GAZEBO

Amirreza Kabiri
Fatemeh Pahlevan Aghababa

Autumn 2017

History

- Started in fall 2002 in USC by Dr. Andrew Howard and his student Nate Koenig as a complementary simulator to Stage, Nate continue to develop during his PhD
- In 2009, John Hsu from Willow integrate ROS and PR2 into Gazebo
- In 2011 Willow Garage started financial support
- In 2012 OSRF became steward of project
- In July 2013 OSRF used gazebo in Virtual Robotics Challenge and DARPA Robotics Challenge

About

- It is licensed under Apache 2.0
- It uses MAJOR.MINOR.PATCH versioning system
- Major when incompatible
- Minot when functionality changes with b/w compatibility
- Patch when b/w compatible bug fixes release
- It uses Tick-tock Release Cycle

Measurement	v1.9	v2.2	v3.0	v4.0	v5.0	v6.0	v7.0
Lines of code	186k	197k	214k	217k	231k	266k	298k
Lines of comments	57k	63k	68k	69k	75k	89k	99k
Test function coverage	45.7%	47.1%	41.3%	40.6%	48.7%	47.9%	52.9%
Test branch coverage	32.2%	35.5%	29.2%	27.6%	38.0%	39.1%	44.5%
Tests	168	376	524	542	743	901	1222

FEATURES

- Dynamics Simulation (Access multiple high-performance physics engines including ODE, Bullet, Simbody, and DART.)
- Advanced 3D Graphics (Utilizing OGRE, Gazebo provides realistic rendering of environments including high-quality lighting, shadows, and textures.)
- Sensors and Noise (Generate sensor data, optionally with noise, from laser range finders, 2D/3D cameras, Kinect style sensors, contact sensors, force-torque, and more.)
- Plugins (Develop custom plugins for robot, sensor, and environmental control. Plugins provide direct access to Gazebo's API.)

FEATURES

- Robot Models (Many robots are provided including PR2, Pioneer2 DX, iRobot Create, and TurtleBot. Or build your own using SDF.)
- TCP/IP Transport (Run simulation on remote servers, and interface to Gazebo through socket-based message passing using Google Protobufs.)
- Cloud Simulation (Use CloudSim to run Gazebo on Amazon, Softlayer, or your own OpenStack instance.)
- Command Line Tools (Extensive command line tools facilitate simulation introspection and control.)

INSTALLATION

- Instructions to install Gazebo on all the platforms supported: Ubuntu, Debian, Fedora, Arch, Gentoo, Mac, Windows is still under development.
- Installation on Ubuntu

One line

```
curl -sSL http://get.gazebosim.org | sh
```

From repository

```
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
```

```
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
```

```
sudo apt-get update
sudo apt-get install gazebo7
# For developers that work on top of Gazebo, one extra package
sudo apt-get install libgazebo7-dev
```

Gazebo Architecture

- **Run Gazebo!**

```
gazebo
```

- **Run Gazebo with a robot**

```
gazebo worlds/pioneer2dx.world
```

(SDF file (Simulation Description Format))

- **Where are the worlds located?**

```
/usr/share/gazebo-version/worlds
```

- **Client and server separation**

```
gzserver gzclient(Qt based)
```

Environment Variables

- **GAZEBO_MODEL_PATH**: colon-separated set of directories where Gazebo will search for models
- **GAZEBO_RESOURCE_PATH**: colon-separated set of directories where Gazebo will search for other resources such as world and media files.
- **GAZEBO_MASTER_URI**: URI of the Gazebo master. This specifies the IP and port where the server will be started and tells the clients where to connect to.
- **GAZEBO_PLUGIN_PATH**: colon-separated set of directories where Gazebo will search for the plugin shared libraries at runtime.
- **GAZEBO_MODEL_DATABASE_URI**: URI of the online model database where Gazebo will download models from.

Environment Variables

- These default values are stored in source
  `<install_path>/share/gazebo/setup.sh`

```
gzserver worlds/empty.world
```

- Example of loading a plugin on the command line:

```
gzserver -s <plugin_filename> <world_file>
```

- The same mechanism is used by the graphical client:

```
gzclient -g <plugin_filename>
```

COMMUNICATION

- **Communication Between Processes**

  Google Protobuf for the message serializationand boost::ASIO for the transport mechanism

- **Gazebo Master**

  This is essentially a topic name server. It provides namelookup, and topic management. A single master can handle multiple physics simulations, sensor generators, and GUIs.

