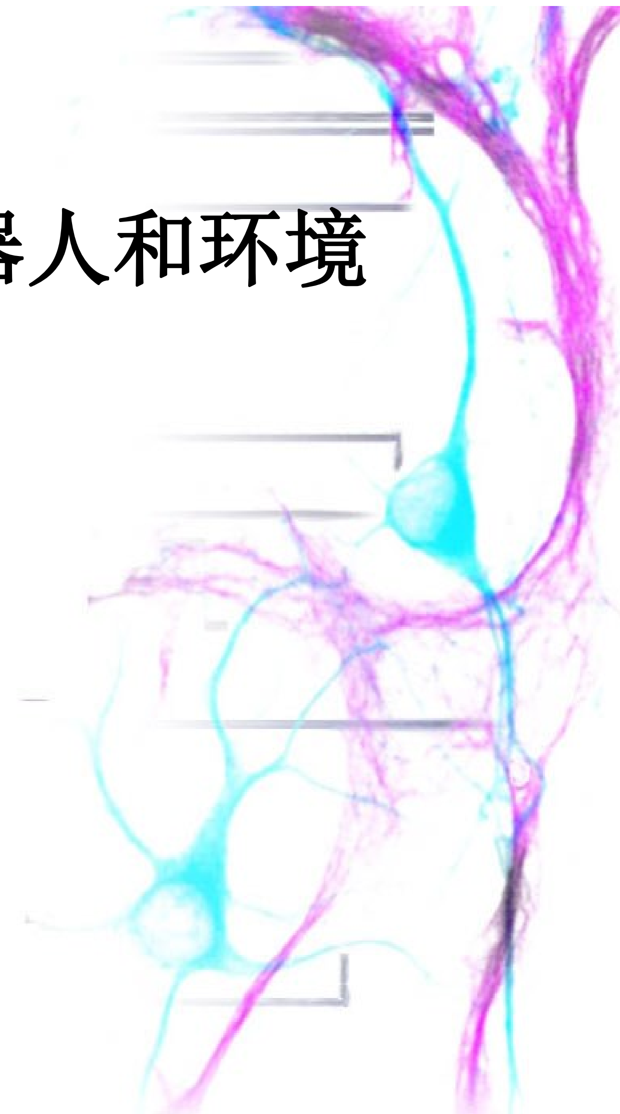
采用ROS和Gazebo模拟机器人和环境

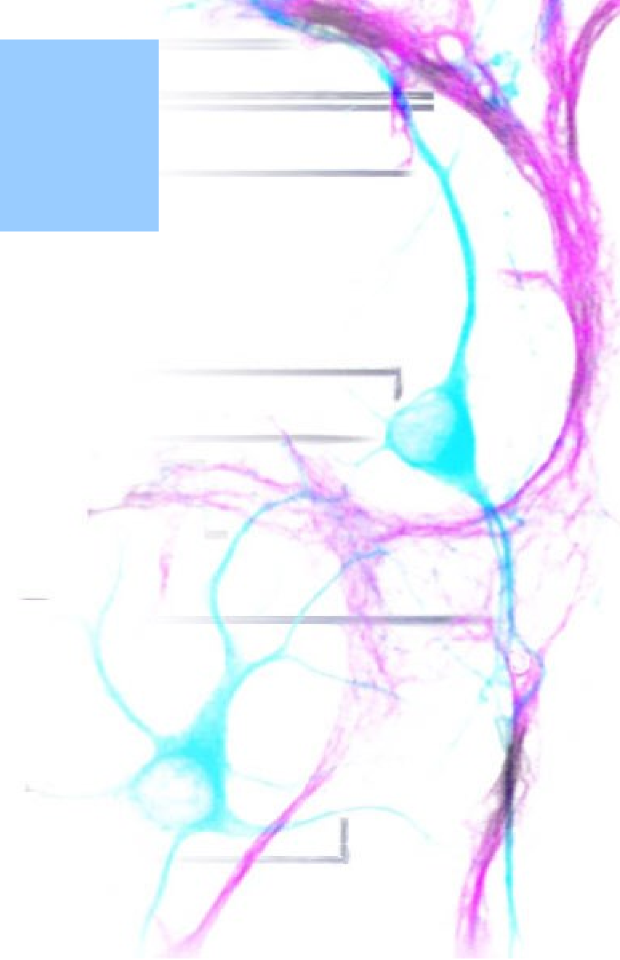
杰克科技 中央研究院

2024.6



6/2024 Hangzhou

Agenda



- Gazebo 3D simulator
- Model SDF files
- Gazebo and ROS integration
- TurtleBot simulation

Gazebo

- A multi-robot simulator
- Like Stage, it is capable of simulating a population of robots, sensors and objects, but does so in 3D
- Includes an accurate simulation of rigid-body physics and generates realistic sensor feedback
- Allows code designed to operate a physical robot to be executed in an artificial environment
- Gazebo is under active development at the OSRF (Open Source Robotics Foundation)

