Using the “Stage” Simulation Environment

Dr. Daniel A. Ray
MCS Dept
UVa-Wise

June 8, 2010
Outline

1. Using Simulation Environments
2. Using the “Stage” Simulation Environments
3. Cracking the Control Code Code
4. Running A Simulation
5. Test Your Skills
Why Simulate?

- Testing code is an incremental process.
Why Simulate?

- Testing code is an incremental process.
- This is fine for normal code, but robots make this hard:
  - Control code run differently each time (changes in environment)
  - Control code can take a long time to run
  - Physically moving robots around is time consuming
  - Hardware nuances make life difficult; simulation allows you to test your logic without dealing with hardware (until later)

Note:
A simulation is only as useful as it is a faithful representation of the real world.
Using the “Stage” Simulation Environments

Outline

1. Using Simulation Environments
2. Using the “Stage” Simulation Environments
3. Cracking the Control Code Code
4. Running A Simulation
5. Test Your Skills
What is Stage?

- Stage is a 2D, physics-based simulation environment that can be used to simulate a variety of robots using a variety of sensors in a variety of environments.
  - Stage provides several different physics-based models for robot sensors and actuators.
  - Stage is not a program that you run by itself.
  - It is simply a framework that we can use when using the Player software (details on that to come shortly; both Player and Stage will be needed to run simulations.)
- Player and Stage can only be run in the Linux environment. You should be getting accustomed to it through the previous C++ lessons.
Let’s Take a Look at an Example

- Online we have several simple examples of using stage
- Stage works for any kind of robot; let’s look at an example using a Pioneer robot first

http://www.mcs.uvawise.edu/wiki/index.php/Examples_with_Stage
What do we need to make that work?

- Player/Stage is a free open-source platform that’s been installed on these computers
- Your job will be to supply the C++ code to control the robot
- 2 parts need to be understood
  1. The various files that tell the simulation environment how to function
  2. How to use the built-in libraries to make it easy to control a robot
1. Files provided for you

There are seven essential files required for running simulations with Player/Stage.

- .png image file: a 2D representation of the world (a map) the robot is in. These will be supplied to you.
- .inc file (roomba.inc): this will set up the parameters (size, shape, sensors) for our simulated robot. Provided for you.
- map.inc: sets up the environment (rules about obstacles on map, etc). Provided for you.
- .world file: describes the simulated world, linking the .inc files and image together into a complete environment. Provided for you.
- .cfg file: configuration file that sets up some important drivers for sensors based on your .world file. Provided for you.
- args.h: special purpose header file. Provided for you.
2. The “command code”

- This is the file that you will have to create
- It is C++ code (a .cc file) where you can use pre-built functions to read the robot’s sensors and tell it what to do
- Note: generally we are going to write code in such a way that it contains an intentionally infinite loop
- `for(;;){...}`
Example #2 - Now Using a Roomba (In a Cave)

http://www.mcs.uvawise.edu/wiki/index.php/Examples_with_Stage

Important Files

- Environment image file (.png)
- The .world file (.world)
- The .cfg file (.cfg)
- The control code (.cc)
  - Full Disclosure: There is a lot that won’t make sense at first with the C++ code
  - PlayerClient, BumperProxy, and Position2dProxy are what we call classes in C++
Outline

1. Using Simulation Environments
2. Using the “Stage” Simulation Environments
3. Cracking the Control Code Code
4. Running A Simulation
5. Test Your Skills
What the heck are classes?
How do we handle PlayerClient, BumperProxy, Position2dProxy

Definition

- **Classes** are structures in C++ that are used sort of like header files.
- Classes actually contain multiple functions and variables.
- The idea is that you, as the programmer, don’t need to know how they work, only how to use them.
- To use them, you must declare an instance of the class, just like declaring a variable. This variable contains its own copy of all the functions and variables defined in the class.
  - Position2dProxy pp (&robot, gIndex);
- When we define variables of a particular class type we use special functions called constructors (examples later)
- So, if you know what functions are defined for the class, then you know what functions to call to get some result.
- Calling functions involves using the “dot” operator
  - pp.SetSpeed(speed, turnrate);
- Let’s look at the control code again . . .
So... what functions are available

http://playerstage.sourceforge.net/doc/Player-2.1.0/player/

- The classes (and thus functions) are included as “client libraries”: `libplayerc++`
- Each different type of hardware (wheel/motors, bumper sensors, etc) are divided into “proxies”
- You won’t have to write any of these classes yourself, it’s already done. (That’s why we have header files!)
- `#include <libplayerc++/playerc++.h>`
- Disclaimer: Player/Stage is open source and free; might not all work exactly as advertised
Classes You Must Know About

1. PlayerClient
2. Position2dProxy
3. BumperProxy
PlayerClient

http://playerstage.sourceforge.net/doc/Player-2.1.0/player/classPlayerCc_1_1PlayerClient.html

