CS264 Practical 1 - Stage (and Player)

Tom Blanchard - ttb7, Myra Wilson - mxw

February 5, 2015

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 (for use in practical sessions 2 and 3) moves on to combining the simulator with a real robot.

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.

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.

[1]Pan Tilt Zoom

- 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.

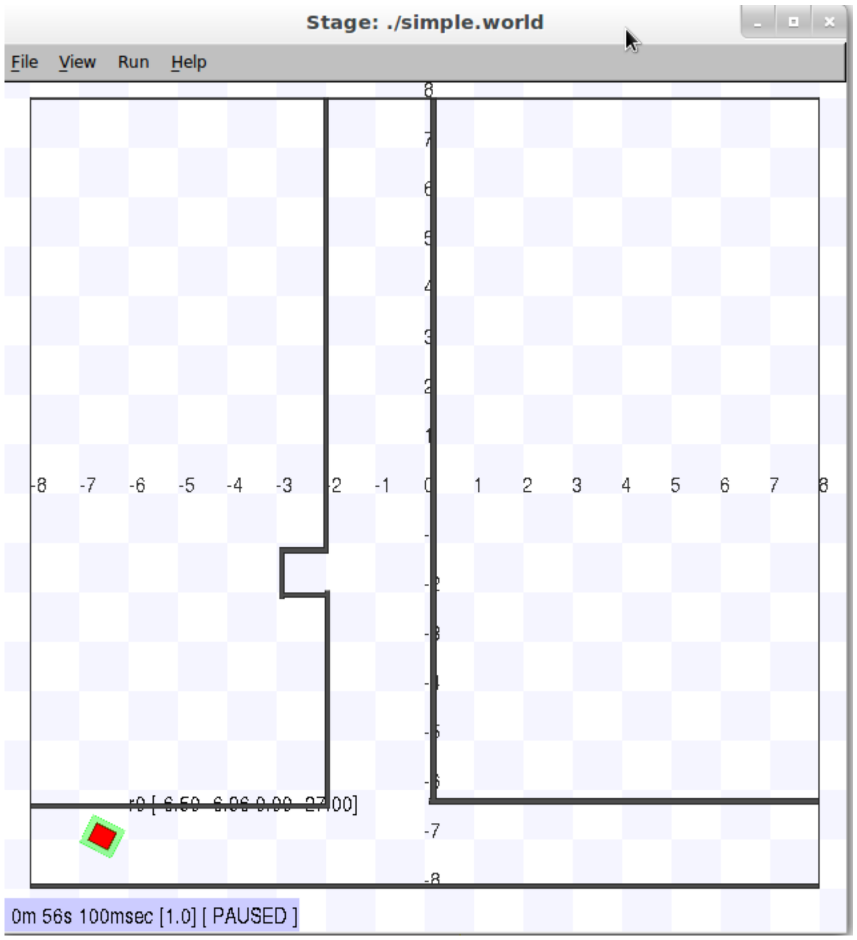
The cpp file already contains some example code, to compile and run it:

1. Click 'Edit'-'Profile Preferences', select the 'Title and Command' tab and check the 'Run command as a login shell' option.
2. Type `` g++ -o simulator `pkg-config --cflags playerc++` simple.cpp `pkg-config --libs playerc++` `` Please note that what looks like single quotes are actually backticks, the key to the left of "1" on a standard keyboard.
3. (In a new konsole window) player simple.cfg
4. (In the original konsole window) ./simulator

If you get an error saying **'Package playerc++ was not found in the pkg-config search path.'** then enter the following into the terminal:

export PKG_CONFIG_PATH=/usr/local/lib64/pkgconfig/

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 its 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);. The speed parameter defines how fast the robot tries to move forwards and turnrate is how fast the robot will turn.

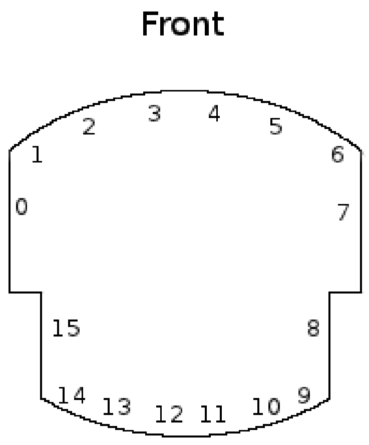
Each Pioneer robot has 16 sonar sensors which can be read using a RangerProxy. The following code will create a RangerProxy for a PlayerClient called 'robot'

- 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.

Using Java

Although we strongly advise the use of c++ to control the robot, there is an option to use Java as detailed below. You can get a preconfigured eclipse project by following these steps:

1. Get the file: wget http://users.aber.ac.uk/ttb7/modules/CS264/prac2/cs264_java.zip
2. Move it to your eclipse workspace: mv cs264_java.zip /workspace/
3. Unzip the file: unzip cs264_java.zip
4. Open Eclipse and go to ‘File’ - ‘Import’ - ‘General’ - ‘Existing Projects into Workspace’.
5. In the ‘Select root directory’ box click ‘Browse’ and select the ‘Player’ folder you just extracted. Press ‘Finish’.

There are three classes you need which can be found in the project package:

- Robot.java - Example code for the physical robots.
- Simulator.java - The same code but for the simulator.
- Runner.java - lets you choose between running Robot or Simulator by changing the boolean ‘simulation=true/false’.

When transferring this to the real robot in pracs 2 and 3, remember to find out the name of your robot, for example bart.islnet and use that as the parameter passed to the Robot constructor. eg ‘new Robot(”bart.islnet”);’