Gazebo

Gazebo Installation

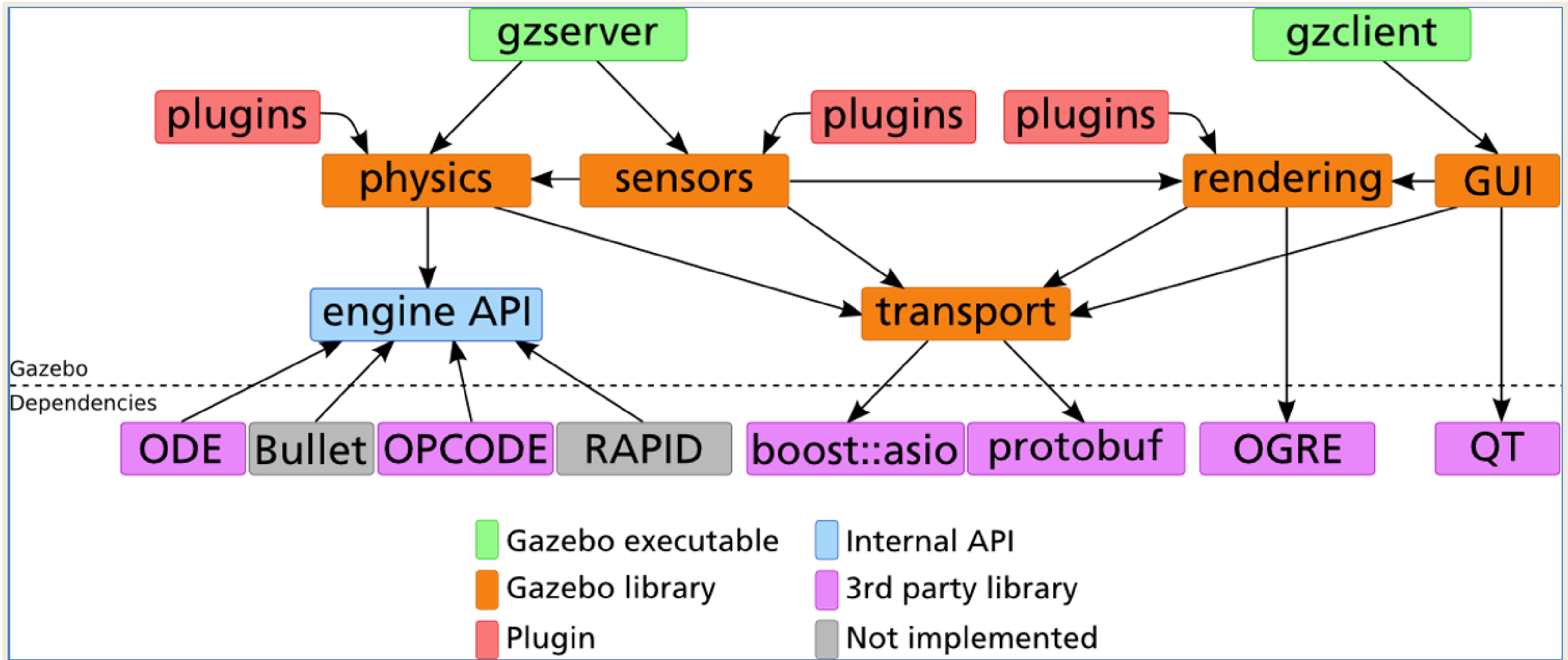
- ROS Melodic comes with Gazebo V9.0
- Gazebo home page - http://gazebosim.org/
- Gazebo tutorials - http://gazebosim.org/tutorials

Gazebo Architecture

Gazebo consists of two processes:

- Server: Runs the physics loop and generates sensor data
  - Executable: gzserver
  - Libraries: Physics, Sensors, Rendering, Transport
- Client: Provides user interaction and visualization of a simulation.
  - Executable: gzclient
  - Libraries: Transport, Rendering, GUI

Gazebo Architecture



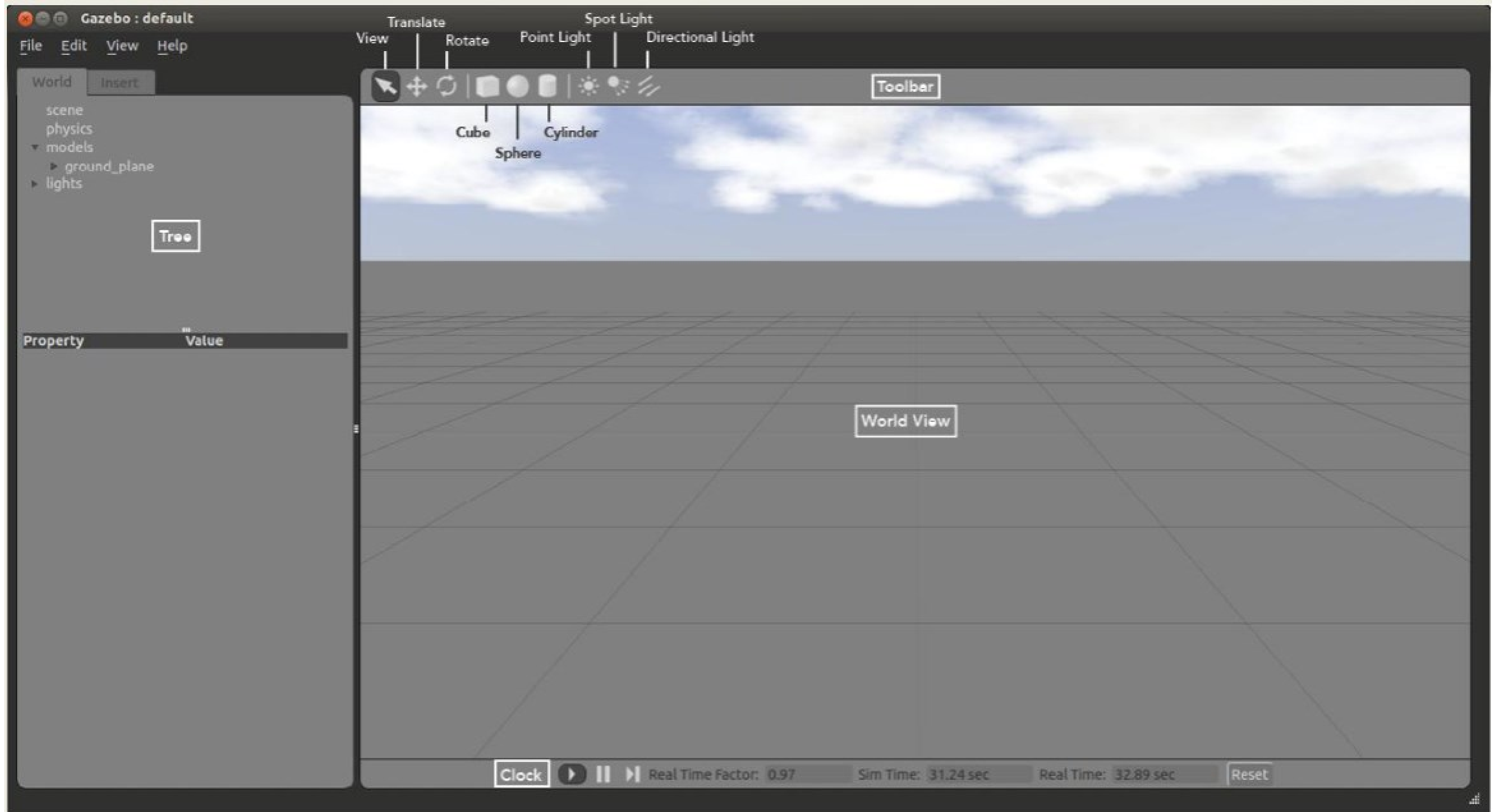
Running Gazebo from ROS

• To launch Gazebo type:

```
$ rosrun gazebo_ros gazebo
```

• Note: When you first launch Gazebo it may take a few minutes to update its model database

Gazebo User Interface



The World View

- The World View displays the world and all of the models therein
- Here you can add, manipulate, and remove models
- You can switch between View, Translate and Rotate modes of the view in the left side of the Toolbar

