Player/Stage
Manual en línea:
http://playerstage.sourceforge.net/doc/Player-1.6.5/player-html/index.php

Introducción.

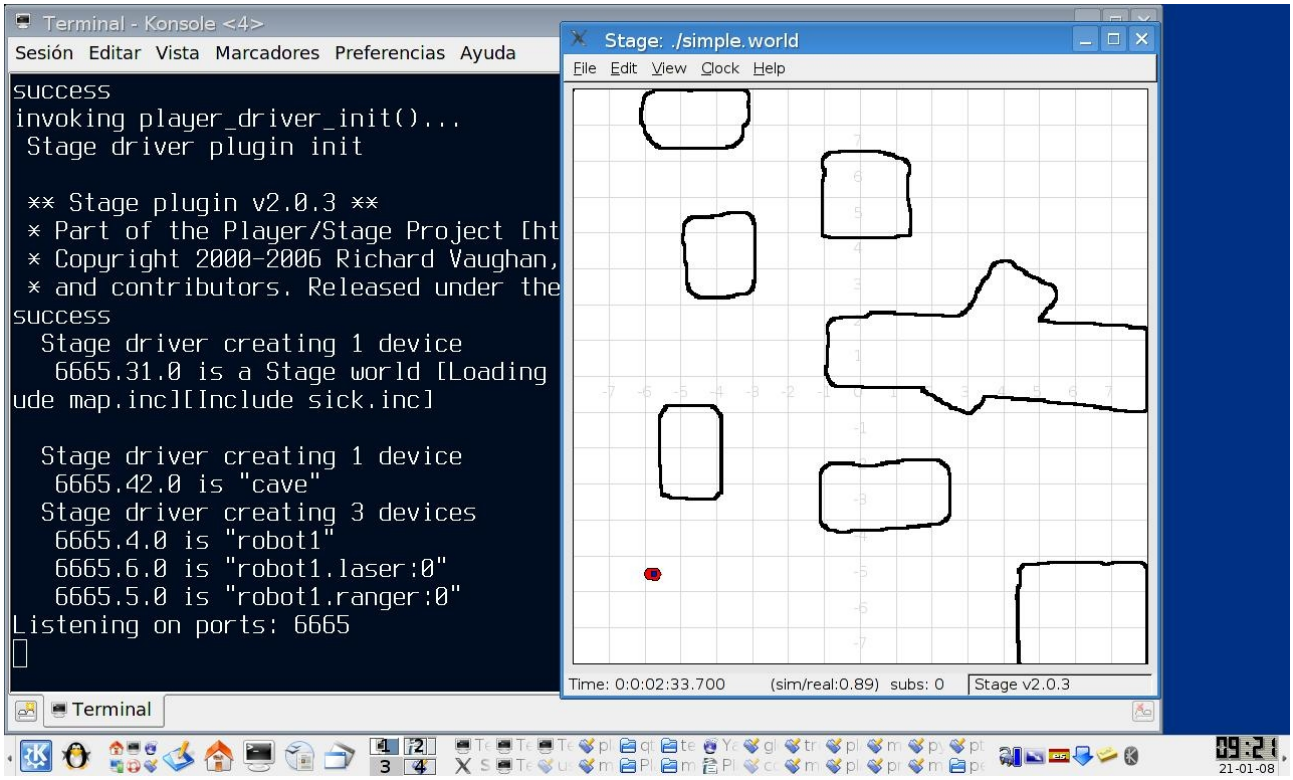
Cada fabricante de robots móviles, los dota de un conjunto de dispositivos sensores y actuadores con sus propios comandos para su operación, por lo que la programación realizada para un determinado robot, debe reescribirse para ser empleada en un robot de otra marca.

Player/Stage es una plataforma general para la programación de robots móviles. Consta de dos productos: El Player (la interfaz) y el Stage (el simulador). Su plataforma original fué la familia ActivMedia Pioneer 2.

Player es una interfaz de los dispositivos de un robot. En vez de manipular directamente a los dispositivos, Player proporciona una serie de comandos generales que facilitan trabajar con diferentes marcas de robots y dispositivos y Stage es el simulador donde podemos realizar pruebas virtuales que posteriormente podemos ejecutar directamente sobre el robot real.

Los programas desarrollados en Player, pueden ejecutarse sobre el simulador o sobre el propio robot.