- **Communication Library**
  - Dependencies: Protobuf and boost::ASIO
  - External API:
  - Internal API: None
  - Advertised Topics: None
  - Subscribed Topics: None
  - It currently supports only publish/subscribe

Physics Library

- **Dependencies:** Dynamics engine (with internal collision detection)
- **External API:** Provides a simple and generic interface to physics simulation
- **Internal API:** Defines a fundamental interface to the physics library for 3rd party dynamic engines.
- The physics library provides a simple and generic interface to fundamental simulation components, including rigid bodies, collision shapes, and joints for representing articulation constraints.

Rendering Library

- **Dependencies:** OGRE
- **External API:** Allows for loading, initialization, and scene creation
- **Internal API:** Store metadata for visualization, call the OGRE API for rendering.
- The rendering library uses OGRE to provide a simple interface for rendering 3D scenes to both the GUI and sensor libraries. It includes lighting, textures, and sky simulation. It is possible to write plugins for the rendering engine.

Sensor Generation

- **Dependencies:** Rendering Library, Physics Library
- **External API:** Provide functionality to initialize and run a set of sensors
- **Internal API:** TBD
- The sensor generation library implements all the various types of sensors, listens to world state updates from a physics simulator and produces output specified by the instantiated sensors.

GUI

- **Dependencies**: Rendering Library, Qt
- **External API:** None
- **Internal API:** None

- The GUI library uses Qt to create graphical widgets for users to interact with the simulation. The user may control the flow of time by pausing or changing time step size via GUI widgets. The user may also modify the scene by adding, modifying, or removing models. Additionally there are some tools for visualizing and logging simulated sensor data.

More..

- **Plugins**

  The physics, sensor, and rendering libraries support plugins. These plugins provide users with access to the respective libraries without using the communication system.

- **Capture screenshots**

  The captured image will be saved to ~/.gazebo/pictures with a timestamped filename.

BUILD ROBOT

- **The Model Database Repository**

  The model database is a bitbucket repository

- **Model.config**

```
<?xml version="1.0"?>
  <model>
    <name>My Model Name</name>
    <version>1.0</version>
    <sdf version='1.5'>model.sdf</sdf>
    <author>
      <name>My name</name>
      <email>name@email.address</email>
    </author>
    <description>
      A description of the model
    </description>
  </model>
```

SDF (Simulation Description Format)

Components of a SDF Models

- **Links**:A link contains the physical properties of one body of the model. This can be a wheel, or a link in a joint chain. Each link may contain many collision and visual elements. Try to reduce the number of links in your models in order to improve performance and stability.

- **Collision:** A collision element encapsulates a geometry that is used to collision checking. This can be a simple shape (which is preferred), or a triangle mesh (which consumes greater resources). A link may contain many collision elements.

- **Visual:** A visual element is used to visualize parts of a link. A link may contain o or more visual elements.

SDF(Simulation Description Format)

- **Inertial:** The inertial element describes the dynamic properties of the link, such as mass and rotational inertia matrix.
- **Sensor:** A sensor collects data from the world for use in plugins. A link may contain 0 or more sensors.
- **Joints:** A joint connects two links. A parent and child relationship is established along with other parameters such as axis of rotation, and joint limits.
- **Plugins:** A plugin is a shared library created by a third party to control a model

MAKE A MOBILE ROBOT

- **Setup your model directory**
  - Read through the Model Database documentation. You will be creating your own model, which must follow the formatting rules for the Gazebo Model Database directory structure.