View Mode

Translate	Left-press + drag
Orbit	Middle-press + drag
Zoom	Scroll wheel
Accelerated Zoom	Alt + Scroll wheel
Jump to object	Double-click object
Select object	Left-click object

Translate Mode

Translate	Left-press + drag
Translate (x-axis)	Left-press + X + drag
Translate (y-axis)	Left-press + Y + drag
Translate (z-axis)	Left-press + Z + drag

(Orbit & Zoom work in this mode, as well)

Rotate Mode

Rotate (spin) object	Left-press + drag
Rotate (x-axis)	Left-press + X + drag
Rotate (y-axis)	Left-press + Y + drag
Rotate (z-axis)	Left-press + Z + drag

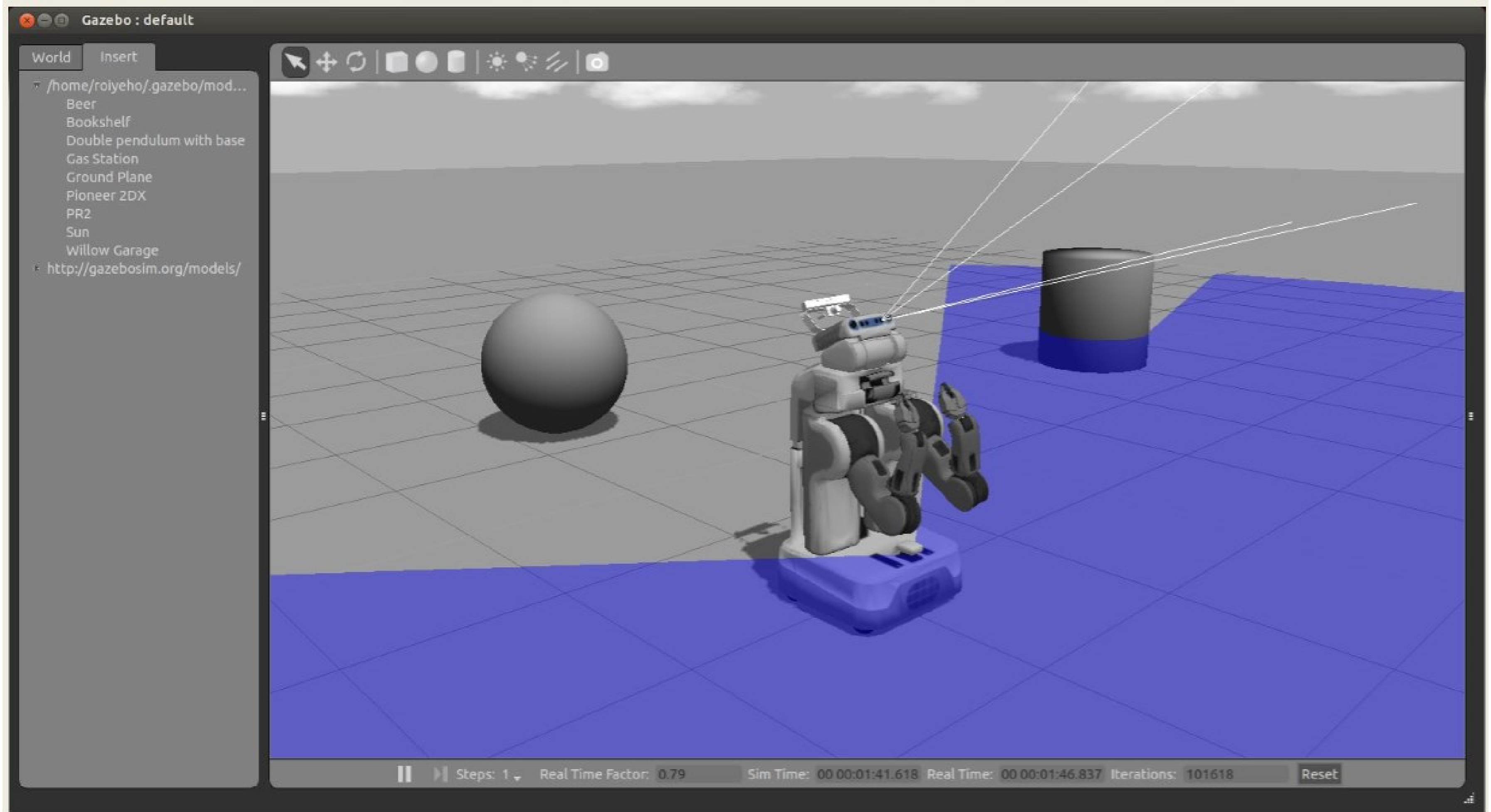
(Orbit & Zoom work in this mode, as well)

Add A Model

To add a model to the world:

- left-click on the desired model in the Insert Tab on the left side
- move the cursor to the desired location in World View
- left-click again to release
- Use the Translate and Rotate modes to orient the model more precisely

Inserting PR2 Robot



Models Item

- The models item in the world tab contains a list of all models and their links

- Right-clicking on a model in the Models section gives you three options:

  – Move to – moves the view to be directly in front of that model
  – Follow
  – View – allows you to view different aspects of the model, such as Wireframe, Collisions, Joints
  – Delete – deletes the model

