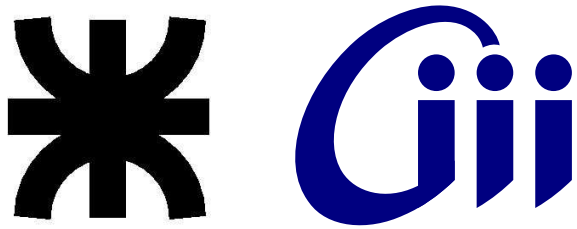
Entorno de desarrollo Player/Stage aplicado a la robótica móvil

Gonzalo F. Perez Paina David A. Gaydou

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



XXII Congreso Argentino de Control Automático
Septiembre de 2010

Contenido

Contenido

El robot móvil de arquitectura abierta, RoMAA

Robot de tracción diferencial para investigación en

- robótica móvil
- visión por computadoras

Desarrollo en dos etapas

- Mecánica y electrónica embebida
- Librerías y herramientas de programación y simulación

Elección de Player/Stage

El Krammer y Scheutz [1] se analizan 9 *RDEs* de código abierto

- Advanced Robotics Interface for Applications (ARIA)
- Player/Stage
- Python Robotics (Pyro)
- Carnegie Mellon Robot Navigation Toolkit (CARMEN)

Alta puntuación $\rightarrow$ `Player/Stage`

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

Contenido

Player/Stage

Originalmente

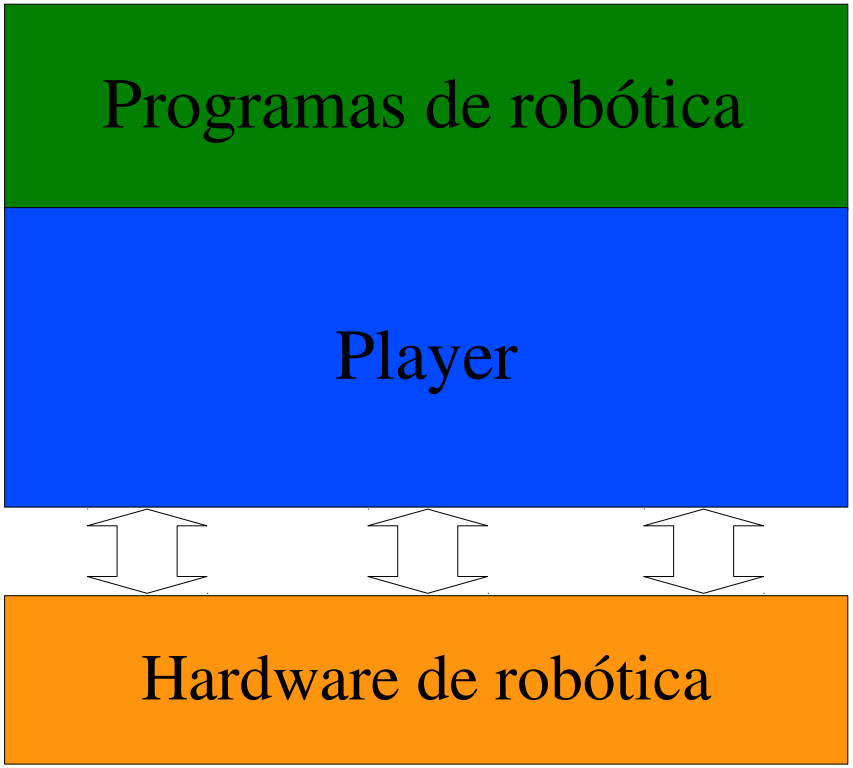
Desarrollado en el laboratorio de investigación de robótica de la *University of Southern California* (USC, Robotics Research Lab) por Brian P. Gerkey y Richard T. Vaughan.

