Gazebo Tutorial

(by Rasko Pjesivac)

What is Gazebo?

Gazebo is a simulator for a small group of robots in a 3D environment. Similarly to Stage, a 2D environment simulator, Gazebo can simulate a population of robots, objects and sensors. There are a few differences between the two simulators. **Gazebo** is designed for a small number of robots while Stage can handle hundreds of robots. In addition, Gazebo has a higher precision then Stage. Since both Gazebo and Stage are Player-compatible, client programs written using one simulator can usually be run on the other one with some or no modifications.

Installing Gazebo

You do not need to worry about installing Gazebo since it is already installed on UTK CS Linux machines. However, if you decide to do it yourself**,** this tutorial describes the process that you have to follow (this process can be tricky!).

First, you should go to official Player-Stage/Gazebo website http://playerstage.sourceforge.net/gazebo/gazebo.html. Then go to the download section to get the latest version of Gazebo along with the updated documentation. Read the documentation!

Steps to install Gazebo.

1) Installing the requirements of Gazebo:

GUI component of Gazebo requires two third-party packages *SWIG* and *wxPython* (Python bindings for *wxWidgets*)**.**

Beside these two packages, minimal Gazebo installation also requires *ODE* (OpenDynamicEngine). In addition, some terrain models and roads require *OPCODE* collision detection library, which is now part of the *ODE* distribution. Therefore, when installing *ODE* package make sure that you are installing it with *OPCODE* collision detection library enabled.

To support terrain builder utility of Gazebo, *GDAL* library (Geospatial Data Abstraction Library) has to be installed as well.

Links to these packages and installation instructions are provided on Gazebo's website. Once all these packages are installed you can start configuring and installing Gazebo.

2) Installing Gazebo:

First, you need to get the source code from the official website (if you still have not done it). After this, you will need to set up some necessary compiler paths for your shell. The paths for UTK-CS machines are listed bellow:

```
export PATH="$PATH:/usr/local/lib"
export PLAYERPATH="/usr/local/lib"
export LD_LIBRARY_PATH="/usr/local/lib:/usr/local/pkgs/Python-2.4.1/lib"
```

In addition to these paths, you might need to set up the following flags (these are UTK-CS specific):

```
export CFLAGS="-I/pkgs/player-stage-2.0.3/include -I/usr/local/include"
export CPPFLAGS="-I/pkgs/player-stage-2.0.3/include -I/usr/local/include"
export LDFLAGS="-L/pkgs/player-stage-2.0.3/lib -L/usr/local/lib -Wl,-
rpath=/pkgs/player-stage-2.0.3/lib -Wl,-rpath=/usr/local/lib"
export PYTHONPATH="/pkgs/player-stage-2.0.3/lib/python2.4/site-
packages/:/pkgs/player-stage-2.0.3/lib/python2.4/site-packages/wx-2.7.2-gtk2-
ansi"
```

You are now ready to run `./configure` and `make` scripts to install Gazebo. With `./configure` script (which you have downloaded with the source code) you can specify the path where you want to install Gazebo. This can be done by typing:

```
linux:~> ./configure -prefix=/home/username/wherever/
```

After this, the `./configure` script will have adapted the Makefile to use the proper libraries, compiler options and installation directory. If you have enabled some optional features of Gazebo, `./configure` script would include them into the Makefile as well.

Now you just need to run:

```
linux:~> make
linux:~> make install
```

`make` will try to compile all the parts of Gazebo. Depending on the speed of your machine, this process can take a while. `make install` will put all the binaries into the directory specified earlier. If everything has been properly set up and `make` and `make install` have been completed without errors, the installation process is finished.

Terrain Builder Utility of Gazebo: Gzbuilder

Gzbuilder is a terrain-builder part of Gazebo. *Gzbuilder* can build different terrains depending on kind of data provided as its input. It can create a 3D terrain of a floor plan or an outdoor map if a Digital Elevation Model (DEM) is passed as its input. Output is `*.gzb` file that is usable by Gazebo. Basic syntax for *Gzbuilder* is the following:

```
linux:~> gzbuilder [options] -i <inputfile> -o <outputfile>
```

where the options are:

-i <filename> Input terrain file.
-o <filename> Output Gazebo terrain file.
-e <double> Acceptable error bound on terrain approximation (meters). Set this to zero for no approximation.
-h <double> Horizontal scaling factor (m).
-v <double> Vertical scaling factor (m).
-n Normalize Z-values before scaling.
-x <double> X offset (m).
-y <double> Y offset (m).
-z <double> Z offset (m).
-u <int> UTM zone. Default is 11
-s <double> X size of texture (meters)
-t <double> Y size of texture (meters)

Here is an example command that creates a 3D map based on the image file `hospital_section.png`:

```
linux:~> gzbuilder –i hospital_section.png –o hospital.gzb –n –v 2.4 –h 0.1 –e 0.1
```

where the options are:

–n : normalizes the input image in the range 0 for black pixels and 1 for white pixels.
–v 2.4 : scales heights so the white pixels correspond to an elevation of 2.4 meters
–h 0.1 : scales the image so the distance between each pixel on the map correspond to distance of 0.1 meter
–e 0.1 : allows *Gzbuilder* to create a terrain with maximum elevation error of 0.1 meter.