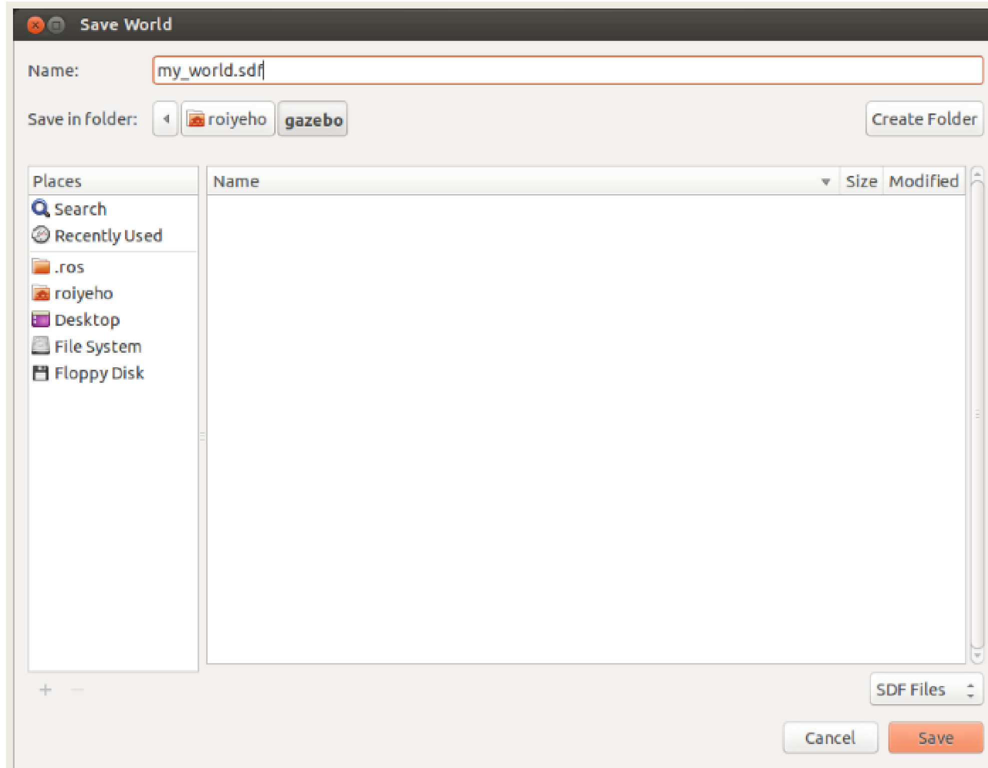
Clock

- You can start, pause and step through the simulation with the clock
- It is located at the bottom of the World View

- **Real Time Factor**: Displays how fast or slow the simulation is running in comparison to real time
  - A factor less than 1.0 indicates simulation is running slower than real time
  - Greater then 1.0 indicates faster than real time

Saving a World

- Once you are happy with a world it can be saved through the File->Save As menu.
- Enter my_world.sdf as the file name and click OK



Loading a World

- A saved world may be loaded on the command line:

```
$ gazebo my_world.sdf
```

- The filename must be in the current working directory, or you must specify the complete path

Simulation Description Format (SDF)

- **SDF** is an XML file that contains a complete description for everything from the world level down to the robot level, including:

– **Scene**: Ambient lighting, sky properties, shadows.
– **Physics**: Gravity, time step, physics engine.
– **Models**: Collection of links, collision objects, joints, and sensors.
– **Lights**: Point, spot, and directional light sources.
– **Plugins**: World, model, sensor, and system plugins.

- http://gazebosim.org/sdf.html

SDF vs URDF

- **URDF** can only specify the kinematic and dynamic properties of a single robot in isolation
  - URDF can not specify the pose of the robot itself within a world
  - It cannot specify objects that are not robots, such as lights, heightmaps, etc.
  - Lacks friction and other properties

- **SDF** is a complete description for everything from the world level down to the robot leve

World Description File

- The world description file contains all the elements in a simulation, including robots, lights, sensors, and static objects

- This file is formatted using SDF and has a .world extension

- The Gazebo server (gzserver) reads this file to generate and populate a world

Example World Files

- Gazebo ships with a number of example worlds

- World files are found within the /worlds directory of your Gazebo resource path
  - A typical path might be /usr/share/gazebo-2.2

- In gazebo_ros package there are built-in launch files that load some of these world files

- For example, to launch willowgarage_world type:

```
$ roslaunch gazebo_ros willowgarage_world.launch
```

willowgarage.world

```xml
<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <include>
      <uri>model://sun</uri>
    </include>
    <include>
      <uri>model://willowgarage</uri>
    </include>
  </world>
</sdf>
```

- In this world file snippet you can see that three models are referenced
- The three models are searched for within your local Gazebo Model Database
- If not found there, they are automatically pulled from Gazebo’s online database

Models Item

- In gazebo_ros package there are built-in launch files that load some of these world files
- For example, to launch willowgarage_world type:

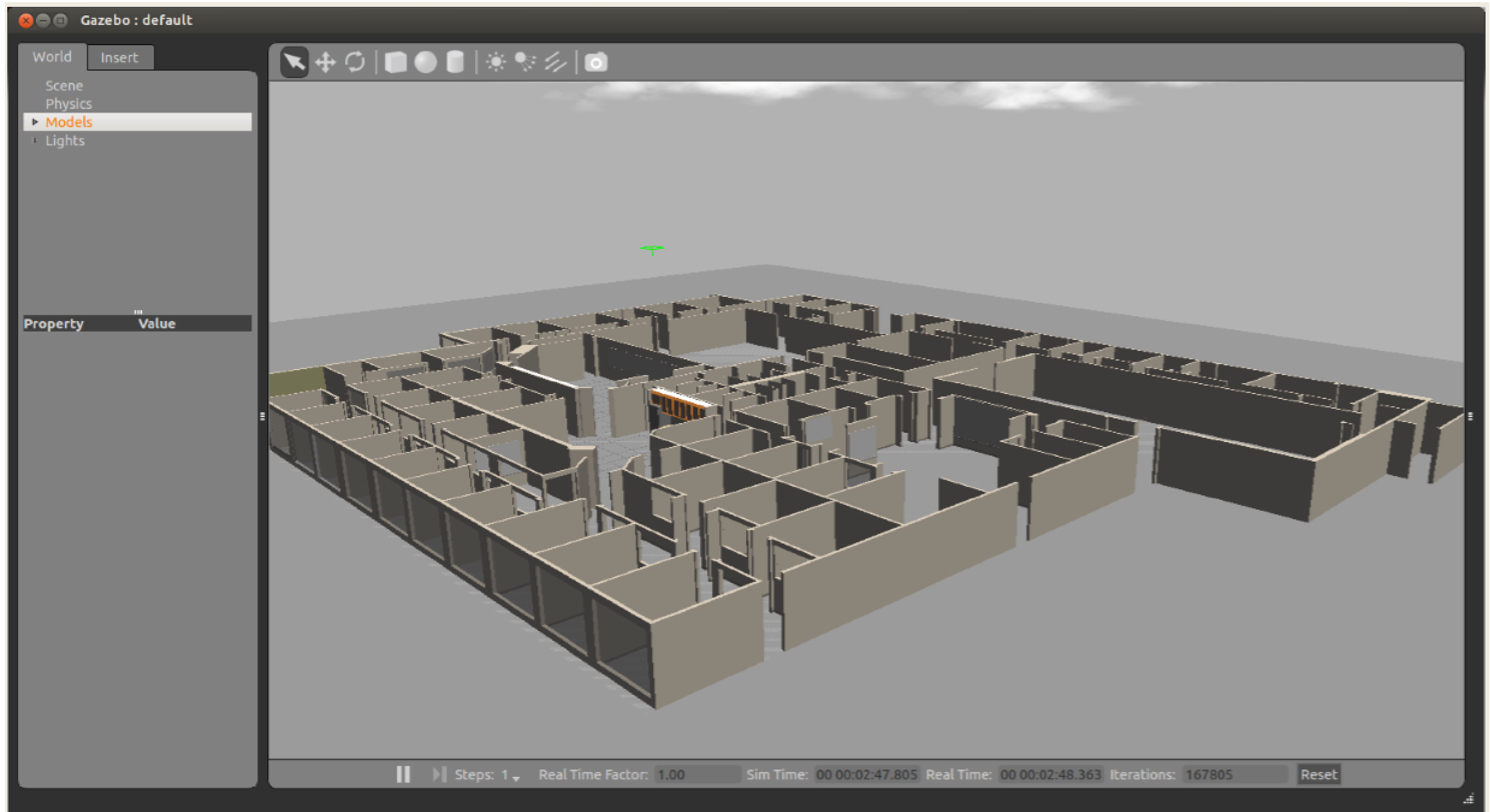
```
$ roslaunch gazebo_ros willowgarage_world.launch
```