En la actualidad

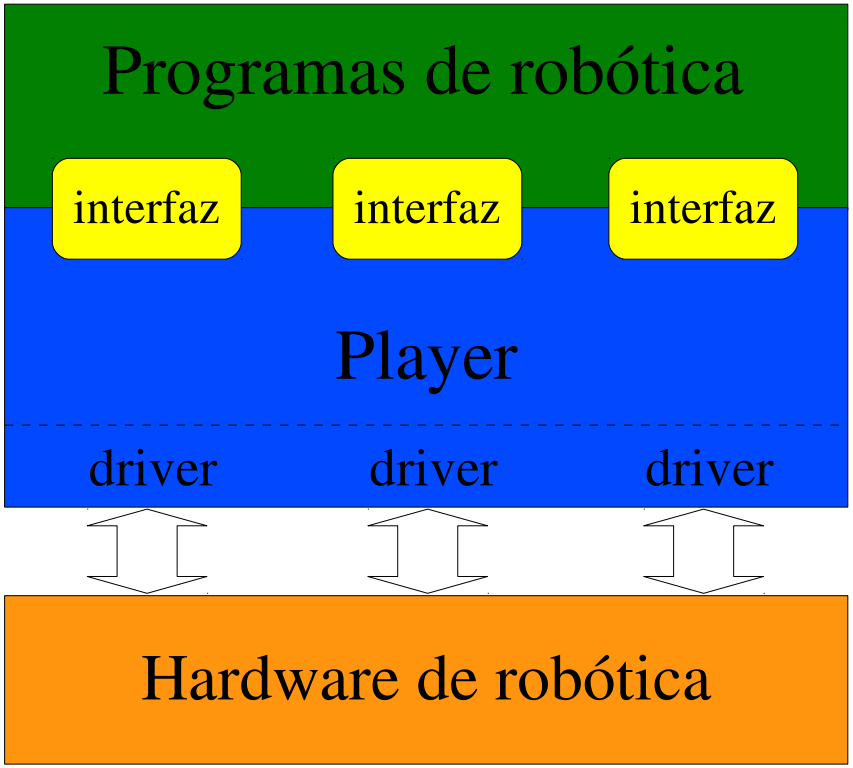
Es un proyecto activo de `sourceforge.net` usado por un gran número de investigadores alrededor del mundo

