CIS 32 Spring 2009, Project I

1 Description

This project involves writing a control program for a robot that is very similar to the wall-following program that we discussed in class (in fact if you try to build *exactly* that controller, you'll be on the right track).

You'll be writing this program in a simulation environment called Player/Stage. The aim of this initial exercise is to get you used to Player/Stage so the robotics bit is easy (we use a simulated robot that is very like the simple agent we discussed in class). When you are more familiar with Player/Stage we'll use mor realistic simulated robots, and hopefully end up running programs on real robots.

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.

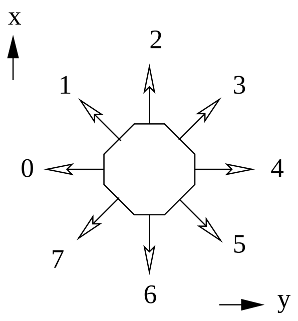
   - Just click on the icon on the menu bar at the top of the screen.
   - You can also get hold of the terminal via:
     Applications> **Accessories**> **Terminal**
   - In fact, open two of these windows.

3 Play with Player

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. In one terminal, type:
   ```
   player wall.cfg
   ```
   This should pop up a square window labelled **Player/Stage: ./wall.world** which contains an orange dot and a strangely shaped lump. This is the simulated world in which your robot will operate.
4. Now, in the second terminal, type:
   ```
   ./follower
   ```
   A stream of text should invade the window, and the orange dot should start moving around the outside of the square window.
5. When you have had enough, close the square window to stop the simulation.

4 What's happening?

1. The orange dot is a simulated robot.
2. The robot has 8 sonar sensors. These are the cones you see projecting from the orange dot, or rather the cones show you the simulated beams of ultrasound projected by the simulated sonar.
3. The layout of the sonar is:

   where the numbers indicate which sonar is pointing in which direction. x and y are the directions you'll need to know about when you write the code for the controller.
4. The robot is *omni-directional*, meaning that you can tell it to move in any direction (rather like the simple robot we discussed in class).
5. The program `follower` is a robot controller that I wrote to make the robot follow the outside wall of the simulated world.
6. This is not a very good controller, but can serve as a model for what I want you to write.
7. The stream of numbers are the readings from the eight sonars.

5 The challenge

1. Write a controller to make the robot do what my controller makes it do — 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 `wall-follow.cc`, which you can also find at the end of these instructions.
4. To edit the program, type:

   ```
   xemacs wall-follow.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 wall-follow
   ```

   into one of your terminal windows. This will create a controller called `wall-follow`.
7. If you type `./build wall-follow.cc` you'll get an error.

8. To run your controller, type:

   ```
   ./wall-follow
   ```

   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.

6 Onwards and upwards

1. If you finish the challenge and want something harder to do, try seeing how your controller runs in the more complex world you get with:

   ```
   player wall2.cfg.
   ```

2. Don't be disheartened if your controller can't handle this rough wall. If you run my example controller:

   ```
   ./follower
   ```

   you'll see it easily gets stuck (due to an unfortunate choice of default action).

```
//----------------------------------------------------------------------
//
// wall-follow.cc
//
// Example player libplayerc++ controller to make a simple wall
// following controller that uses sonar for an omnidirectional robot.
//
//
// This skeleton written: 18th January 2008
// 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 8
    // values
    std::cout << "Sonar" << 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;

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

    // In here you need to write your controller!

    // 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
    pp.SetSpeed(speed);
  }

  // Control loop ends
}
```