willowgarage_world.launch

```
<launch>
 <!-- We resume the logic in empty_world.launch, changing only the name of the world to
be launched -->
 <include file="$(find gazebo_ros)/launch/empty_world.launch">
   <arg name="world_name" value="worlds/willowgarage.world"/> <!-- Note: the
world_name is with respect to GAZEBO_RESOURCE_PATH environmental variable -->
   <arg name="paused" value="false"/>
   <arg name="use_sim_time" value="true"/>
   <arg name="gui" value="true"/>
   <arg name="headless" value="false"/>
   <arg name="debug" value="false"/>
 </include>
</launch>
```

- This launch file inherits most of the necessary functionality from empty_world.launch
- The only parameter we need to change is the world_name parameter, substituting the empty.world world file with willowgarage.world
- The other arguments are simply set to their default values

Willow Garage World



Model Files

• A model file uses the same SDF format as world files, but contains only a single <model> tag

• Once a model file is created, it can be included in a world file using the following SDF syntax:

```
<include filename="model_file_name"/>
```

• You can also include any model from the online database and the necessary content will be downloaded at runtime

willowgarage Model SDF File

```
<?xml version="1.0" ?>
<sdf version="1.4">
 <model name="willowgarage">
  <static>true</static>
  <pose>-20 -20 0 0 0 0</pose>
  <link name="walls">
   <collision name="collision">
    <geometry>
     <mesh>
      <uri>model://willowgarage/meshes/willowgarage_collision.dae</uri>
     </mesh>
    </geometry>
   </collision>
   <visual name="visual">
    <geometry>
     <mesh>
      <uri>model://willowgarage/meshes/willowgarage_visual.dae</uri>
     </mesh>
    </geometry>
    <cast_shadows>false</cast_shadows>
   </visual>
  </link>
 </model>
</sdf>
```

Components of 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, visual and sensor elements
- 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).
- Visual: A visual element is used to visualize parts of a link.
- 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.
- 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 shared library created by a 3rd party to control a model.

Gazebo + ROS Integration

- ROS integrates closely with Gazebo through the **gazebo_ros** package
- This package provides a Gazebo plugin module that allows bidirectional communication between Gazebo and ROS
- Simulated sensor and physics data can stream from Gazebo to ROS, and actuator commands can stream from ROS back to Gazebo.
- By choosing consistent names and data types for these data streams, it is possible for Gazebo to exactly match the ROS API of a robot

Gazebo + ROS Integration

Meta Package: **gazebo_ros_pkgs**

gazebo
Stand Alone Core

urdfdom

gazebo_msgs
Msg and Srv data structures for interacting with Gazebo from ROS.

gazebo_tests
Merged to gazebo_plugins
Contains a variety of unit tests for gazebo, tools and plugins.

gazebo_ros
Formally simulator_gazebo/gazebo

This package wraps gzserver and gzclient by using two Gazebo plugins that provide the necessary ROS interface for messages, services and dynamic reconfigure

ROS node name:
gazebo

Plugins:
gazebo_ros_api_plugin
gazebo_ros_paths_plugin

Usage:
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

gazebo_plugins
Robot-independent Gazebo plugins.

Sensory
gazebo_ros_projector
gazebo_ros_p3d
gazebo_ros_imu
gazebo_ros_laser
gazebo_ros_f3d
gazebo_ros_camera_utils
gazebo_ros_depth_camera
gazebo_ros_openni_kinect
gazebo_ros_camera
gazebo_ros_bumper
gazebo_ros_block_laser
gazebo_ros_gpu_laser

Motory
gazebo_ros_joint_trajectory
gazebo_ros_diffdrive
gazebo_ros_force
gazebo_ros_template

Dynamic Reconfigure
vision_reconfigure
hokuyo_node
camera_synchronizer

gazebo_worlds
Merged to gazebo_ros
Contains a variety of unit tests for gazebo, tools and plugins.

wg
simple_erratic
simple_office
wg_collada_throttled - delete
wg_collada
grasp
empty_throttled
3stacks
elevator
simple_office_table
scan
empty
simple
balcony
camera
test_friction
simple_office2
empty_listener

gazebo_tools
Removed

gazebo_ros_api_plugin

Gazebo Subscribed Topics
~/set_link_state
~/set_model_state

Gazebo Published Parameters
/use_sim_time

Gazebo Published Topics
/clock
~/link_states
~/model_states

Gazebo Services
~/spawn_urdf_model
~/spawn_sdf_model
~/delete_model

State and properties getters
...

State and properties setters
...

Simulation control
~/pause_physics
~/unpause_physics
~/reset_simulation
~/reset_world

Force control
~/apply_body_wrench
~/apply_joint_effort
~/clear_joint_forces
~/clear_body_wrenches

gazebo_ros_paths_plugin
Provides ROS package paths to Gazebo

ROS packages | Gazebo Plugin | Depreciated from simulator_gazebo

Gazebo ROS Services

```
roiyeho@ubuntu: ~
roiyeho@ubuntu:~$ rosservice list
/gazebo/apply_body_wrench
/gazebo/apply_joint_effort
/gazebo/clear_body_wrenches
/gazebo/clear_joint_forces
/gazebo/delete_model
/gazebo/get_joint_properties
/gazebo/get_link_properties
/gazebo/get_link_state
/gazebo/get_loggers
/gazebo/get_model_properties
/gazebo/get_model_state
/gazebo/get_physics_properties
/gazebo/get_world_properties
/gazebo/pause_physics
/gazebo/reset_simulation
/gazebo/reset_world
/gazebo/set_joint_properties
/gazebo/set_link_properties
/gazebo/set_link_state
/gazebo/set_logger_level
/gazebo/set_model_configuration
/gazebo/set_model_state
/gazebo/set_parameters
/gazebo/set_physics_properties
/gazebo/spawn_gazebo_model
/gazebo/spawn_sdf_model
/gazebo/spawn_urdf_model
/gazebo/unpause_physics
/rosout/get_loggers
/rosout/set_logger_level
roiyeho@ubuntu:~$
```

