Player, Stage and Gazebo

Dylan Shell

These slides are based on Nate Koenig's “Instant Expert's Guide.” Thanks Nate!
Overview

- Player, Stage and Gazebo Project
- Player
  - Device Abstraction
- Simulators
  - When, why, how, what
  - The right tool: Stage or Gazebo?
- Stage in depth
  - Installation, usage
  - Examples
- Gazebo in depth
  - Installation and usage
  - Examples
- Where to get help
Player, Stage and Gazebo Project

- Project that grew out of USC
  - Player: Interface for communicating with a robot's resources
  - Stage & Gazebo: Simulation tools
  - Mezzanine, pmap: Misc. other robotics tools

- GPL'd code, documentation and developer community available at:
  http://playerstage.sourceforge.net

Robotics research tools for robotics researchers by robotics researchers!
Player: Device Abstraction

- **What**
  - Hide unnecessary details from controller programs

- **Why**
  - Much common structure at the device (sensor and actuator) level
  - Produce (semi-)portable code
  - Ignore irritating details e.g., hardware revisions
  - Make client code/binary more general

- **How**
  - Standards based approaches (e.g., JAUS)
  - “What works” based approach (e.g., player)
Player: Architecture
Working with Simulators

- **What**
  - Mimic the real world, to a certain extent
- **When**
  - Always!!
- **Why**
  - Save time and your sanity
  - Experiments less destructive
  - Use hardware you don’t have
  - Create really cool videos
- **How**
  - Someone has probably already done it, so use it
Which Simulator?

**Stage**
- 2D
- Sensor-based
- Player interface
- Kinematic
- $O(1) \sim O(n)$
- Large teams (100’s)

**Gazebo**
- 3D
- Sensor-based
- Player
- Dynamics
- $O(n) \sim O(n^3)$
- Small teams (10’s)
Stage: Environment

- Simulates mobile robots, sensors and objects on 2-D bitmapped environment
Stage: Models

- Designed for multi-agent autonomous research
- Each model simple (computationally cheap)
- Can simulate many such models
Stage: Control

- Control models via Player
- Simulated hardware appear as their real counterparts
- Easy transition between simulation and real world
- Supports a wide range of devices:
  - Pioneers, laser, sonar, camera, pucks, etc.

Not yet in Stage v2.0.0a
Stage In Depth

- The result?
Stage: Worldfile

- Stage environment specified in a worldfile
- Describes the robots, sensors and objects
- Many examples provided in distribution
Example Stage Worldfile

# size of the world in meters
size [16 16]

# set the resolution of the underlying raytrace model in meters
resolution 0.02

# configure the GUI window
window
  (size [ 470.000 520.000 ]
   center [0 0]
   scale 0.035
  )

# load an environment bitmap
map
  (bitmap "bitmaps/cave.png"
   size [16 16]
  )

# create a robot
pioneer2dx
  (name "robot1"
   color "red"
   pose [-3.700 -1.400 54.612]
   sick_laser( )
  )
Running Stage

- Stage isn't a standalone executable (anymore!)
- To run stage, you run player with a configuration file that calls plugin module.

$ player <myworld.cfg>

Configuration Overview:

- **your-client**
  - Command-line arguments, custom config files...

- **player**
  - mysetup.cfg

- **stage**
  - mysetup.world
# load the Stage plugin simulation driver
driver
  (  name "stage"
      provides ["simulation:0" ]
      plugin "libstageplugin"

# load the named file into the simulator
   worldfile "simple.world"
  )

driver
  (  name "stage"
      provides ["position:0" "laser:0" "sonar:0" ]
      model "robot1"
  )

.
Gazebo
Gazebo: In Depth

• Simulates robots, sensors, and objects in a 3-D dynamic environment
• Generates realistic sensor feedback and physical interactions between objects
Gazebo In Depth

- Gazebo aims to be a high fidelity simulator
- Each model is NOT computationally cheap
- Small robot populations are possible.
  - Dependent on number and type of robots
Gazebo Client Code

- Client code (your program), can interface to Gazebo in two ways
  - Player
    - Simulation transparency
    - Get all of Player’s goodies
    - Easier, but with more computational overhead
    - Recommended for most cases
  - libgazebo
    - Shared Memory, direct interface
    - Fast, but requires more work
Gazebo Architecture

Gazebo

Client

Player

Joint

Interface

Model

World

Hinge

Slider

Ball and Socket

Universal

Hinge 2-axis

Sensor

Body

Camera

Laser

Odometer

Box

Ray

Plane

Sphere

Cylinder

Open Dynamics Engine

GLUT

Shared Memory

Commands

Data

0,1

0..n

1..n
The World

- A world is composed of a model hierarchy
  - Models define simulated devices
  - Models can be nested
    - Specifies how models are physically attached to one another
    - Think of it as “bolting” one model to another

Worldfile snippet:
- Pioneer with a sick laser attached
- Sick’s <xyz> relative location to Pioneer

```xml
<model:Pioneer2AT>
  <id>robot1</id>
  <model:SickLMS200>
    <id>laser1</id>
    <xyz>0.10.0 0.2</xyz>
  </model:SickLMS200>
</model:Pioneer2AT>
```
Gazebo Architecture
Available Models

- Factory
- GarminGPS
- GroundPlane
- LightSource
- MapExtruder
- ObserverCam
- Pioneer2AT
- Pioneer2DX
- Pioneer2Gripper
- Pioneer2Sonars
- SegwayRMP
- SickLMS200
- SimpleSolid
- SonyVID30
- Terrain
- TruthWidget
Gazebo Architecture
Working with Player

- Steps to creating a simulation:
  - Write Gazebo worldfile
  - Start Gazebo
  - Write corresponding Player configuration file
  - Start Player
  - Start client program

- Note:
  - Gazebo must be started *before* Player
  - Player must be *re-started* whenever Gazebo is restarted
Example: Using a single robot

- Single robot with a scanning laser range finder

```xml
<model:Pioneer2DX>
  <id>robot1_position</id>
  <xyz>0 0 0.40</xyz>
  <rpy>0 0 45</rpy>
  <model:SickLMS200>
    <id>robot1_laser</id>
    <xyz>0.15 0 0.20</xyz>
    <rpy>0 0 0</rpy>
  </model:SickLMS200>
</model:Pioneer2DX>
```
Example: Using a single robot

• Run Gazebo

  $\texttt{gazebo <myworld>}$

• Run Player

  $\texttt{player \ --g\ default\ <myconfig>}$

• Now run your code
Getting Player/Stage/Gazebo

- Source distributions from the Player/Stage website:
  - http://playerstage.sourceforge.net/
- No binary (pre-built) distributions
- Basic requirements:
  - Posix-like OS (Linux, OS X)
  - GCC
- Gazebo additional requirements:
  - OpenGL with hardware acceleration
  - Open Dynamics Engine (www.ode.org)
Installing Packages

• To use Gazebo:
  • Download, install:
    1. gazebo-0.5.2 (or CVS)
    2. player-1.6.5

• To use Stage:
  • Download, install
    1. player-1.6.5
    2. stage-2.0.0a

Recent version of ODE (0.5)
Recent version of wxpython (2.6.1.0)
(may need to set PYTHONPATH)

Check output of ./configure
Try local install, or set LD_LIBRARY_PATH
Getting Help

• Where to get help:
  • Your peers
  • Player/Stage website
  • Player/Stage/Gazebo mailing list archives
  • Player/Stage/Gazebo mailing lists
Questions?