NOTE1: *Gzbuilder* can take any file in format supported by GDAL library as an input. For the full list of supported formats visit:
www.gdal.org/formats_list.html

NOTE2: The version of `hospital_section.png` given earlier in the class had black pixels representing the walls and white pixels representing the free area. However, Gzbuilder needs this to be reversed. The reversed version of `hospital_section.png` can be found in `/research/playerstage/examples/` directory.

How to use Gazebo

Creating a world file:

As mentioned above, Gazebo is a multi-robot simulator for both indoor and outdoor environments. Gazebo takes a `*.world` file as an input. This is an XML file containing the description of the world. The following header block is required at the beginning of each `*.world` file.

```
<?xml version="1.0"?>
<gz:world xmlns:gz="http://playerstage.sourceforge.net/gazebo/xmlschema/#gz"
xmlns:model="http://playerstage.sourceforge.net/gazebo/xmlschema/#model"
xmlns:sensor="http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor"
xmlns:window="http://playerstage.sourceforge.net/gazebo/xmlschema/#window"
xmlns:param="http://playerstage.sourceforge.net/gazebo/xmlschema/#params"
xmlns:ui="http://playerstage.sourceforge.net/gazebo/xmlschema/#params">

****** definition of the world is entered here ********

</gz:world>
```

The following code is an example of the definition of the world. This world contains one *Pioneer2DX* robot equipped with a laser sensor.

```
  <model:Pioneer2DX>
    <id>robot1</id>
    <xyz>0 0 0.200</xyz>
    <model:SickLMS200>
      <id>laser1</id>
      <xyz>0.0 0.0 0.00</xyz>
    </model:SickLMS200>
  </model:Pioneer2DX>
```

To run the simulation with a world file (assume the file is named `example.world`) you need to run the following command:

```
linux:~> gazebo example.world
```

In addition to describing robots and sensors in the environment (world), you can also describe many other models such as Terrain, GroundPlane, LightSource, ObserverCam, etc. Since Gazebo simulates a realistic environment, in the world file, you can also describe global parameters such as gravity force (gravity), skyColor, utmOffset, etc.

The last page of this tutorial has the instructions how to download the `hospital.world` file. This file contains definitions of several Gazebo models and parameters. I would suggest you to go to the official Gazebo website and obtain the full descriptions and definitions of these models.

After creating a world file and running the simulation, you need to be able to control the robots. There are two ways to control robots and sensors in Gazebo. One way is by working with the *Player*. The other way is by using *libgazebo* library.

Working with the Player:

Using the Player with Gazebo is very simple. Gazebo interacts with Player through *libgazeboplugin*, and Player provides full control over the physical sensors and actuators on the robots described in the world (environment). Player needs to know which sensors robot is using and this is defined in the `*.cfg` file. The following `*.cfg` file correseponds to the `*.world` file described earlier:

```
driver
(
  name "gazebo"
  provides ["simulation:0"]
  plugin "libgazeboplugin"
  server_id "default"
)
driver
(
  name "gazebo"
  provides ["position2d:0"]
  gz_id "robot1"
)
driver
(
  name "gazebo"
  provides ["laser:0"]
  gz_id "laser1"
)
```

After creating the `*.cfg` file, you need to:

1. run Gazebo with the `*.world` file
2. run Player with the `*.cfg` file
3. run the client program

After completing these steps, you are done ☺

Using libgazebo library:

External programs can use libgazebo to interact with the Gazebo simulator. Libgazebo is a simple C library that allows other programs to peek and poke into a running simulation. Through this library, programs may read sensor data from and send commands to simulated devices. The Player device server, for example, uses libgazebo in this way. However, I would suggest you to stick with the Player since that is the easiest way to control the robot.

Code Examples:

Example files on how to use Gzbuilder, Player and Gazebo together as well as libgazebo can be found in: `/research/playerstage/examples` directory.

In this directory, you can find `hospital_section.png` (needed by Gzbuilder to create a gzb file), `hospital.world` and `hospital.cfg`. In addition to these files, there is a Makefile that will compile `laseravoidobstacles.cc` program and `simple.c` program. You can run `laseravoidobstacles.cc` together with Player and Gazebo, while `simple.c` is an example on how to use Gazebo with libgazebo library.

Important note:

After Gazebo version 0.5, Gazebo is available in two different modes. One is a console-mode (no GUI) and the other one is a GUI mode.

If you want to run GUI version of Gazebo, you need to run *wxgazebo*:

```
linux:~> wxgazebo example.world
```

If you want to run the console-mode, you need to run *gazebo*:

```
linux:~> gazebo example.world
```

The console-mode can be useful if you are running Gazebo remotely or you are running automated tests or batch experiments.

GUI version of Gazebo (*wxgazebo*) is a python script that simply wraps around the underlying console-mode server (running the latter as a child process).