CS264 Practical 1 - Player and Stage

Tom Blanchard - ttb7

February 14, 2013

Introduction

The practicals for this module are designed to give you some experience of working with both simulators and real robots. The first practical (this one) provides an opportunity to work with one of the most commonly used simulators for robotics. The second practical moves on to combining the simulator with a real robot. Finally for practicals three to five you will work purely with real robots.

Along with the five practicals there are three pieces of assessed work for the module. Together they make up your mark for this module; there is no exam. The assessed work takes the form of written reports. The reports will require you to spend extra time working on the simulator or on robots outside of lectures/practicals.

Player, Stage and Gazebo

Throughout these practicals we will be using Player. Player is possibly the most widely used robot control interface in the world. It could be called a 'robot server' as it provides a network interface to a large number and variety of robots and sensor hardware. It also enables programs to be written in pretty much any language and run on any computer with a network connection to the robot.

The simulation environment that we will be using for these practicals is called Stage. It is a 2D simulation environment, with provided models for sensors such as sonar, laser rangefinders, PTZ[1] cameras with blob detection and odometry. These sensors, or 'devices', present a standard Player interface and therefore require almost no changes to move between simulation and robot.

Although not used in these practicals, Gazebo provides a 3D simulation environment for Player. It is capable of generating realistic sensor feedback and physical interaction between objects. Like Stage it is very simple to move from simulation with Gazebo to real robots. In fact most controllers written for Stage can be used, without modification, with Gazebo.[2]

Working in the ISL

Everything in the ISL[3] is on a separate network from the rest of the university. Before you do anything else, log in to your computer to make sure that your account has been added to the network. If you cannot log in, inform one of the demonstrators.

Another important thing to note is that **there is no guarantee that work save locally to these machines will be here for your next practical**. Work on these machines is not backed up, it is **YOUR** responsibility to save a copy of your work to a safer location (Pen Drive, Central File store, etc).

[1] Pan Tilt Zoom
[2] playerstage.sourceforge.net
[3] Intelligent Systems Lab

The ISL is a robot lab and as such, it is very likely that there will be robots roaming around the floor. Please watch where you step to avoid tripping over a robot, they have an impressive ability to sneak up on people.

Getting started

In your home directory make a new directory called something sensible such as 'prac1'. You then need to download and extract the resources from: `http://users.aber.ac.uk/ttb7/modules/CS264/prac1/files.tar`. The following commands will do this from the command line.

1. mkdir prac1
2. cd prac1
3. wget http://users.aber.ac.uk/ttb7/modules/CS264/prac1/files.tar
4. tar xvf files.tar

In your prac1 directory you should now have 4 files; files.tar, simple.cfg, simple.cpp and simple.world. It's best to keep the tar file in case you need a clean copy of any of these files.