Gazebo ROS Package Struction

- Typical Gazebo+ROS package structure:
  - The robot's model and description are located in a package named /MYROBOT_description
  - The world and launch files used with Gazebo is located in a package named /MYROBOT_gazebo

- Replace MYROBOT with the name of your robot or something like "test" if you don't have one

Gazebo ROS Package Struction

```
../catkin_ws/src
    /MYROBOT_description
        package.xml
        CMakeLists.txt
        /urdf
            MYROBOT.urdf
        /meshes
            mesh1.dae
            mesh2.dae
            ...
        /materials
        /cad
    /MYROBOT_gazebo
        /launch
            MYROBOT.launch
        /worlds
            MYROBOT.world
        /models
            world_object1.dae
            world_object2.stl
            world_object3.urdf
        /materials
        /plugins
```

Meet Turtlebot

- http://turtlebot.org.
- A minimalist platform for ROS-based mobile robotics education and prototyping.
- Has a small differential-drive mobile base
- Atop this base is a stack of laser-cut “shelves” that provide space to hold a netbook computer and depth camera and other devices
- Does not have a laser scanner – Despite this, mapping and navigation can work quite well for indoor spaces.

Turtlebot Simulation

- To install Turtlebot simulation stack type:

```
$ sudo apt-get install ros-kinetic-turtlebot-gazebo ros-kinetic-turtlebot-apps
```

- To launch a simple world with a Turtlebot, type:

```
$ roslaunch turtlebot_gazebo turtlebot_world.launch
```

turtlebot_world.launch

```
<launch>
 <arg name="world_file" default="$(env TURTLEBOT_GAZEBO_WORLD_FILE)"/>
 ...
 <include file="$(find gazebo_ros)/launch/empty_world.launch">
  <arg name="use_sim_time" value="true"/>
  <arg name="debug" value="false"/>
  <arg name="gui" value="$(arg gui)" />
  <arg name="world_name" value="$(arg world_file)"/>
 </include>

 <include file="$(find turtlebot_gazebo)/launch/includes/$(arg
base).launch.xml">
  <arg name="base" value="$(arg base)"/>
  <arg name="stacks" value="$(arg stacks)"/>
  <arg name="3d_sensor" value="$(arg 3d_sensor)"/>
 </include>
```

turtlebot_world.launch

```
  <node pkg="robot_state_publisher" type="robot_state_publisher"
name="robot_state_publisher">
    <param name="publish_frequency" type="double" value="30.0" />
  </node>

  <!-- Fake laser -->
  <node pkg="nodelet" type="nodelet" name="laserscan_nodelet_manager"
args="manager"/>
  <node pkg="nodelet" type="nodelet" name="depthimage_to_laserscan"
        args="load depthimage_to_laserscan/DepthImageToLaserScanNodelet
laserscan_nodelet_manager">
    <param name="scan_height" value="10"/>
    <param name="output_frame_id" value="/camera_depth_frame"/>
    <param name="range_min" value="0.45"/>
    <remap from="image" to="/camera/depth/image_raw"/>
    <remap from="scan" to="/scan"/>
  </node>
</launch>
```

Spawning URDF Robots

- The spawn_model node in gazebo_ros package makes a service call request to the gazebo ROS node in order to add a custom URDF into Gazebo

- You can use this script in the following way:

```
$ rosrun gazebo_ros spawn_model -file `rospack find [robot_description package]`/urdf/myrobot.urdf -urdf -x 0 -y 0 -z 1 -model myrobot
```