- Allows us to define how Player/Stage will communicate with the robot (send commands, get sensor data)
- Will likely only need to use one function from this class: constructor (used when we define a variable of this classes type)
- PlayerClient robot (gHostname, gPort); – if doing a simulation
- PlayerClient robot ("192.168.0.213", 6665); – if using an actual robot
- robot – the variable name of type PlayerClient that defines our particular robot (simulated or real)
- As you’ll see, we can pass this in when we define variable names of types of other classes
Pretty much always the same up to this point. Now, we use these next two classes to actually control the robot.

http://playerstage.sourceforge.net/doc/Player-2.1.0/player/classPlayerCc_1_1Position2dProxy.html

- void SetSpeed (double aXSpeed, double aYawSpeed)
  - Note: There are three functions called SetSpeed, which do we use? Our robot is non-holonomic.
  - pp.SetSpeed(speed, turnrate);
  - Important: This doesn’t tell you anything about how far you’ve been or where you are.

- double GetXPos() : double x = pp.GetXPos();
- double GetYPos() : double y = pp.GetYPos();
- double GetYaw() : double yaw = pp.GetYaw();
BumperProxy: Most Important Functions

http://playerstage.sourceforge.net/doc/Player-2.1.0/player/classPlayerCc_1_1BumperProxy.html

- uint32_t IsBumped (uint32_t aIndex)
  - uint32_t test = bp.IsBumped(1);
  - bp[1] : BumperProxy also defines array-style notation for use with the bumpers
  - if(bp[0] || bp[1])...
  - On the Create there are two bumpers : left and right
Outline

1. Using Simulation Environments
2. Using the “Stage” Simulation Environments
3. Cracking the Control Code Code
4. Running A Simulation
5. Test Your Skills
Three Main Steps

1. Compile the control code: similar to before, but the class libraries complicate things a little

2. Start the Stage simulation environment in one terminal: It then waits on your control code to tell it what to do

3. Start your control code

Let's all download the code for Example #3

http://www.mcs.uvawise.edu/wiki/index.php/Examples_with_Stage
Three Main Steps

1. Compile the control code: similar to before, but the class libraries complicate things a little

2. Start the Stage simulation environment in one terminal: It then waits on your control code to tell it what to do
Three Main Steps

1. Compile the control code: similar to before, but the class libraries complicate things a little
2. Start the Stage simulation environment in one terminal: It then waits on your control code to tell it what to do
3. Start your control code

Let’s all download the code for Example #3

http://www.mcs.uvawise.edu/wiki/index.php/Examples_with_Stage
1. Compile Your Control Code

- Remember how we compiled C++ code before?

```
Remember how we compiled C++ code before?
```

```g++
−o example 'pkg-config −−cflags playerc++' example.cc
'pkg-config −−libs playerc++'
```

If there are no syntax errors in the code then you will get a blank prompt back.

Don't close the terminal window yet.
1. Compile Your Control Code

Remember how we compiled C++ code before? Use the `g++` command inside a terminal window.
1. Compile Your Control Code

- Remember how we compiled C++ code before? Use the g++ command inside a terminal window.
- Now we are going to use this command, plus some extra options to include the right libraries and to rename our executable from a.out.
- Make sure you have all 7 necessary files in a single folder: (.png, .world, .cfg, roomba.inc, map.inc, somefile.cc).
- Open a command terminal and navigate to the folder where you have put your command code: cd Desktop\myfile
- Note: because this command is a little difficult to remember, I like to keep it as a comment in my control code. Copy the line, click on the terminal and hit both mouse keys at once.
- g++ -o example ‘pkg-config --cflags playerc++‘ example.cc
  ‘pkg-config --libs playerc++‘
- If there are no syntax errors in the code then you will get a blank prompt back.
- Don’t close the terminal window yet.
2. Starting the Stage Simulation Environment

- You will need to open a second terminal (Applications > System Tools > Terminal)
- Navigate to the folder with all the necessary files: cd Desktop\myfile
- Type: player <configfilename>.cfg
- This will open a new window where an instance of stage starts running. You will see your environment and a red dot which represents the robot
3. Running Your Control Code

- Go back to the first terminal window that you opened
- Type in the name of your executable (in our case, it’s named example) and press enter
- This will run the program, instructing the robot to move and reading sensor data from the robot
- **IMPORTANT:** Whenever you alter your control code, you need to recompile it or the changes won’t be recognized by the robot!
Outline

1. Using Simulation Environments
2. Using the “Stage” Simulation Environments
3. Cracking the Control Code Code
4. Running A Simulation
5. Test Your Skills
Simulation Assignment #1

- Let’s get in groups for this next assignment

http://spirit.mcs.uvawise.edu/SimAssignments/assign1.pdf
http://spirit.mcs.uvawise.edu/SimAssignments/assign1.tar.gz
Simulation Assignment #2

http://spirit.mcs.uvawise.edu/SimAssignments/assign2.pdf
http://spirit.mcs.uvawise.edu/SimAssignments/assign2.tar.gz