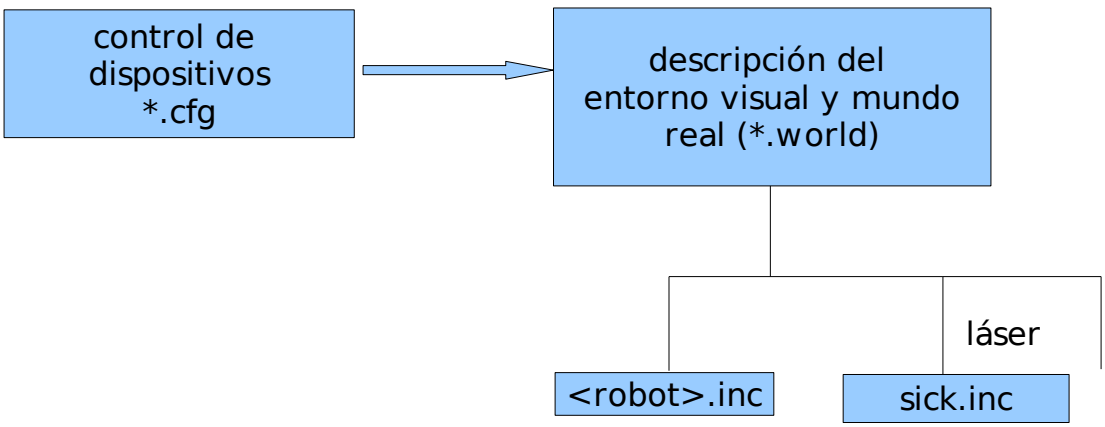
El simulador Stage.



Stage es un simulador para robots móviles. Se emplea como un plugin con Player o como una librería de C. En este último caso, ya no se usarían las interfaces del Player, sino a través de nuestros propios programas en C.

Para instalar el simulador Stage como plugin de Player, se utilizan dos archivos:

- un archivo de configuración, con extensión “cfg”, que contiene la descripción para el control de los dispositivos. Por ejemplo, si se emplea una cámara, contiene el nombre del “driver” y del puerto de entrada empleado.
- otro archivo, (que se referencía en el de configuración) contiene la descripción del mundo (entorno) donde va a desplazarse el robot, el tamaño de la ventana visual en pixeles. También se incluyen las referencias a los archivos de descripción de los dispositivos empleados (incluyendo al propio robot).



Un ejemplo de archivo de configuración, lo encontramos en la instalación del Stage:

/usr/local/share/stage/worlds/simple.cfg

```
# Desc: Player sample configuration file for controlling Stage devices
# Author:  Richard Vaughan
# Date: 1 December 2004
# CVS: $Id: simple.cfg,v 1.30.2.1 2006/07/13 17:59:10 gerkey Exp $


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

  # load the named file into the simulator
  worldfile "simple.world"       # enlaza el archivo simple.world
)

driver
(
  name "stage"
  provides ["map:0"]
  model "cave"
)

# Create a Stage driver and attach position2d and laser interfaces
# to the model "robot1"
driver
(
  name "stage"
  provides ["position2d:0" "laser:0" "sonar:0"]
  model "robot1"
)

# Demonstrates use of a Player "abstract driver": one that doesn't
# interface directly with hardware, but only with other Player devices.
# The VFH driver attempts to drive to commanded positions without
# bumping into obstacles.
driver
(
  name "vfh"
  provides ["position2d:1"]
  requires ["position2d:0" "laser:0" ]
)
```