`http://playerstage.sourceforge.net/`

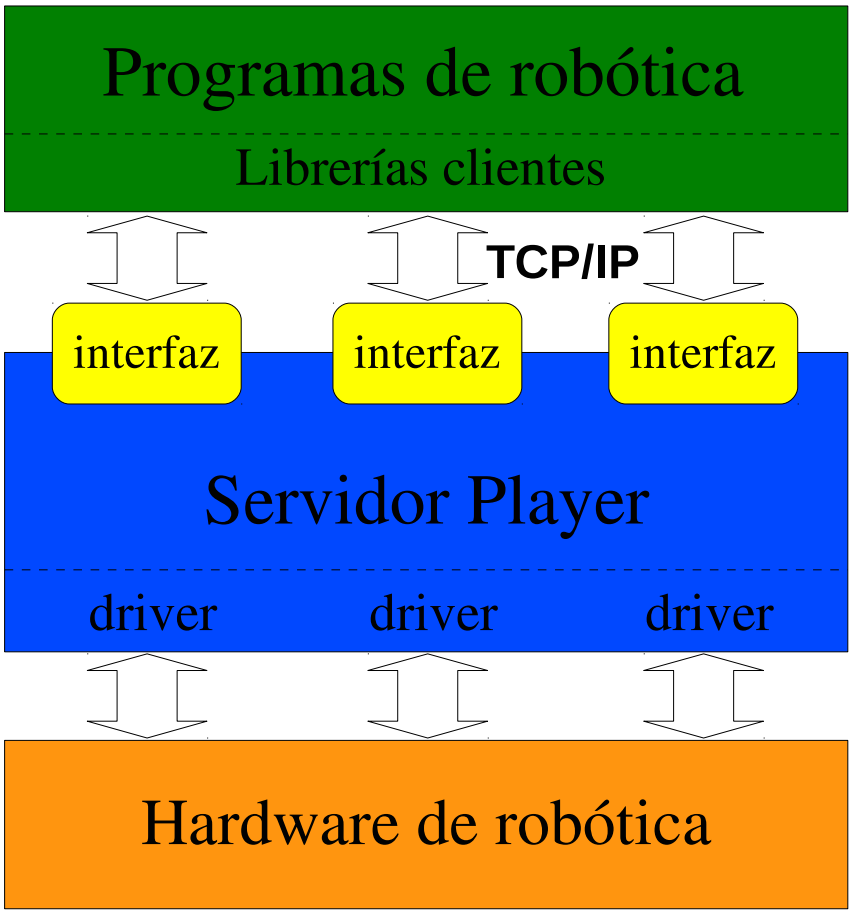
Player/Stage, descripción general



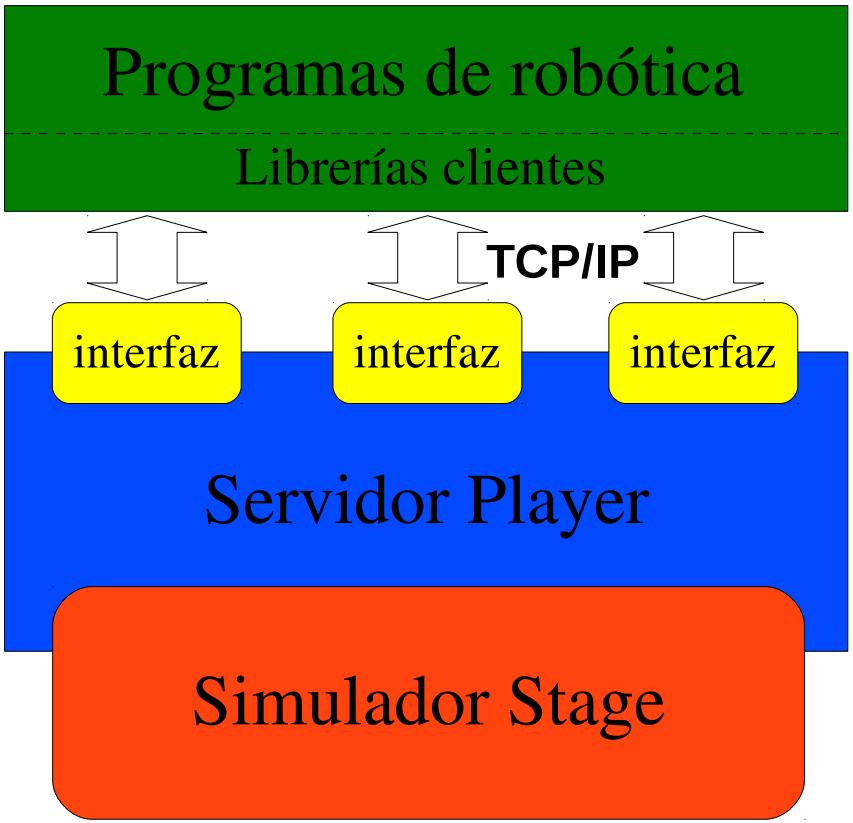
Player/Stage, descripción general



Player/Stage, descripción general



Player/Stage, descripción general



Interfaces definidas en Player

- **position2d** Planar mobile robot
- **laser** Laser range-finder
- **camera** Camera imagery
- **ptz** Pan-tilt-zoom unit
- **imu** Inertial measurement unit
- **gps** Global positioning system

- **localize** Planar localization system
- **planner** A planar path-planner
- **map** Access maps
- **log** Log read/write control

Servidor Player - Archivo de configuración

Ejecutar el servidor Player

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

Independientemente si es sobre hardware real o simulado por Stage.

Sección `driver`

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

Tipos de drivers

- driver normal
- driver plugin

Dirección de dispositivo `host:robot:interface:index`

Otros parámetros pueden ser `requires` y `plugin`

Librerías clientes

Disponibles por el proyecto

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

Contribuciones

Para lenguajes como `MATLAB`, `Java`, `GNU/Octave`, `Smalltalk`, etc.

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

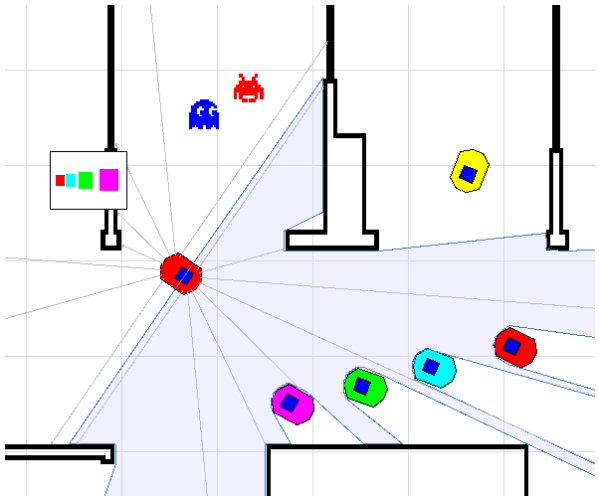
Algunos ejemplos en C++ son: `Position2dProxy`, `LaserProxy`, `LocalizeProxy`, `MapProxy`, `PtzProxy`, etc

