# CIS 32 Spring 2009, Project II

## 1 Description

This project involves writing a control program for a more realistic robot. Again, you'll be writing this program in Player/Stage. The aim of this exercise is to transition from the simple simulated robot we used in the first project to a realistic simulation of the Pioneer robots we have in the lab.

## 2 Login, get ready.

1. Login to your computer. The user name is `student`, the password is `student`.
2. The machines run Ubuntu, a flavor of Linux. Some of the things you have to do should be familiar from CIS 15 where you should have used Unix.
3. If you didn't use Unix, you'll get the hang of it quickly.
4. The first thing to do is to open a terminal window.

   - Just click on the  icon on the menu bar at the top of the screen.
   - You can also get hold of the terminal via:
     **Applications**> **Accessories**> **Terminal**
   - In fact, open two of these windows.

## 3 A new simulated world

1. Most of our interaction with Ubuntu will be through these terminal windows.
2. Start by getting to the right place in the file system. To do that type:

   `cd cis32`

   in both terminal windows.
3. You need to add four new files to this directory:

   - `pioneer.cc`
   - `pioneer.inc`
   - `world2.cfg`
   - `world2.world`
4. In one terminal, type:

   `player world2.cfg`

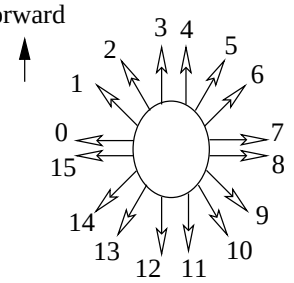
   This should pop up a square window labelled **Player/Stage: ./world2.world** which contains a red blob and a strangely shaped lump insides a jagged line. This is the simulated world in which your robot will operate.
5. The world is more complex than the one you used in the first project, but the robot has more sensors, and that makes it even.

# 4 What's happening?

1. The red blob is the simulated Pioneer.
2. In another window type:

   `playerv --position2d --sonar`

   This will activate the simulation of the sonars.
3. The robot has 16 sonar sensors. These are the cones you see projecting from the red blob, or rather the cones show you the simulated beams of ultrasound projected by the simulated sonar.
4. The layout of the sonar is:

   where the numbers indicate which sonar is pointing in which direction. $forward$ is the direction to robot will move in with a positive value of $xSpeed$.
5. This robot has a *differential drive*, which means that you can make it go forwards and backwards, and you can also make it turn.
6. A positive value of $tSpeed$ will make the robot turn to its left.

# 5 The challenge

1. Write a controller to make the robot move around the outside of the space.
2. To do that you have to write a C++ program.
3. To get you started, you have the skeleton program `pioneer.cc`, which you can also find at the end of these instructions.
4. To edit the program, type:

   `xemacs pioneer.cc &`

   into one of your terminal windows. This starts up an editor which you should find pretty easy to use — you can do all you need to do from the menu bar at the top of the editor window.
5. The `&` runs the program in the background so that you don't need to close the editor to continue to use the terminal.
6. To compile the program, type:

   `./build pioneer`

   into one of your terminal windows. This will create a controller called `pioneer`.
7. If you type `./build pioneer.cc` you'll get an error.
8. To run your controller, type:

   `./pioneer`

   into one of your terminal windows.

9. Try running through steps 6 and 8 with the skeleton program (remember to start up the simulated world first). This should not move the robot, but it should produce a stream of sonar values.
10. You can pick up the robot and move it around the simulated world with the mouse/trackpad.
11. Now, enough playing, let's write that controller.

```
//---------------------------------------------------------------------
//
// pioneer.cc
//
// Example player libplayerc++ controller that uses sonar to handle a
// pioneer robot.
//
// This skeleton written: 2nd March 2009
// Written by: Simon Parsons
//
// Usage:
//
// wall-follow [-h <host>] [-p <port>] [-i <index>] [-m]
//
//  -h <host> : connect to Player on this host
//  -p <port> : connect to Player on this TCP port
//  -i <index>: connect to this device (default 0)
//  -m        : turn on motors

//---------------------------------------------------------------------

#include <iostream>
#include <libplayerc++/playerc++.h>

#include "args.h" // Code that allows us to handle standard arguments

using namespace PlayerCc;

//---------------------------------------------------------------------

int main(int argc, char *argv[])
{
  //---------------------------------------------------------------------
  //
  // Variables

  // For setting speed

  player_pose2d speed;

  const  double stdSpeed = 0.4;
  double xSpeed;
  double ySpeed;
  double tSpeed;

  //---------------------------------------------------------------------

  // Parse the arguments and set up the relevant variables.
  parse_args(argc,argv);

  // Use the arguments to connect to the specified device.
  PlayerClient robot (gHostname, gPort);
  Position2dProxy pp (&robot, gIndex);
  SonarProxy sp (&robot, gIndex);
```

```
  // Control loop starts

  while(true)
  {
    // Read sensor values. This reads new sensor values into the array sp
    robot.Read();

    // Print out the sonar values. The sonar readings are an array of 16
    // values
    //
    // Left to right across the front.
    std::cout << "Sonar front" << std::endl;
    std::cout << sp[0] << std::endl << sp[1] << std::endl;
    std::cout << sp[2] << std::endl << sp[3] << std::endl;
    std::cout << sp[4] << std::endl << sp[5] << std::endl;
    std::cout << sp[6] << std::endl << sp[7] << std::endl << std::endl;
    //
    // Right to left across the back
    std::cout << "Sonar rear" << std::endl;
    std::cout << sp[8] << std::endl << sp[9] << std::endl;
    std::cout << sp[10] << std::endl << sp[11] << std::endl;
    std::cout << sp[12] << std::endl << sp[13] << std::endl;
    std::cout << sp[14] << std::endl << sp[15] << std::endl << std::endl;

    // Start with speed zero
    xSpeed = 0;
    ySpeed = 0;
    tSpeed = 0;

    //-------------------------------------------
    //
    // Write your controller here
    //
    //

    // Setup speed values to send them to the robot.
    // speed.px is speed in the x direction
    // speed.py is speed in the y direction
    // speed.pa is rotational speed.
    speed.px = xSpeed;
    speed.py = ySpeed;
    speed.pa = tSpeed;

    // Now use those values to drive the robot
    //
    // This time around, setting ySpeed won't have any effect since the robot
    // is differential drive.
    pp.SetSpeed(speed);
  }

  // Control loop ends
}
```