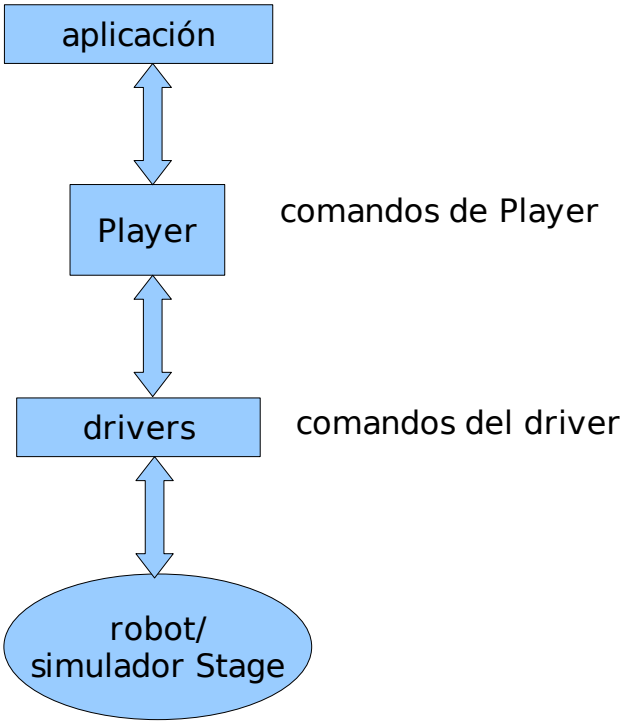
Este enlaza con el archivo /usr/local/share/stage/worlds/simple.world
que incluye las referencias a los archivos de descripción de los robots y sus dispositivos.

```
# Desc: 1 pioneer robot with laser
# CVS: $Id: simple.world,v 1.63 2006/03/22 00:22:44 rtv Exp $

# defines Pioneer-like robots; por ejemplo, sus dimensiones, el número y la posición de los sonares.
include "pioneer.inc"

# defines 'map' object used for floorplans
include "map.inc"

# defines sick laser
include "sick.inc"

# size of the world in meters
size [16 16]

# set the resolution of the underlying raytrace model in meters
resolution 0.02

# update the screen every 10ms (we need fast update for the stest demo)
gui_interval 20

# configure the GUI window
window
(
  size [ 591.000 638.000 ]
  center [-0.010 -0.040]
  scale 0.028
)

# load an environment bitmap
map
(
  bitmap "bitmaps/cave.png"
  size [16 16]
  name "cave"
)

# create a robot
pioneer2dx
(
  name "robot1"
  color "red"
  #pose [-6.5 -6.5 45]
  pose [-5.25 6.2 359]
  sick_laser( samples 361 laser_sample_skip 4 )
)
```

La interfaz de Player

Player proporciona interfaces abstractas al robot y sus dispositivos. Y estas intefaces abstractas a su vez se comunican con los “drivers” del robot y dispositivos.

1) Estructura del player/Stage



Las interfaces del Player son estándar para diferentes “drivers” de diferentes dispositivos. Por ejemplo, la interfaz “**position”** permite manejar comandos de desplazamiento de un robot abstracto. En los archivos de configuración se estable el “driver” que corresponda al robots. Igualmente la interfaz “laser” que puede corresponder a un SICK LMS-200 o algún otro.

El acceso a los dispositivos en Player es a través de sockets TCP (Transmission Control Protocol) que son dispositivos de comunicación de dos vías, lo que permite programar el control de sensores y actuadores en varios robots en paralelo.

Los comandos del player, corresponden a los controles y sensores disponibles. Se presentan en dos librerías disponibles: libplayerc y libplayerc++

Usando la libplayerc.

Un programa básico para mover al robot usando la libplayerc, es /usr/local/share/player/examples/libplayerc/simple.c

```
#include <stdio.h>

#include <libplayerc/playerc.h>

int
main(int argc, const char **argv)
{
  int i;
  playerc_client_t *client;  // la estructura client
  playerc_position2d_t *position2d; // la estructura position2d

  // Create a client and connect it to the server.
  client = playerc_client_create(NULL, "localhost", 6665);
  if (0 != playerc_client_connect(client))
    return -1;

  // Create and subscribe to a position2d device.  // corresponde al ROBOT
  position2d = playerc_position2d_create(client, 0);
  if (playerc_position2d_subscribe(position2d, PLAYER_OPEN_MODE))
    return -1;

  // Make the robot spin!  // pone al ROBOT a girar
  if (0 != playerc_position2d_set_cmd_vel(position2d, 0, 0, DTOR(40.0), 1))
    return -1;

  for (i = 0; i < 200; i++)
  {
    // Wait for new data from server
    playerc_client_read(client);

    // Print current robot pose  // ejes X, Y y ángulo en radianes
    printf("position2d : %f %f %f\n",
           position2d->px, position2d->py, position2d->pa);
  }

  // Shutdown
  playerc_position2d_unsubscribe(position2d);
  playerc_position2d_destroy(position2d);
  playerc_client_disconnect(client);
  playerc_client_destroy(client);

  return 0;
}
```

Para compilar:
gcc -o simple `pkg-config --cflags playerc` simple.c `pkg-config --libs playerc`

Usando la libplayerc++.