- The x,y,z arguments indicate the initial location of the robot

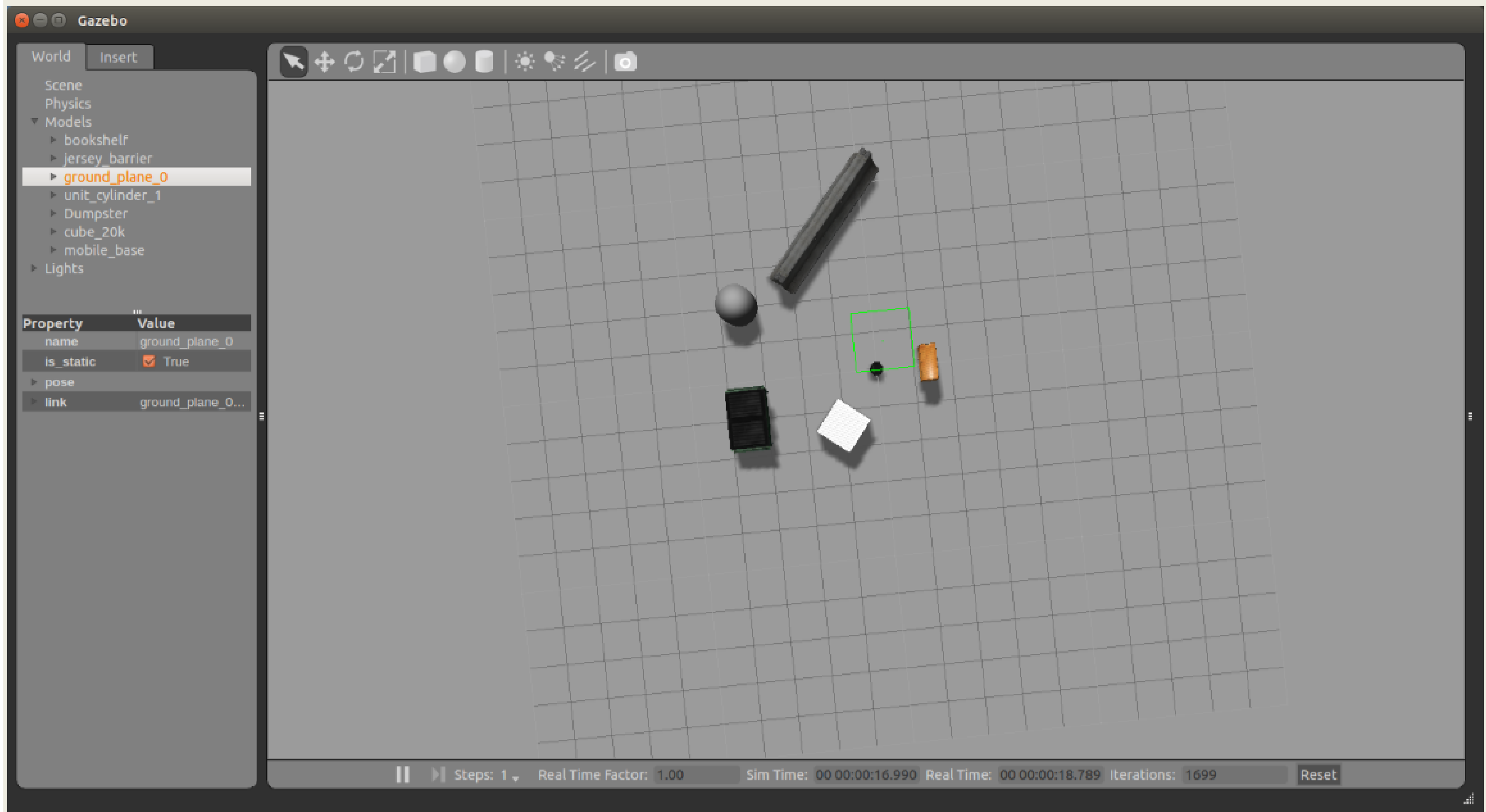
Spawn a URDF Robot

- In the file turtlebot_gazebo/launch/includes/create. launch.xml, the Turtlebot model is spawned:

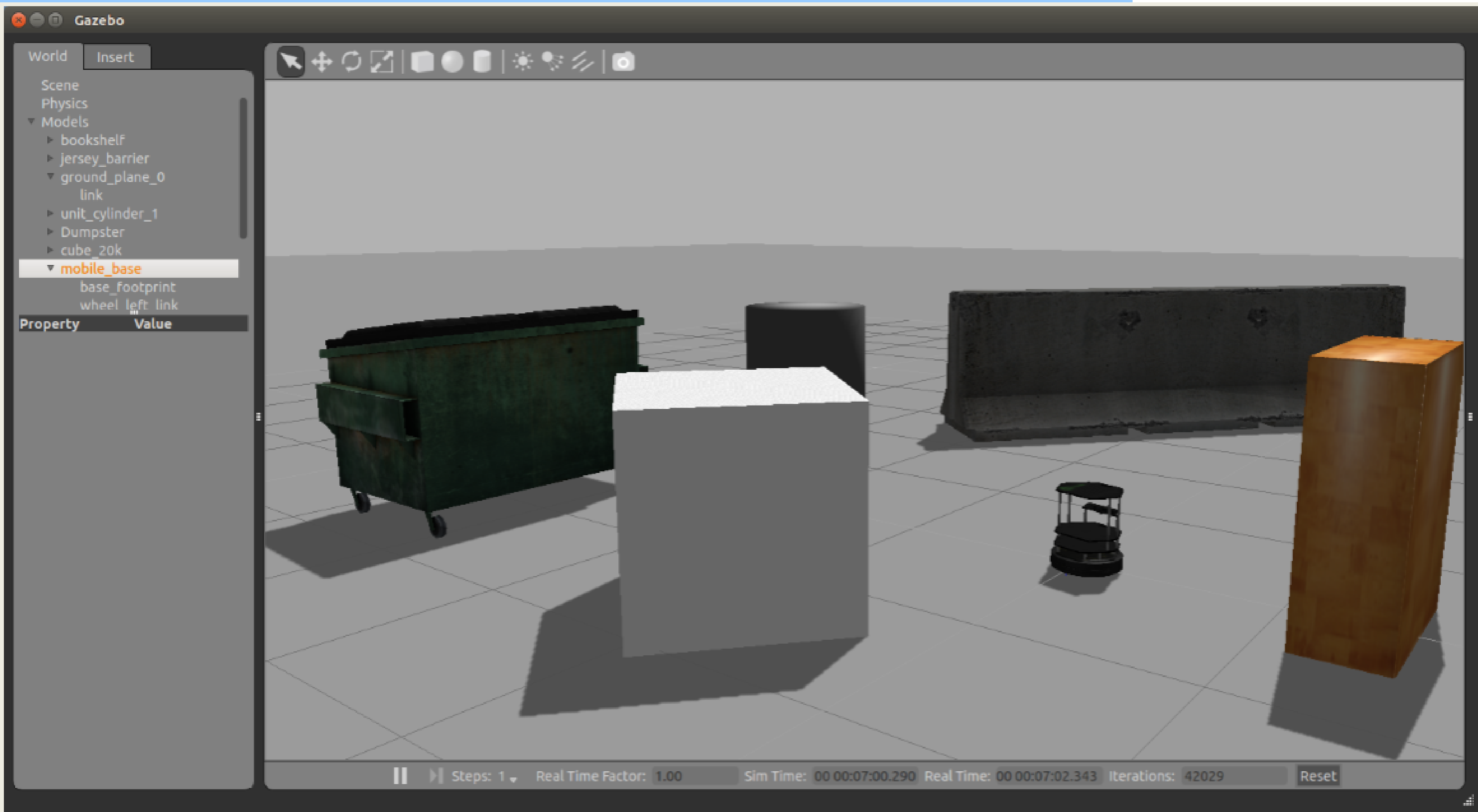
```
<arg name="urdf_file" default="$(find xacro)/xacro.py '$(find
turtlebot_description)/robots/$(arg base)_$(arg stacks)_$(arg
3d_sensor).urdf.xacro'" />
  <param name="robot_description" command="$(arg urdf_file)" />

  <!-- Gazebo model spawner -->
  <node name="spawn_turtlebot_model" pkg="gazebo_ros"
type="spawn_model"
      args="$(optenv ROBOT_INITIAL_POSE) -unpause -urdf -param
robot_description -model turtlebot"/>
```