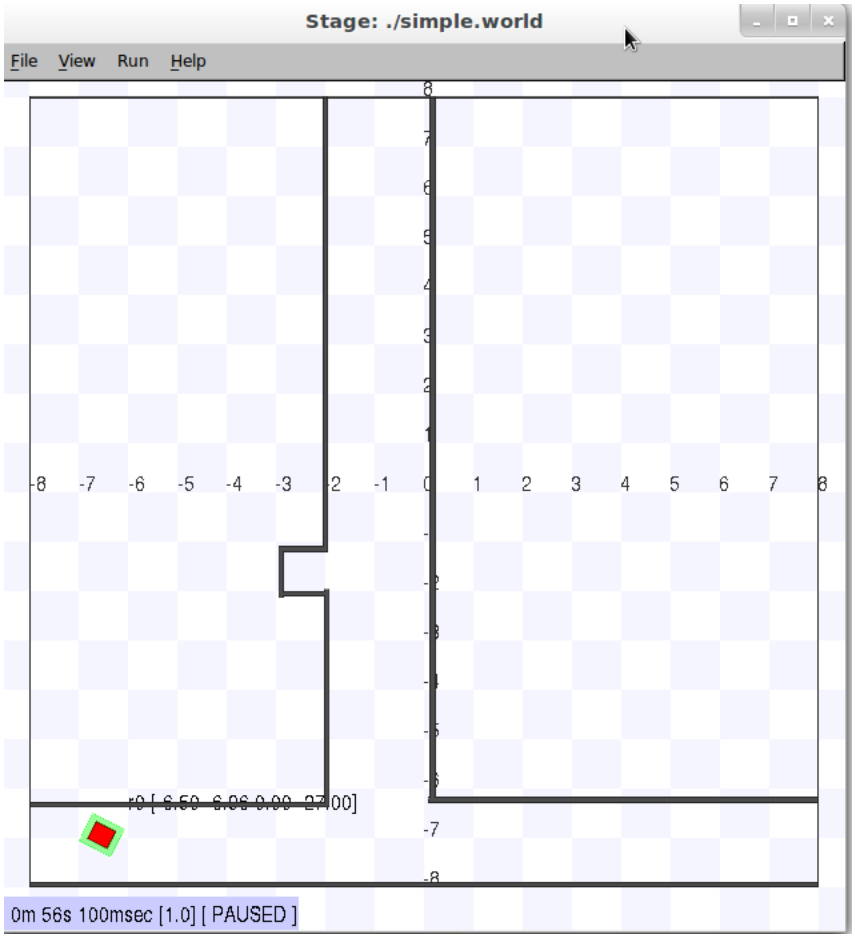
- simple.cfg is the config file for player, it details which simulator to use and creates a driver to control our simulated robot.
- simple.world describes the world used in the simulation, including the window size/scale/rotation, the floor plan and any robots to be used.
- simple.cpp is the c++ code to actually control your robot. It is the file you will be editing throughout these practicals.

You should NOT have to edit either the world or cfg files.

The cpp file already contains some example code, to compile and run it type:

1. `g++ -o simulator ‘pkg-config --cflags playerc++‘ simple.cpp ‘pkg-config --libs playerc++‘`
2. (In a new konsole window) player simple.cfg
3. (In the original konsole window) ./simulator

A Stage window should open and show a small red robot and some obstacles. You should be able to see the robot slowly move around the world using it's sonars to try to avoid obstacles.



You can stop your program by pressing Ctrl-C and you can close the Stage window either through the GUI or using Ctrl-C.

Task 1 - Basic Movement and Sonar

Spend some time making your robot move around; try making it go in straight lines, curves, circles and a square. You can find the Player C++ documentation on line, an example of a function used to move the robot is pp.SetSpeed(speed, turnrate);.

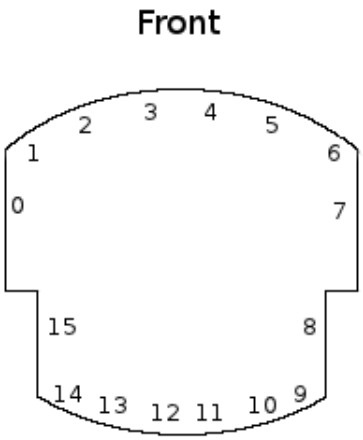
Each Pioneer robot has 16 sonar sensors which can be read using a RangerProxy. The following code will create a RangerProxy for a PlayerClient called 'robot'

- RangerProxy rp(&robot,0);

This will create a RangerProxy for the first ranger device, in our case the sonar, on the robot. Each of the 16 individual sonars would then be read like so:

- rp[n]

Where 'n' is a integer from 0-15. The layout of the sonar sensor's and their corresponding indexes are shown below.



To refresh the data from the robot, ie acquire new sonar or odometry data, you need to call '.Read()' on your PlayerClient object. A fully working example, using the sonars is shown in the sample code in simple.cpp.

Task 2 - Precise turns and movement

By now you may have noticed that there are no functions to turn to a specific angle or move forwards by a set distance.

Write two functions, one that turns the robot to an angle and one that moves the robot forwards/backwards a set distance. Both the desired angle and distance should be passed as parameters to the functions.

The following functions of the Position2dProxy may be of use:

- .GetXPos() - uses odometry to get the robots X position.
- .GetYPos() - uses odometry to get the robots Y position.
- .GetYaw() - uses odometry to get the robots angle.

Also of use are the utility functions 'dtor' and 'rtod', which convert to and from radians and degrees.

Document History

Version	Date	Author(s)	Description
1.1	Semester 2, 2013	Tom Blanchard	Initial Version