Un programa básico para mover al robot usando la libplayerc++, es /usr/local/share/player/examples/libplayerc++/example0.cc

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

int main(int argc, char *argv[])
{
  using namespace PlayerCc;

  PlayerClient    robot("localhost");
  SonarProxy      sp(&robot,0);
  Position2dProxy pp(&robot,0);

  for(;;)
  {
    double turnrate, speed;


    // read from the proxies
    robot.Read();

    // print out sonars for fun
    std::cout << sp << std::endl;

    // do simple collision avoidance
    if((sp[0] + sp[1]) < (sp[6] + sp[7]))
      turnrate = dtor(-20); // turn 20 degrees per second
    else
      turnrate = dtor(20);

    if(sp[3] < 0.500)
      speed = 0;
    else
      speed = 0.100;

    // command the motors
    pp.SetSpeed(speed, turnrate);
  }
}
```

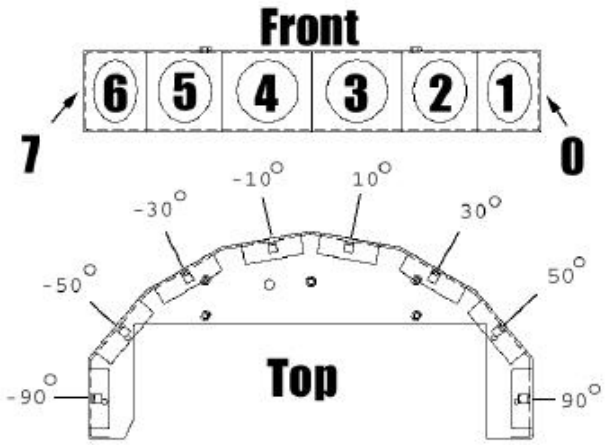


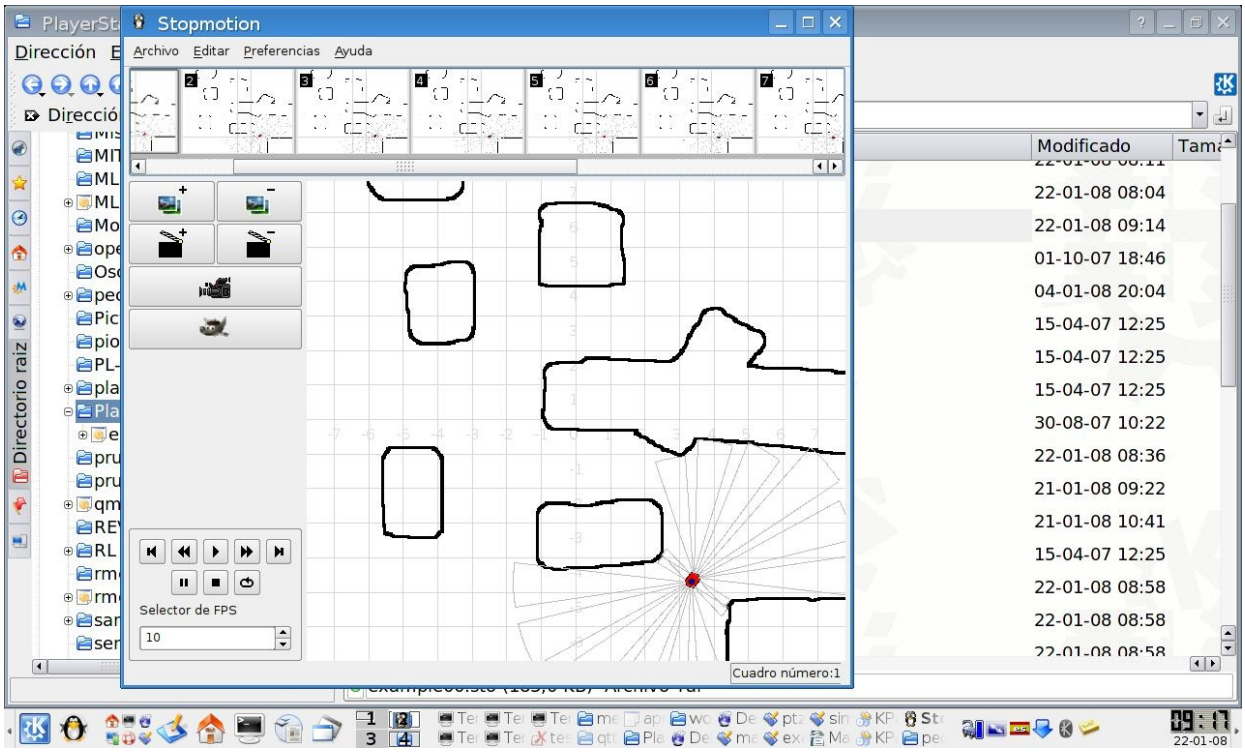
Figure 12. ActivMedia robot sonar array

Compilación:

