Player Stage -- Free Software tools for robot and sensor applications

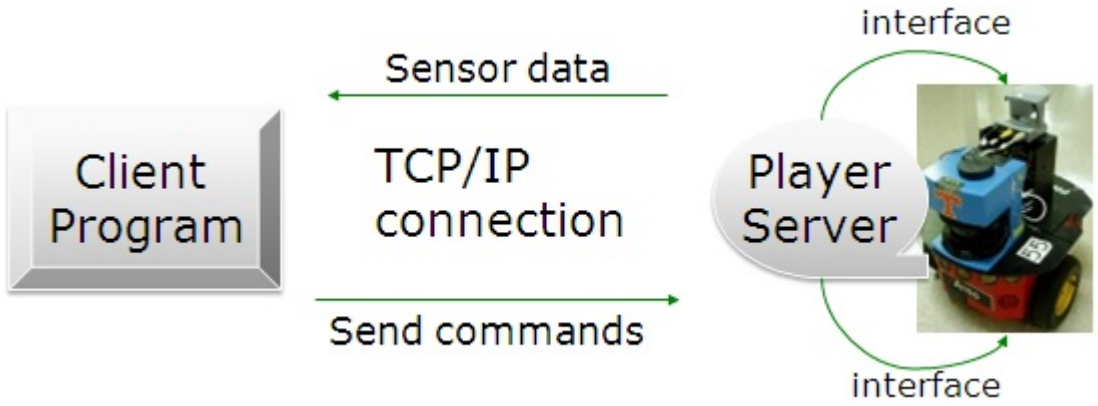
---

The Player/Stage main website: http://playerstage.sourceforge.net/

Check the FAQ for various questions. The information from this site is adapted from the main website and Jenny's tutorial.

Player and Stage Introduction

Player is a network server for robot control. It provides an interface to the robot's sensors and actuators over the IP network. Your client program talks to Player over a TCP socket, reading data from sensors, writing commands to actuators and configuring devices on the fly.



A simulation is composed of three parts:

- Your code. This talks to Player.
- Player. This takes your code and sends instructions to a robot. From the robot it gets sensor data and sends it to your code.
- Stage. It receives instructions from Player and moves a simulated robot in a simulated world, it gets sensor data from the robot in the simulation and sends this to Player.

Basics

Important File Types

Three kinds of file you'll need to understand:

- a .world file tells Player/Stage what things are available to put in the world. You describe your robot, any items which populate the world and the layout of the world.
- a .cfg (configuration) file is what Player reads to get all the information about the robot that you are going to use. This file tells Player which drivers it needs to use in order to interact with the robot (if you use simulation, the driver is always Stage). It also tells Player how to talk to the driver, and how to interpret any data from the driver so that it can be presented to your code.
- a .inc file, which follows the same syntax and format of a .world file but it can be included, which is easily reusable.

Interfaces, Drivers and Device

- Interface: A specification of how to interact with robotic hardware. The interface defines the syntax and semantics of all messages that can be exchanged with entities in the same class.
- Driver: A piece of software (usually written in C++) that talks to hardware and translates its inputs and outputs to conform to one or more interfaces.
- Device: A driver bound to an interface so that Player can talk to it directly.

Example: Consider the laser interface. This interface defines a format in which a planar range-sensor can return range readings (basically a list of ranges, with some meta-data). Now consider the sicklms200 driver, which controls a SICK LMS200 laser. The sicklms200 driver knows how to communicate with the laser hardware and retrieve data from it. It also knows how to translate the retrieved data to make it

conform to the format defined by the laser interface. The sicklms200 driver can be bound to the laser interface to create a device, which might have the following address: *localhost:6665:laser:0*.

The files in the above address correspond to the entries in the player_devaddr_t structure: host, robot, interface and index.

Player talks to the robot using ports (the default port is 6665). Player and Stage communicate through these ports (even if they're running on the same computer). The above line tells Player which port to listen to and what kind of data to expect. In the example, it's laser data which is being transmitted on port 6665 of the computer that Player is running on (localhost).

Building a World

We can run a world and configuration file that comes bundled with Stage. By default, it's in the /usr/local/share/stage/worlds folder. Once in the correct folder, you can run the following command: "player simple.cfg".

A worldfile is a list of models that describes all the stuff in the simulation, including the basic environment, robots and other objects.

Let's look at map.inc for an example.

```
include "map.inc"

# configure the GUI window
window
(
   size [700.000 700.000] # size of the simulation window in pixels [width height]
   scale 41 # how many pixels each meter requires, < window_size/floorplan_size
)

# load an environment bitmap
floorplan
(
   bitmap "bitmaps/cave.png" # can by type bmp, jpeg, gif or png
   size [15 15 1.5] # size in meters [x y z] dimensions
)
```

Writing a Configuration (.cfg) File

For each model in the simulation or device on the robot that you want to interact with, you will need to specify a driver. The driver specification is in the form:

```
driver
(
   name "driver_name" # a list of supported driver names is in the Player Manual, with Stage, the only one needed is "stage"
   provides [device_address] # what kind of information is coming from the driver
   # other_parameters
)
```

