaras montadas en robots

Herramientas





Herramientas



Contenido

Player con Stage

Las aplicaciones robóticas desarrolladas mediante programas clientes de Player se puede simular mediante Stage simplemente ejecutando el servidor Player con un archivo de configuración que en lugar de cargar driver de dispositivos reales, ejecute el simulador

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

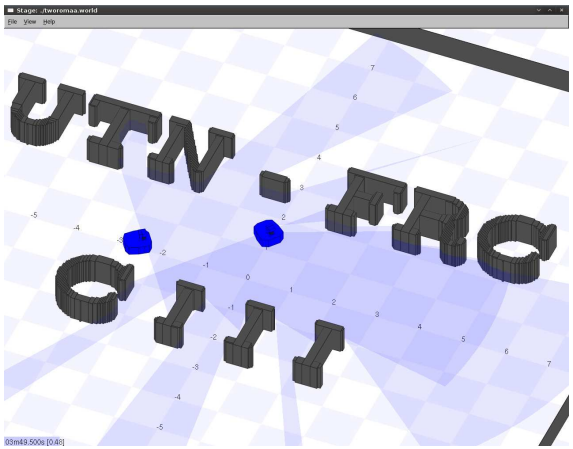
Player con Stage - Ejemplo

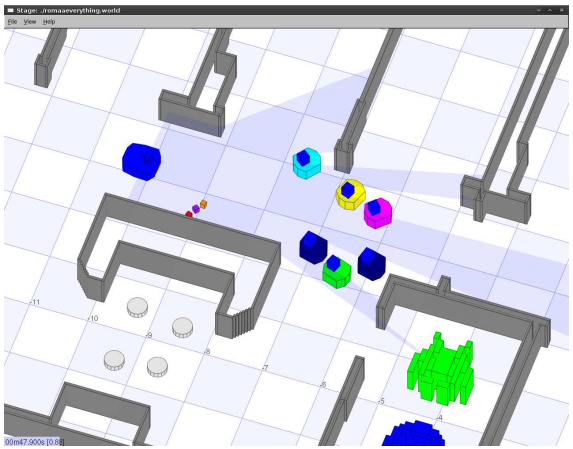
Ejecutar

```
$>player romaalasersick200.cfg
```

Archivo .cfg

```
# load the Stage plugin simulation driver
driver
(
name "stage"
provides [ "simulation:0" ]
plugin "libstageplugin"

# load the named file into the simulator
worldfile "romaalasersick200.world"
)

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



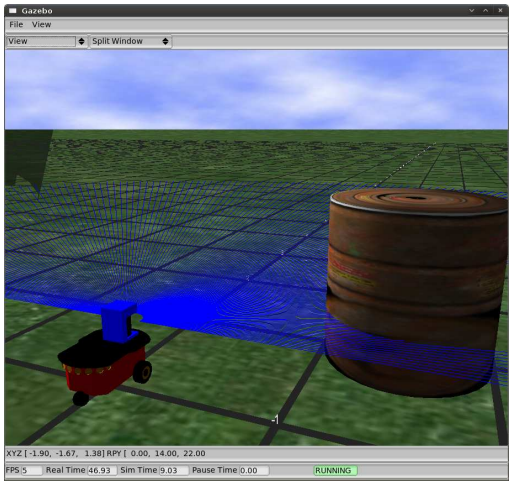


Player con Gazebo

Gazebo toma la descripción del entorno de archivos `.world`, donde además se describe el terreno, la fuente de luz, la cámara de observación, etc.

Ejecutar

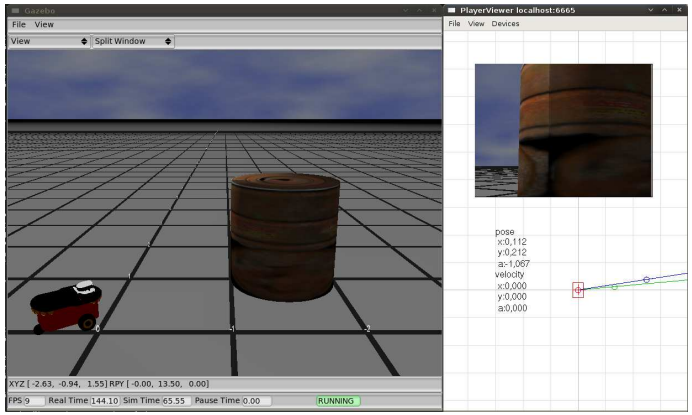
```
$> gazebo worldfile.world
```



Y en otra terminal

```
$> player cfgfile.cfg
```

donde se indican las interfaces de los controladores instanciados en el archivo `.world`.



Referencias

Geoffrey Biggs and Bruce Macdonald.
A survey of robot programming systems.
In *in Proceedings of the Australasian Conference on Robotics and Automation, CSIRO*, page 27, 2003.

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.

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.

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.

Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard.
The player/stage project: Tools for multi-robot and distributed sensor systems.
In *In Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, 2003.

Nick Wong, Jui-Chun Peng Hsu, Toby H.J. Collet, and Bruce A. MacDonald.
Improving the 2.5d stage robotic simulator.
2008.