

CIS 32.5 Fall 2009, Project I

1 Description


This project involves writing a couple of control programs for a simple robot, in fact for the Roomba, the robot vacuum cleaner that I talked about in class:



You'll be writing this program in a simulation environment called Player/Stage. The main aim of this initial exercise is to get you used to Player/Stage so the robotics bit is easy. When you are more familiar with Player/Stage we'll use more 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 `r0b0t`.
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 cis325
```

in both terminal windows.
3. In one terminal, type:

```
player world.cfg
```

This should pop up a square window labelled **Player/Stage: ./world.world** which contains a grey dot and some strangely shaped lumps. This is the simulated world in which your robot will operate.
4. Now, in the second terminal, type:

```
./build roomba-roam
```

and then:

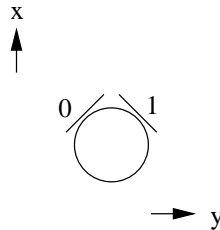
```
./roomba-roam
```

A stream of text should invade the window, and the grey dot should start moving, and keep going until it collides with one of the strangely shaped objects..

5. You can pick up the robot and move it around the simulated world with the mouse/trackpad.
6. When you have had enough, close the square window to stop the simulation.

4 What's happening?

1. The grey dot is a simulated roomba.
2. The robot has two bumpers. These are the lines you see projecting from the grey dot.
3. The layout of the bumpers are:



where the numbers indicate which bumper is which. x and y are the directions you'll need to know about when you write the code for the controller.

4. This robot has a *differential drive*, which means that you can make it go forwards and backwards (in the direction of x , and you can also make it turn.

5 The challenge

1. The controller `roomba-roam` just makes the robot run straight ahead until the robot hits something. Then the robot stops and doesn't move.
2. The challenge is to write a controller that has the robot roam around the world, that is keep moving, not get stuck and eventually explore the whole space.
3. To do that you have to write a C++ program.
4. To get you started, you have the program `roomba-roam.cc`, which you can also find at the end of these instructions.
5. To edit the program, type:

```
xemacs roomba-roam.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.
6. The `&` runs the program in the background so that you don't need to close the editor to continue to use the terminal.
7. To compile the program, type:

```
./build roomba-roam
```

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

```
./roomba-roam
```

into one of your terminal windows.

10. Now, enough playing, let's write that controller.
11. Note that a positive value of `turnrate` will make the robot turn to its left. A negative value of `speed` will make the robot go backwards.
12. When you have something you think works okay, test it on the real Roomba.
13. When the real Roomba behaves as you expect, save your program as **proj1-part1.cc** and make sure you put your name in the comments.
14. You'll need to submit this to Prof. Parsons after you are done with the project.

6 Onwards and upwards

1. Try making the roomba follow a wall — either the outside of the space it is in, or the outside of an obstacle it encounters.
2. As before, when you have a program that behaves as you want it to on the simulator, try it on the real robot.
3. When you are done, save your program as **proj1-part2.cc** and make sure you put your name in the comments.
4. You'll need to submit this to Prof. Parsons after you are done with the project.

```

/*
 * roomba-roam.cc
 *
 * Sample code for a robot that has two front bumpers. Suitable for use with
 * the roomba.
 *
 * This is example0.cc, changed to (hopefully) make it easier to understand.
 *
 * Modified: Simon Parsons
 * Date:     10th September 2009
 *
 */

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

int main(int argc, char *argv[])
{
    using namespace PlayerCc;

    // Variables

    double speed;      // How fast do we want the robot to go forwards?
    double turnrate;   // How fast do we want the robot to turn?

    // Set up proxies. These are the names we will use to connect to
    // the interface to the robot.

    PlayerClient    robot("localhost");
    BumperProxy     bp(&robot,0);
    Position2dProxy pp(&robot,0);

    // Allow the program to take charge of the motors (take care now)
    pp.SetMotorEnable(true);

    // Main control loop
    while(true)
    {

        // Read new information from the robot.
        robot.Read();

        // How far do we think we have gone?
        std::cout << pp << std::endl;

        // Print out what the bumpers tell us:
        std::cout << "Left bumper: " << bp[0] << std::endl;
        std::cout << "Right bumper: " << bp[1] << std::endl;
        // Can also print the bumpers with:
        //std::cout << bp << std::endl;

        // If either bumper is pressed, stop. Otherwise just go forwards

```

```
if(bp[0] || bp[1]){
    speed= 0;
    turnrate= 0;
}
else {
    speed=.1;
    turnrate = 0;
}

// What are we doing?
//std::cout << pp << std::endl;
std::cout << "Speed: " << speed << std::endl;
std::cout << "Turn rate: " << turnrate << std::endl << std::endl;

// Send the commands to the robot
pp.SetSpeed(speed, turnrate);
}
}
```