```
gcc -o example0 `pkg-config --cflags playerc++` example0.cc `pkg-config --libs playerc++`
```

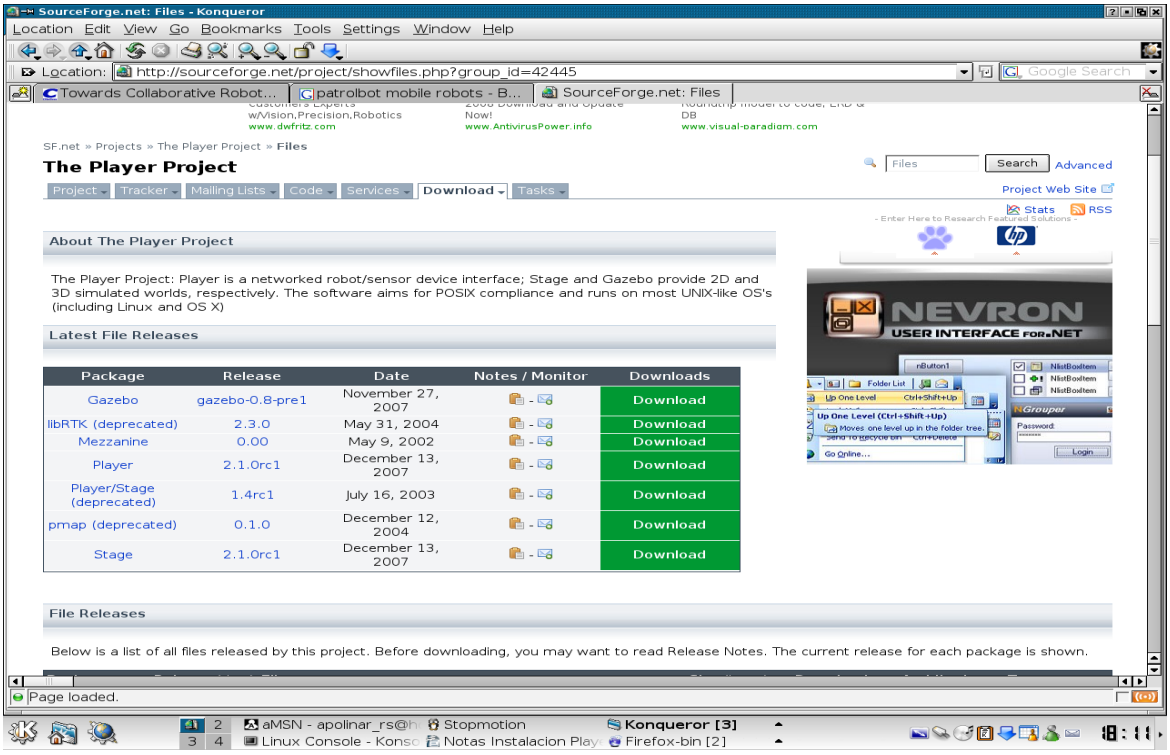
Algunos tips para desarrollo de videos a partir de la captura de la pantalla del Simulador Stage.

1) Seleccionar File->ScreenShot para activar la captura de imágenes de la ventana del simulador, (indicar “ScreenShot interval” y “sequence of frames”)
2) Utilizar Stopmotion http://developer.skolelinux.no/info/studentgrupper/2005-hig-stopmotion/index.php para armar el video y guardar en el formato conveniente.

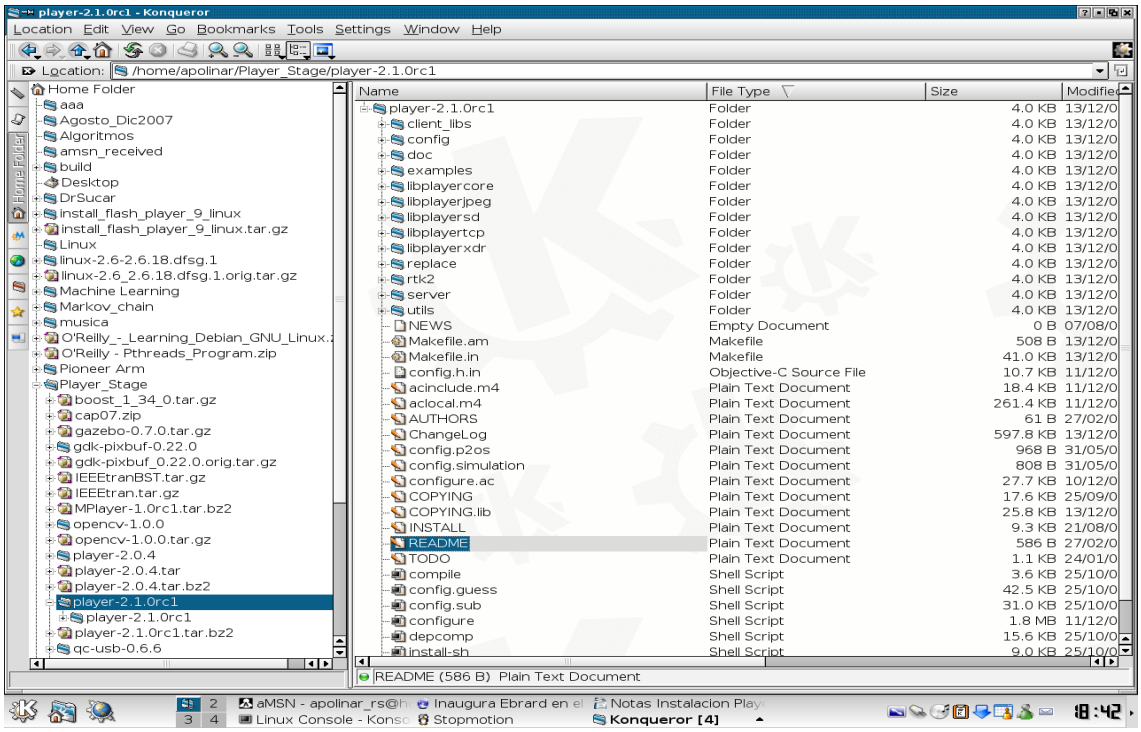


Notas de instalación del Player/Stage

Sitio: http://playerstage.sourceforge.net/



Una vez que se baja el archivo a tu cuenta de usuario, se descomprime en un subdirectorio.



1er Paso: Configurar el paquete: ./configure

Revisa paquetes indispensables para la instalación apropiada del Player:

Si sale este mensaje:

- Support for plugin drivers will NOT be included.
- You need plugin support to use Stage
- To add plugin support, install libltdl which is part of GNU libtool

- multithreading NOT included; Install BOOST (www.boost.org) libraries to enable multithreaded and/or signaling in libplayerc++

lista de drivers que se instalarán y los que no se instalarán

Player will be installed in: /usr/local/ (en /usr/local/bin/player y en /usr/local/share/player)

2) Cambiar a Superusuario y Teclear: make install

Instalación de Stage.

Bajar el archivo a un directorio y descomprimir:

- bzip2-d Stage...bz2
- tar -xvf Stage...tar

Instalar GTK+ http://www.gtk.org
herramienta para crear interfaces gráficas de usuario.
o librería libgtkmm-2.4

1er Paso: Configurar el paquete: ./configure
2o. paso: Cambiar a Superusuario y Teclear: make install

Stage queda instalado en /usr/local (/usr/local/share/stage/worlds, con archivos de configuración y del ambiente)

Probando el simulador Stage:

En /usr/local/share/stage/worlds/player simple.cfg

Probando la interfaz Player:

En /usr/local/share/player/examples/libplayerc/simple
En /usr/local/share/player/examples/libplayerc++/example0

Para comentarios, dudas:

Apolinar Ramírez, apolinar_r@ccc.inaoep.mx
Cubo 1114 -INAOE