Turtlebot Simulation



Turtlebot Simulation

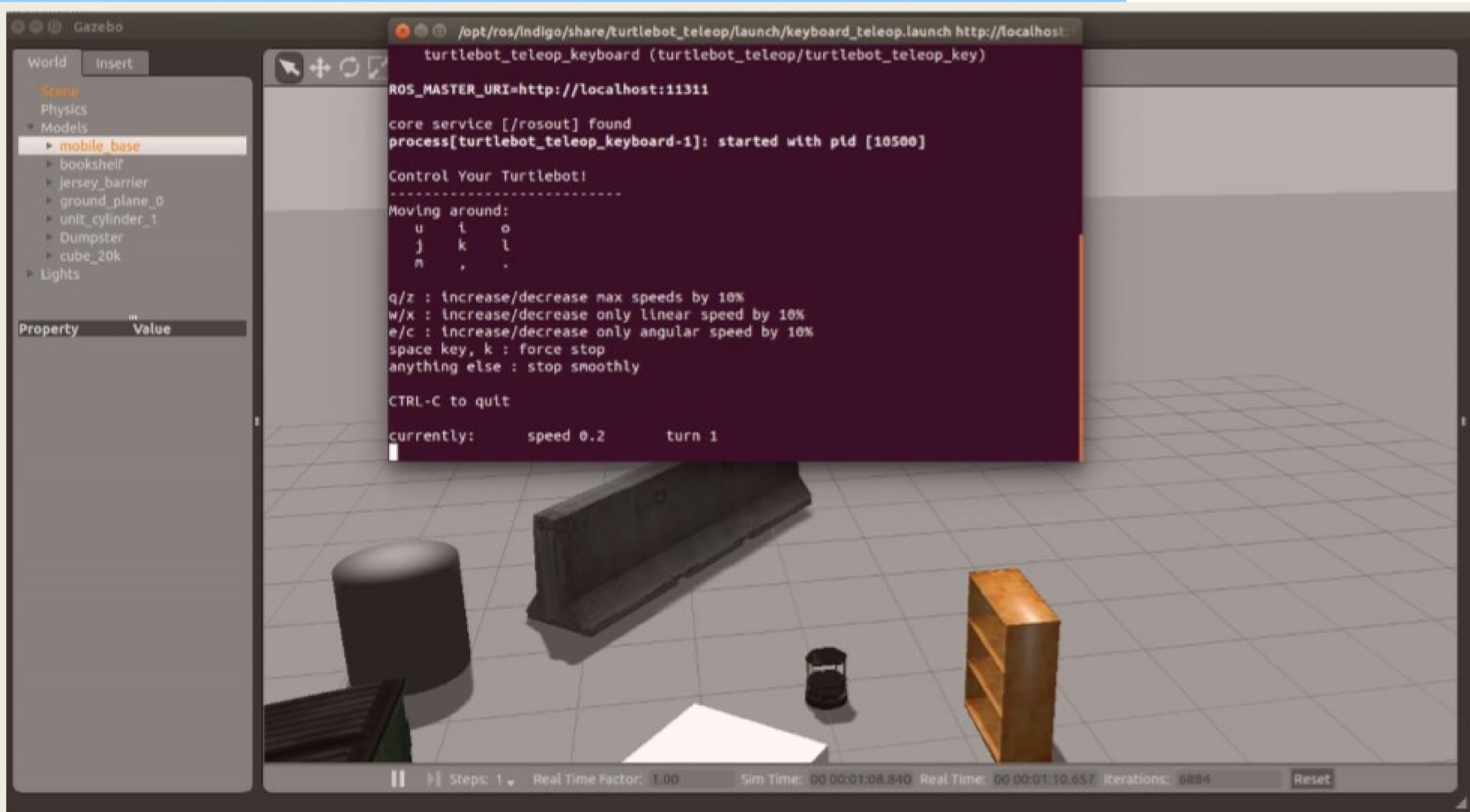


Move the Turtlebot with Teleop

- Let’s launch the teleop package so we can move it around the environment
- Run the following command:

```
$ roslaunch turtlebot_teleop keyboard_teleop.launch
```

Moving Turtlebot with Teleop



Moving Turtlebot from Code

- We will now add a node that will make turtlebot random walk in the environment
  - Gazebo is publishing the some topics
- Create a new package turtlebot_random_walk

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg turtlebot_random_walk std_msgs rospy roscpp
```

- Add the following cpp file to the src directory of the package

random_walk.cpp (1)

```
#include <ros/ros.h>
#include <tf/transform_listener.h>
#include <iostream>
#include <vector>
#include "geometry_msgs/Twist.h"
#include "sensor_msgs/LaserScan.h"

using namespace std;

#define LINEAR_SPEED 0.2
#define ANGULAR_SPEED 0.2
#define MIN_DIST_FROM_OBSTACLE 0.8

void readSensorCallback(const sensor_msgs::LaserScan::ConstPtr &sensor_msg);

bool obstacleFound = false;
```

random_walk.cpp (2)

```
int main(int argc, char **argv) {
    ros::init(argc, argv, "random_walk_node");
    ros::NodeHandle nh;

    ros::Publisher cmd_vel_pub = nh.advertise<geometry_msgs::Twist>("/cmd_vel_mux/input/teleop", 10);
    ros::Subscriber base_scan_sub = nh.subscribe<sensor_msgs::LaserScan>(
            "scan", 1, &readSensorCallback);

    geometry_msgs::Twist moveForwardCommand;
    moveForwardCommand.linear.x = LINEAR_SPEED;

    geometry_msgs::Twist turnCommand;
    turnCommand.angular.z = ANGULAR_SPEED;

    ros::Rate loop_rate(10);

    while (ros::ok()) {
        if (obstacleFound) {
            cmd_vel_pub.publish(turnCommand);
            ROS_INFO("Turning around");
        } else {
            cmd_vel_pub.publish(moveForwardCommand);
            ROS_INFO("Moving forward");
        }

        ros::spinOnce(); // let ROS process incoming messages
        loop_rate.sleep();
    }
    return 0;
}
```

random_walk.cpp (3)

```
void readSensorCallback(const sensor_msgs::LaserScan::ConstPtr &scan) {
    bool isObstacle = false;

    for (int i = 0; i < scan->ranges.size(); i++) {
        if (scan->ranges[i] < MIN_DIST_FROM_OBSTACLE) {
            isObstacle = true;
            break;
        }
    }

    if (isObstacle) {
        ROS_INFO("Obstacle found in front!");
        obstacleFound = true;
    } else {
        obstacleFound = false;
    }
}
```

Moving Turtlebot from Code

- Create a launch subdirectory within the package and add the launch file random_walk.launch to it

```
<launch>
  <param name="/use_sim_time" value="true" />

  <!-- Launch turtle bot world -->
  <include file="$(find turtlebot_gazebo)/launch/turtlebot_world.launch"/>

  <!-- Launch random walk node -->
  <node name="turtlebot_random_walk_node" pkg="turtlebot_random_walk"
type="turtlebot_random_walk_node" output="screen"/>
</launch>
```

Launch Random Walk Node

- To launch the random walk node type:

```
$ roslaunch turtlebot_random_walk random_walk.launch
```