1 Dates

- Assignment 1 handin due: Friday, September 25, 2009 (10 pm)

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 implemented using PSG.

3 PSG Framework

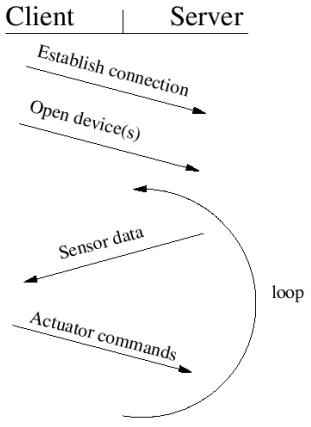

Figure 1.1: Example client/server interaction

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[1]

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
```

- Start by making sure you have the sample code for assignment one.
  - If you haven't done so already, check out your groups svn repository by running: `svn checkout svn://foxwood/groupname/trunk <dirname>`.
  - Pull the skeleton code from the repository into your trunk: svn merge `svn://foxwood/groupname/branches/asgn1_skel`.
  - Ask one of the TAs for access to the Roomba Lab (CIT 404), an iRobot Create, and an IP address. Also ask them for your svn username/password and verify your group name, which you'll need for the second part of the assignment.
  - Turn on the iRobot Create and the mounted EeePC computer. Make sure that the EeePC is connected to the AIBO wireless network (no password is necessary).
  - Each EeePC has an IP address in the form 10.100.0.x, where x is posted next to each robot name on the whiteboard in the Roobma Lab. Determine the IP address for the robot you are currently working with by looking at the whiteboard or by running the command `/sbin/ifconfig` on the EeePC.
  - From either foxwood or sandworm (the two computers in the Roomba Lab), ping the IP address of the robot to make sure it is on the AIBO network.
  - The Player server requires a configuration file that tells it what robot systems to load drivers for. A basic configuration file, suitable for this assignment, is included with the assignment 1 files. The file is called `create.cfg` and must reside in the home directory of the EeePC you are working with. If `create.cfg` is not already in the home directory of the EeePC, you will need to put it there.
  - You have to start the Player server to be able to connect to the robot and run any Player client. Also with the assignment 1 files is a script called runplayer.sh that can be used to start a Player server on one of the robots. If you check out your assignment 1 files onto a robot (using the same svn commands to checkout the repository), you can start a Player server on it by running `sh runplayer.sh` from the terminal on one of the EeePCs. Note that, if a Player server is already running, you may get an error message when you try to start a new one. To start a new server, you will have to kill the existing Player process.

[1] 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 $[y_d]$).

  - `cd /course/cs148/bin` and 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 `asgn1_skel` directory, there is a sample libplayerc++ client. Type `cmake .` to build your makefiles. Type `make` to compile the code.
  - After starting the Player server on the EeePC with `sh runplayer.sh` (or with the command `robot-player <config_file>`, you can the the Player client by typing `./client <IP address>` from foxwood/sandworm.
  - Look through the code for this client. It currently only rotates the robot in place until Ctrl-C is pressed. The next time you want to run your client, you will need to restart the Player server on the EeePC.

- Your tasks for this assignment is:
  1. Write a player controller to drive a SmURV robot randomly around a real-world environment while reacting to 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:

Project Implementation	
- 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%
Written Report	
- 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%