

## 1 Dates

- Assignment 1 handin due: Friday, February 6, 2009

## 2 Introduction

This project is structured to acquaint you with the basics of using Player/Stage/Gazebo (PSG) robot interface suite for writing robot controllers. Player is a network server for robot control. Player runs onboard a single robot and provides a clean interface to the robot's sensors and actuators over an IP network. Stage is a 2D robot simulator suited for large populations of robots in an environment specified by a bitmap image. Gazebo is a 3D physics-based robot simulator suitable for smaller numbers of robots simulated at high fidelity. The physics for Gazebo is provided by the Open Dynamics Engine (ODE), which integrates physical dynamics for arbitrary kinematic structures through optimization.

PSG provides an infrastructure for developing robot controllers. You will write controllers as client programs that send control commands to and request information from a robot through its Player server. Stage and Gazebo can simulate various types of robot platforms (i.e., hardware) and populations. The same interface, provided by the Player robot server, is used to control a robot in the real world or its equivalent in a Stage/Gazebo simulation. Robot platforms that are not currently supported in PSG can be developed through implementing appropriate Player server interfaces and devices in Stage or Gazebo.

Because of its flexibility and portability, PSG will give you good experience with developing robot controllers. Additionally, many interesting research-level projects can be implemented using PSG.



## 3 PSG Framework

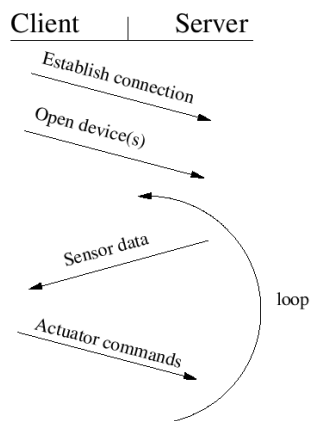


Figure 1.1: Example client/server interaction

As illustrated (left), PSG is a framework for robot control consisting of devices, robot servers, and robot clients.

Devices (e.g., a laser, a camera, or a complete robot) are actual hardware in the real world or simulated hardware that exists in a virtual environment maintained by Stage or Gazebo.

A robot server (e.g., Player) is the information interface between the robot and any program that requests information from or sends commands to the robot. Regardless of whether a device is real or simulated, the robot server provides the same interface to the robot for client programs. Thus, controllers developed on a simulated device will immediately run the equivalent real robot device. (Note: A robot's **behavior** is function of both its controller and environment. A robot controller working in one environment does not necessarily imply the same controller will yield the same behavior in new or similar environments, given PSG support for the device.)

A robot client is a user-developed program that accesses robot functions through the robot server. The robot client will first establish a connection to the robot's server and then command the robot by reading data from the server and sending appropriate control command

back. The job of the developer is to write control programs that produce commands that yield desired behavior from information received from the robot server<sup>1</sup>

Lastly, one of the major advantages of Player as a robot server is its independence from a particular client-development language. The interaction between Player and a client program is done completely over a TCP/IP (or UDP/IP) network connection. Thus, any language with libraries that supports Player functionalities can be used to develop robot clients. The most supported client languages are C and C++ (supported through `libplayerc` and `libplayerc++`).

## 4 Instructions

For this assignment, you will be expected to work with our installation of PSG to write a simple random traversal controller. Your `PATH` variable must point to the course directory for the tools to work, so add these lines to your `.environment` file:

```
pathappend PATH /courses/cs148/bin
setenvvar PKG_CONFIG_PATH /course/cs148/pkgconfig
setenvvar LD_LIBRARY_PATH /course/cs148/lib
pathappend PKG_CONFIG_PATH /contrib/projects/psg/local/lib/pkgconfig
pathappend LD_LIBRARY_PATH /contrib/projects/psg/local/lib
```

- Start by copying the directory `/course/cs148/asgn/asgn1/` somewhere you can use it.
  - Ask one of the TAs for access to the Roomba Lab (CIT 404), a SmURV robot, its IP address, and how to power up the robot.
  - Turn on the iRobot Create base (button located on top of the base towards the front-left and below the mounted ITX computer) and the mounted ITX computer (button located on its front face). Wait for the robot to boot. Booting is effectively complete when the wireless card on the back of the ITX computer is flashing/active.
  - From a machine in the Roomba Lab, ping the IP address of the robot to make sure it is on the network.
  - Run `playerv` with the following command `playerv -h <robot IP> -p 6665` and a window will pop up with a grid. Select the following:
    - \* `devices-position-subscribe`
    - \* `devices-position-command`
  - Now you can control the robot by dragging the '+' in the center around different directions. `Playerv` is simply a Player client with a gui.
- Now it is time to write our own client
  - In the directory `asgn1`, there is a sample `libplayerc++` client. Type 'make' to compile it.
  - To run on a SmURV, you must ensure the SmURV accessible via the Roomba Lab wireless running as well as player server, and type `./client <smurv ip addr>`. If the SmURV is not responsive, you may need to login to the SmURV as root and restart the Player server. The client argument is the IP address of the SmURV which is typically of the form `10.100.0.1xx`.
  - Look through the code for this client. It currently only rotates the robot in place until Ctrl-C is pressed.
- Your tasks for this assignment is:

---

<sup>1</sup>Referring to the lecture on autonomous control, the description of the robot client should remind you of feedback control (i.e., commands  $[u_t]$  that produce desired state  $[x_d]$  given observations  $[y_d]$ ).

1. Write a player controller to drive a SmURV robot randomly around a real-world environment while reacting obstacles using bumper sensing (as given through Player's bumper proxy).
2. Documentation for this is found on the player/stage website, <http://playerstage.sourceforge.net/doc/Player-2.1.0/player/>

## 5 Grading

Your grade for Assignment 1 will be determined by equal weighting of your group's implementation (50%) and your individual written report (50%). The weighted breakdown of grading factors for this assignment are as follows:

<b>Project Implementation</b>	
- Movement Control → Does your robot move smoothly in the environment?	20%
- Obstacle Reaction → Does your robot reasonably detect and respond to obstacles?	20%
- Controller Robustness → Does your controller run without interruption?	10%
<b>Written Report</b>	
- Introduction and Problem Statement → What is your problem? → Why is it interesting?	7%
- Approach and Methods → What is your approach to the problem? → How did you implement your approach and algorithms? → Could someone reproduce your algorithms?	15%
- Experiments and Results → How did you validate your methods? → Describe your variables, controls, and specific tests. → Could someone reproduce your results?	20%
- Conclusion and Discussion → What conclusions can be reached about your problem and approach? → What are the strengths of your approach? → What are the shortcomings of your approach?	8%