# CISC 3415 Fall 2011, Project I

## 1 Description

This project involves writing control programs for a simple robot, in fact for the Create, the education/research version of the Roomba robot vaccum cleaner.

All of the projects that we will do with this course will use the Player system. This provides an API for talking to robots — it reads data from sensors and writes commands to the robot.

Sometimes we will use the companion package Stage, which provides simulated robots and an environment for the robots to run in.

Today we will concentrate on learning about Player, using it to get the Create to do some simple things.

## 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/CISC 3110 where you should have used Unix.
3. If you didn't use Unix, you'll quickly get the hang of the few simple things we use it for.
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.
5. You also need to make sure that the laptop is connected to the robot. There should be a white cable that plugs into the right hand side of the robot (into a strangely shaped serial port). The other end of this cable is a USB connector that needs to go into a USB port on the laptop.

## 3 Play with Player and Create

1. Most of our interaction with Ubuntu will be through the terminal windows.

2. Start by getting to the right place in the file system. To do that type:

   `cd ~/Desktop/project1`

   in both terminal windows.

3. Now turn the Create on. The power button is just under the front of the wooden "deck" that supports the laptop, on the lefthand side of the robot.

4. In one terminal, type:

   `player roomba.cfg`

   This tells Player to open up a connection to the robot. The file roomba.cfg (so-called to remind us that the Create is just a vacuum cleaner with the cleaning bits removed) tells Player which device driver to use, and which port the device is connected to.

5. Now, in the second terminal, type:

   `./build roomba-roam`

   This compiles the program `roomba-roam.cc` which is listed later in these notes.

6. If you type `./build roomba-roam.cc` you'll get an error.

7. Once the program is compiled, run it using:

   `./roomba-roam`

   A stream of text should invade the window, and the robot should start moving,and keep going until it collides with something that it can't push out of the way.

8. When you have had enough of watching the robot run, hit `Ctrl-C` in both windows and turn the robot off.

# 4 The first challenge

1. Your first task for today is easy to state — make the Create drive in a square, 3 feet on a side, turning clockwise at each corner.

2. To do that you have to write a C++ program which makes function calls that talk to the robot hardware. The functions are provided by Player.

3. To get you started, you have the program `roomba-roam.cc`, which you can also find at the end of these instructions.

4. To edit the program, type:

   `gedit 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.

   Of course, if you have a prefered Unix editor, you can use it.

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. This robot has a *differential drive*, which means that you can make it go forwards and backwards, and you can also make it turn.

7. To compile the program, as before you type:

   `./build roomba-roam`

   into one of your terminal windows. This will create a controller called `roomba-roam`.

8. To run your controller, you go back and do exactly what you did in Section 3.

9. When the robot runs more or less in a square, run it five times and mark the location in which the robot stops.

   Don't spend a lot of time fine-tuning the performance of the robot. If it approximates a square, and gets back to within a foot of its starting point, that is god enough.

10. Measure the error for each run — the distance (in $x$ and $y$ directions) that the robot stopped away from the point it started at.

    This is a classic experiment in *dead reckoning navigation*.

11. Save these measurements into your program as a comment, and save the program as ⟨**your-names**⟩-**proj1-part1.cc**.

    The ⟨**your-names**⟩ should be the family names of the students in the group. So, if I was working in a group with Reid and Hart, my project would be called **parsons-reid-hart-proj1-part1.cc**.

12. Take a copy of the program with you and email it to Prof Parsons.

# 5 Some hints

1. 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.

2. You might want to start by making the robot drive three feet in a straight line.

3. Then make the robot turn through 90 degrees.

# 6 The second challenge

1. Repeat the first part, but with the robot going the other way around the square.

2. Again save the error measurements into your program, and save the program as ⟨**your-names**⟩-**proj1-part2.cc**.

3. Take a copy of the program with you and email it to Prof Parsons.

# 7 Onwards and upwards

1. If you have finished the two challenges, you might like to play with the simulator Stage.

2. In one of your terminal windows, 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.

3. Now, in the second terminal, type the usual:

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

   You should get the same kind of messages as before, but rather than the robot moving, the grey dot (the simulated robot) will move around the square window.

4. You can pick up the robot and move it around the simulated world with the mouse/trackpad.

5. When you run your programs from the two challenges on the simulator, how does the behavior of the simulated robot differ from how the real robot behaves?

6. When you have had enough, close the square window to stop the simulation.

```
/*
 *  roomba-roam.cc
 *
 *  Sample code for a robot that is suitable for use with the Roomba
 *  and Create.
 *
 *  Based on an example provided by Monica Anderson and Jeff Forbes,
 *  via Carlos Jaramillo, and changed to (hopefully) make it easier to
 *  understand.
 *
 *  Modified:    Simon Parsons
 *  Date:        15th June 2009
 *  Last change: 20th September 2011
 *
 */

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

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

  // Set up proxy. Proxies are the datastructures that Player uses to
  // talk to the simulator and the real robot.

  PlayerClient    robot("localhost");
  Position2dProxy pp(&robot,0);       // The 2D proxy is used to send
                                      // motion commands.

  int timer = 0;                      // A crude way to time what we do
                                      // is to count.

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

  // Control loop
  while(true)
    {
      double turnrate, speed;

      // Increment the counter.

      timer++;

      // Read from the proxies.
      robot.Read();

      // First make the rbot go straight ahead, then make it turn, and
      // finally make it stop.
```

```
      if(timer < 30){
speed = 0.1;
turnrate = 0;
      }
      else
      if((timer >= 30) && (timer < 60)){
speed = 0;
turnrate = 0.1;
      }
      else{
speed = 0;
turnrate = 0;
      }

      // Print out what we decided to do?
      std::cout << "Speed: " << speed << std::endl;
      std::cout << "Turn rate: " << turnrate << std::endl << std::endl;

      // Send the motion commands that we decided on to the robot.
      pp.SetSpeed(speed, turnrate);
    }

}
```