1. Create a model directory

```
mkdir -p ~/.gazebo/models/my_robot
```

2. Create a model config file

```
gedit ~/.gazebo/models/my_robot/model.config
```

```
<?xml version="1.0"?>
<model>
  <name>My Robot</name>
  <version>1.0</version>
  <sdf version='1.4'>model.sdf</sdf>
  <author>
    <name>My Name</name>
    <email>me@my.email</email>
  </author>
  <description>
    My awesome robot.
  </description>
</model>
```

MAKE A MOBILE ROBOT

3.create model.sdf file

~/.gazebo/models/my_robot/model.sdf

```
gedit ~/.gazebo/models/my_robot/model.sdf
    <?xml version='1.0'?>
        <sdf version='1.4'>
        <model name="my_robot">
        </model>
        </sdf>
```

- **Build the Model's Structure**

- This step will create a rectangular base with two wheels.
- It is important to start simple, and build up a model in steps. The first step is to layout the basic shapes of the model. To do this we will make our model static, which means it will be ignored by the physics engine. As a result the model will stay in one place and allow us to properly align all the components.

Make a Mobile Robot

1. Make the model static by adding a <static>true</static> element to the ~/.gazebo/models/my_robot/model.sdf file:

```
Do not copy. The following is for reference purposes.
<?xml version='1.0'?>
<sdf version='1.4'>
  <model name="my_robot">

    <static>true</static>

Do not copy. The following is for reference purposes.
  </model>
</sdf>
```

Make a Mobile Robot

2. Add the rectangular base by editing the ~/.gazebo/models/my_robot/model.sdf file:

Do not copy. The following is for reference purposes.

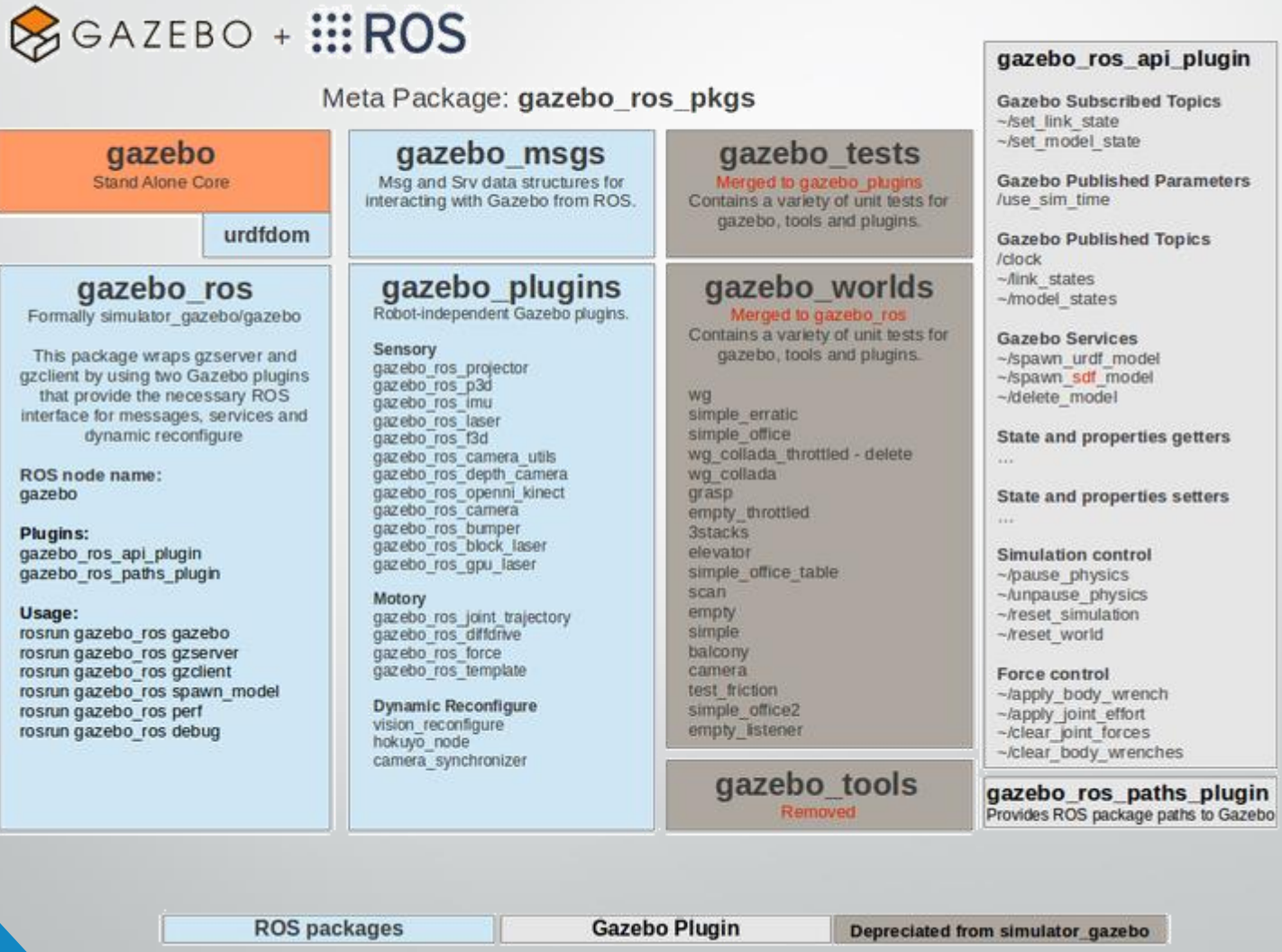
```
<?xml version='1.0'?>
  <sdf version='1.4'>
    <model name="my_robot">
    <static>true</static>

      <link name='chassis'>
        <pose>0 0 .1 0 0 0</pose>

        <collision name='collision'>
          <geometry>
            <box>
              <size>.4 .2 .1</size>
            </box>
          </geometry>
        </collision>

        <visual name='visual'>
          <geometry>
            <box>
              <size>.4 .2 .1</size>
            </box>
          </geometry>
        </visual>
      </link>
```