In your configuration file (.cfg), it always starts with the following for simulation:

```
driver
(
   name "stage" # there is a driver called stage
   plugin "stageplugin" # in the stageplugin library

   provides ["simulation: 0"]  # it conforms to the simulation interface

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

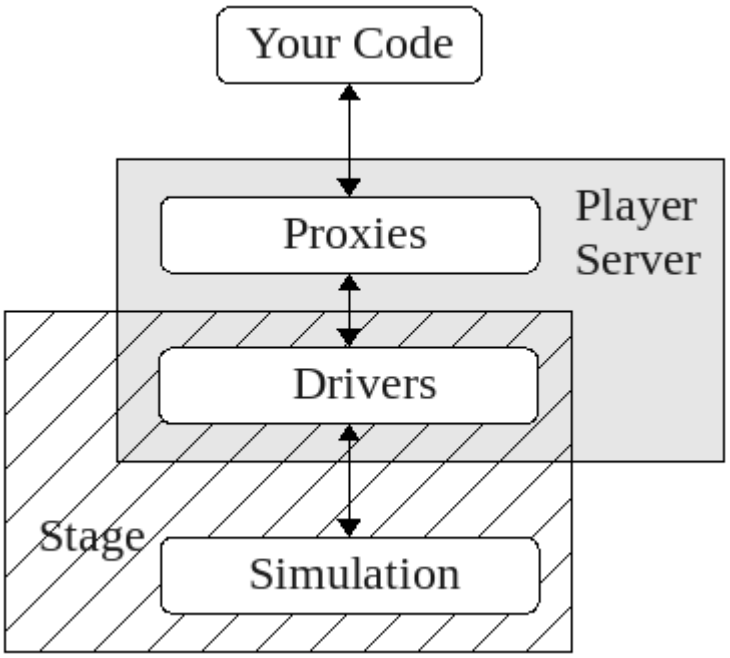
The input to the provides parameter is a "device address", which has the format "key:host:robot:**interface**:**index**" and parameters are separated by white space. For Stage, the key field doesn't need to be used. The default host is "localhost" which means the computer on

which Player is running. The robot field is the port number. The default port is 6665. The interface and index fields are mandatory.

```
driver
(
   name "stage"
   provides ["6665:position2d:0"
             "6665:sonar:0"
             "6665:blobfinder:0"
             "6665:laser:0" ]
   model "bot1"
)
```

Writing Your Code

Player uses a Server/Client structure in order to pass data and instructions between your code and the robot's hardware. Player is a server, and a hardware device on the robot is subscribed as a client to the server via **proxy**.



There are two kinds of proxies:

- the special server proxy PlayerClient: establish a connection to a Player server.
- the various device-specific proxies: initialized through PlayerClient proxy.

Here's an example to set up the connection to the Player server and proxies.

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

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

   Position2dProxy p2dProxy(&robot,0);
   SonarProxy sonarProxy(&robot,0);
   BlobfinderProxy blobProxy(&robot,0);
   LaserProxy laserProxy(&robot,0);

   //some control code
```

```
    return 0;
}
```

Jenny's guide also provides a useful introduction to various common proxies that we may use. Check this link for more information on proxy with C++.

To fully understand Player/Stage, check out the main website --> Manual --> Player Manual or Stage Manual with the version you install.

Installation Guide

Here're the steps that I took to install the latest versions of Player (v. 3.0.2) and Stage (v. 3.2.2).

- Install Ubuntu. (I used Ubuntu 14.10, you can find that release here).
- Before installing Player and Stage in Ubuntu, we need to install certain dependency packages. They can be installed from the graphical installer **Ubuntu Software Center**. This file contains the list of packages that I have installed on an earlier release of Ubuntu. But you should be able to install the latest versions.
- Double check with packages provided in this installation guide here.
- After the proper packages are installed, just follow the steps to download the latest player/stage. Both Player and Stage will use the same installation guide.
- This following script will automatically install all necessary packages, download and install player/stage for you. Click here.

To Compile and Run Your Control Code:

To compile your code, assume you name your project as proj.cc, and your player/stage is installed under /usr/local/bin (the default directory, please replace /usr/local/ with whatever directory where the player/stage is installed, follow the steps here:

- First ensure that player is in the PATH, by typing: $which player
- This should result in the following output: $/usr/local/bin/player
- Add the following lines to your .bashrc file located in your home directory:
    - export LD_LIBRARY_PATH+=/usr/local/lib:/usr/local/lib64
    - export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:$PKG_CONFIG_PATH
- Run the .bashrc file by source
- Now everything should be working. To test "Stage", move to the "worlds" directory in stage, and run "stage simple.world", a simple world window should pop out. To test "player", run "player simple.cfg", a simple world window should pop out.
- Compile your program (note that ` is different from '):
  $ g++ -o proj `pkg-config --cflags playerc++` proj.cc `pkg-config --libs playerc++`
- An executable file named "proj" should be generated and you can test it with the stage simulator.
- There are some examples provided in the player directory "player/examples/libplayerc++". Just comple some examples and run.
- Note that the configuration files are outdated in the download. Please use these files here.

Daisy Tang, Last Updated: Jan. 2017