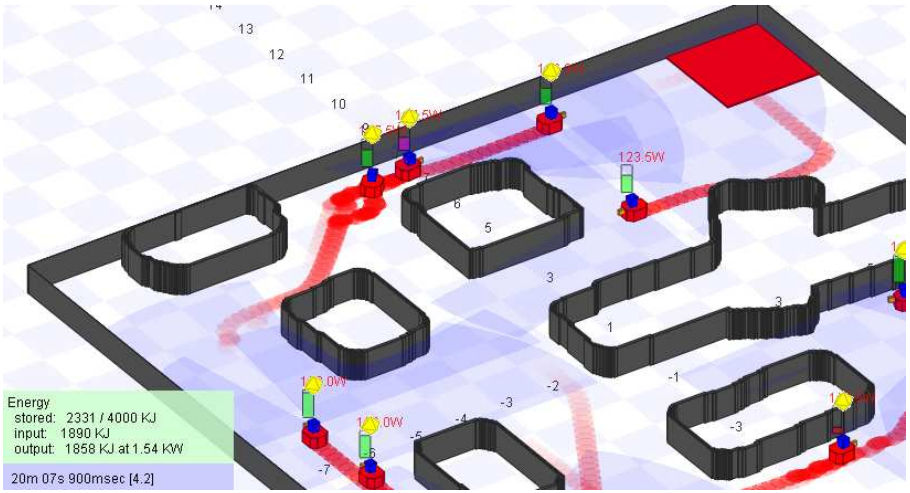
Simulador Stage

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

Simulación 2.5D a partir de la versión 3.0

Dispone de robots virtuales y de varios modelos de sensores incluyendo sonares, sensores láser rangefinder, odometria, etc.





Contenido

Driver del robot RoMAA para Player

Programación del driver

Se programa en C++ utilizando las librerías provistas por Player (`playercore`). Se hereda una clase de `ThreadedDriver`.

Generación del driver

La compilación del driver genera una librería dinámica `libromaa.so`

Archivo de configuración

Para cargar el driver y definir la interfaz que utiliza, además de otros parámetros específicos

```
$>player romaareal.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
)
```

Simulador del robot RoMAA en Stage

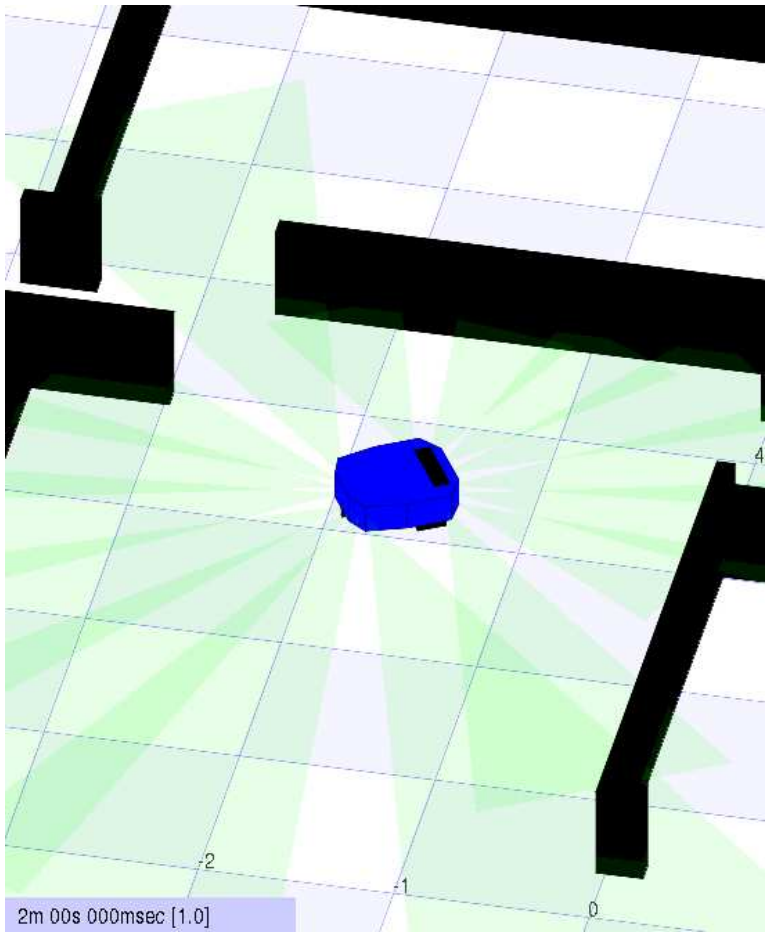
```
$>player romaasim.cfg
```

```
# 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 "lab.world"
)

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

Stage utiliza el archivo `.world`



Descripción del entorno

Archivo .world