ROS INTEGRATION

- To achieve ROS integration with stand-alone Gazebo, a set of ROS packages named gazebo_ros_pkgs provides wrappers around the stand-alone Gazebo. They provide the necessary interfaces to simulate a robot in Gazebo using ROS messages, services and dynamic reconfigure Some features of gazebo_ros_pkgs:

  - Supports a stand alone system dependency of Gazebo, that has no ROS bindings on its own
  - Builds with catkin
  - Treats URDF and SDF as equally as possible
  - Reduces code duplication with Gazebo
  - Improves out of the box support for controllers using ros_control
  - Integrates real time controller efficiency improvements from the DARPA Robotics Challenge
  - Cleans up old code from previous versions of ROS and Gazebo

GAZEBO_ROS_PKGS INTERFACE



Running Gazebo

- The names of the ROS nodes to launch Gazebo have changed slightly to coincide with the Gazebo executable names:

  rosrun gazebo_ros gazebo launch both the Gazebo server and GUI.
  rosrun gazebo_ros gzclient launch the Gazebo GUI.
  rosrun gazebo_ros gzserver launch the Gazebo server.

- Available nodes to run:

  rosrun gazebo_ros gazebo
  rosrun gazebo_ros gzserver
  rosrun gazebo_ros gzclient
  rosrun gazebo_ros spawn_model
  rosrun gazebo_ros perf
  rosrun gazebo_ros debug

Using roslaunch to open world models

- The roslaunch tool is the standard method for starting ROS nodes and bringing up robots in ROS. To start an empty Gazebo world similar to the rosrun command in the previous tutorial, simply run

```
roslaunch gazebo_ros empty_world.launch
```

roslaunch Arguments

- You can append the following arguments to the launch files to change the behavior of Gazebo:
  - ✓ **Paused**: Start Gazebo in a paused state (default false)
  - ✓ **use_sim_time**: Tells ROS nodes asking for time to get the Gazebo-published simulation time, published over the ROS topic /clock (default true)
  - ✓ **Gui**: Launch the user interface window of Gazebo (default true)
  - ✓ **Headless**: Disable any function calls to simulator rendering (Ogre) components. Does not work with gui:=true (default false)
  - ✓ **Debug**: Start gzserver (Gazebo Server) in debug mode using gdb (default false)

ROSLAUNCH COMMAND

- Normally the default values for these arguments are all you need, but just as an example:

```
roslaunch gazebo_ros empty_world.launch paused:=true
use_sim_time:=false gui:=true throttled:=false headless:=false
debug:=true
```

- **Launching Other Demo Worlds**

- Other demo worlds are already included in the gazebo_ros package, including:

```
roslaunch gazebo_ros willowgarage_world.launch
roslaunch gazebo_ros mud_world.launch
roslaunch gazebo_ros shapes_world.launch
roslaunch gazebo_ros rubble_world.launch
```