```
# lab.world
# Author: Gonzalo F. Perez Paina

include "map.inc"
include "romaa.inc"

interval_sim 100
interval_real 100

paused 0

# size of the whole simulation
size [15 15]

# configure the GUI window
window
(
  size [ 700.0 700.0 ]
  # 700/46 rounded up a bit
  scale 46
  show_data 1
)
```

```
# load an environment bitmap
map
(
  bitmap "lab.png"
  size [15 15 0.5]
)

# robot model
romaa_sonar
(
  name "romaarobot"
  pose [-2 -2 0 90]
)
```

Modelo del simulador Stage

Archivo `romaa.inc`

```
define romaa position
(
  # specify the size of the model
  size [0.570 0.520 0.200]
  # the romaa's center of rotation
  # offset from its center of area
  origin [0.111 0 0 0]
  # main body
  block(
    points 10
    point[0] [ 0.174 0.142 ]
    ...
    z [0.07 0.3]
    color "blue"
  )
  ...
  # differential-steer model
  drive "diff"
  localization "odom"
  #odom_error [ 0.05 0.05 0.1 ]
)
```

```
define romaa_sonar_ring ranger
(
  # number of transducers
  scount 16
  # pose of each transducer
  spose[0] [ 0.075 0.240 90 ]
  ...
  spose[15] [ -0.324 -0.200 -90 ]
  # field of view
  sview [0 5.0 15]
  # size of each transducer
  ssize [0.01 0.05]
)

define romaa_sonar romaa
(
  #use sonar array defined above
  #w/ a small vertical offset to
  #drop sensors into the robot body
  romaa_sonar_ring(
    pose [0 0 -0.1 0])
)
```

Contenido

Flexibilidad de Player. Ejemplos

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

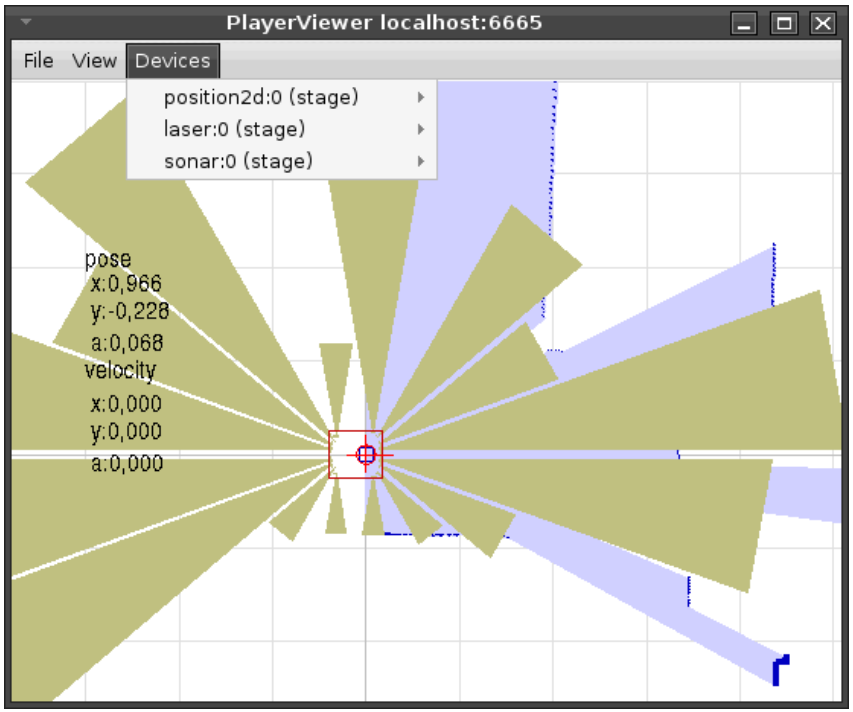
```
# Config file. Laser sick lms200
driver
(
  name            "sicklms200"
  provides        [ "laser:0" ]
  port            "/dev/ttyUSB1"
  resolution      100
  transfer_rate   38400
  alwayson        1
  delay           35
)
```

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

```
# 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
)
```

Flexibilidad de Player. Herramientas

- **playerprint** imprime datos de sensores
- **playerv** muestra datos gráficamente
- **playerjoy** teleoperación
- **playervcr** control de logging
- **playercam** muestra imagen de video



Conclusiones

- Se presentó la forma de agregar un nuevo hardware no soportado originalmente por Player como el robot RoMAA. De forma similar se agrega cualquier otro hardware
- Y como generar el modelo de simulación para Stage del robot RoMAA
- Se mostro la flexibilidad en la utilización de Player/Stage
- Permite ciclos de desarrollo, simulación y experimentación más ágiles en un campo de constante evolución como la robótica móvil

Gracias!