Software Engineering & Simulation of Robotic Systems

PDE4420

Gazebo and ROS

Zhijun Yang
Middlesex University
The Burroughs, London NW4 4BT, UK
Email: Z.Yang@mdx.ac.uk
Internal Telephone: 12845
Room: TG21

Agenda

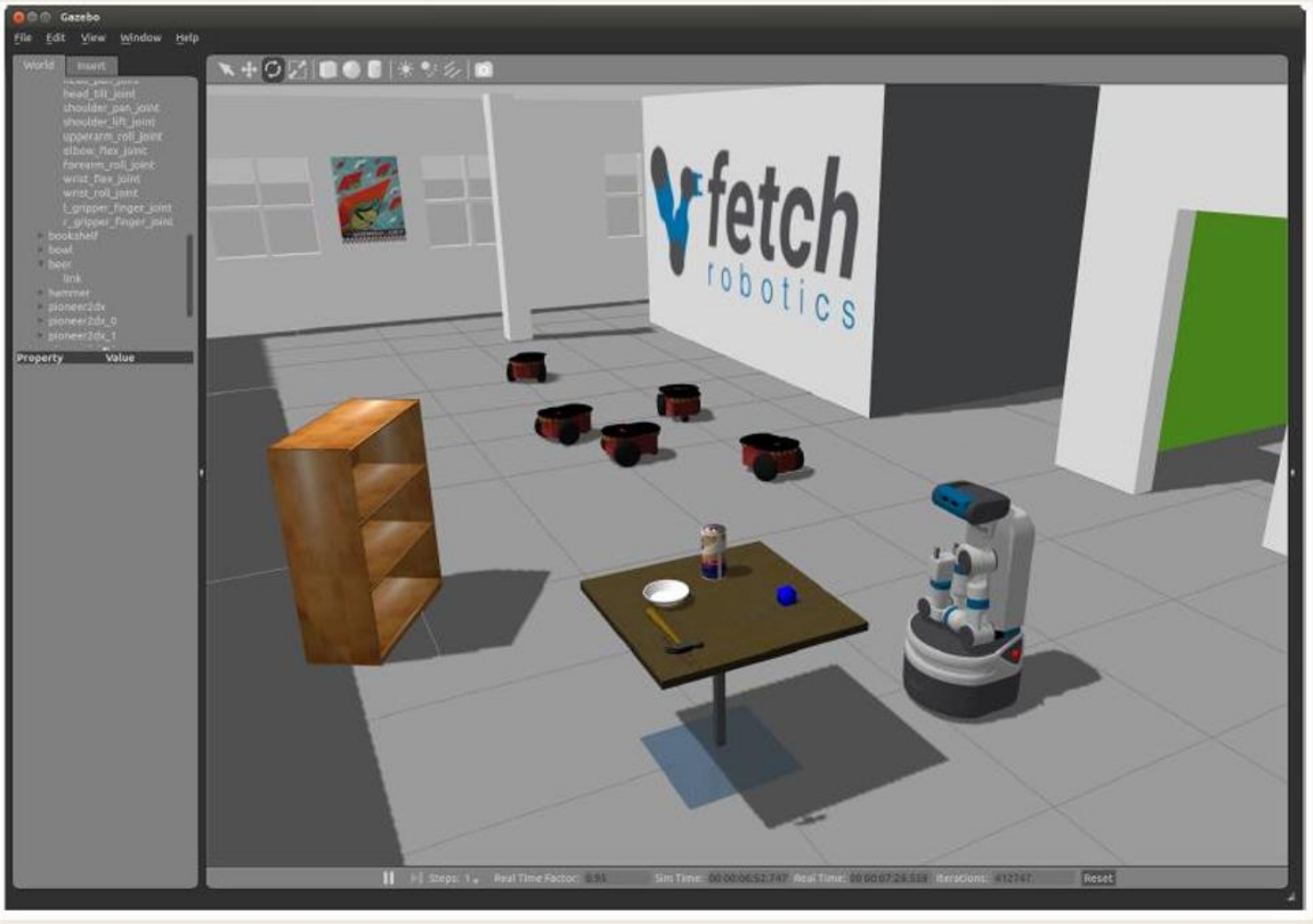
- Gazebo 3D simulator
- Model SDF files
- Gazebo and ROS integration
- TurtleBot simulation

Gazebo

Middlesex University London

- 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
File Edit View Window Help
World Insert
Property Value
fetch
robotics
Reset

Gazebo Installation

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

Gazebo Architecture

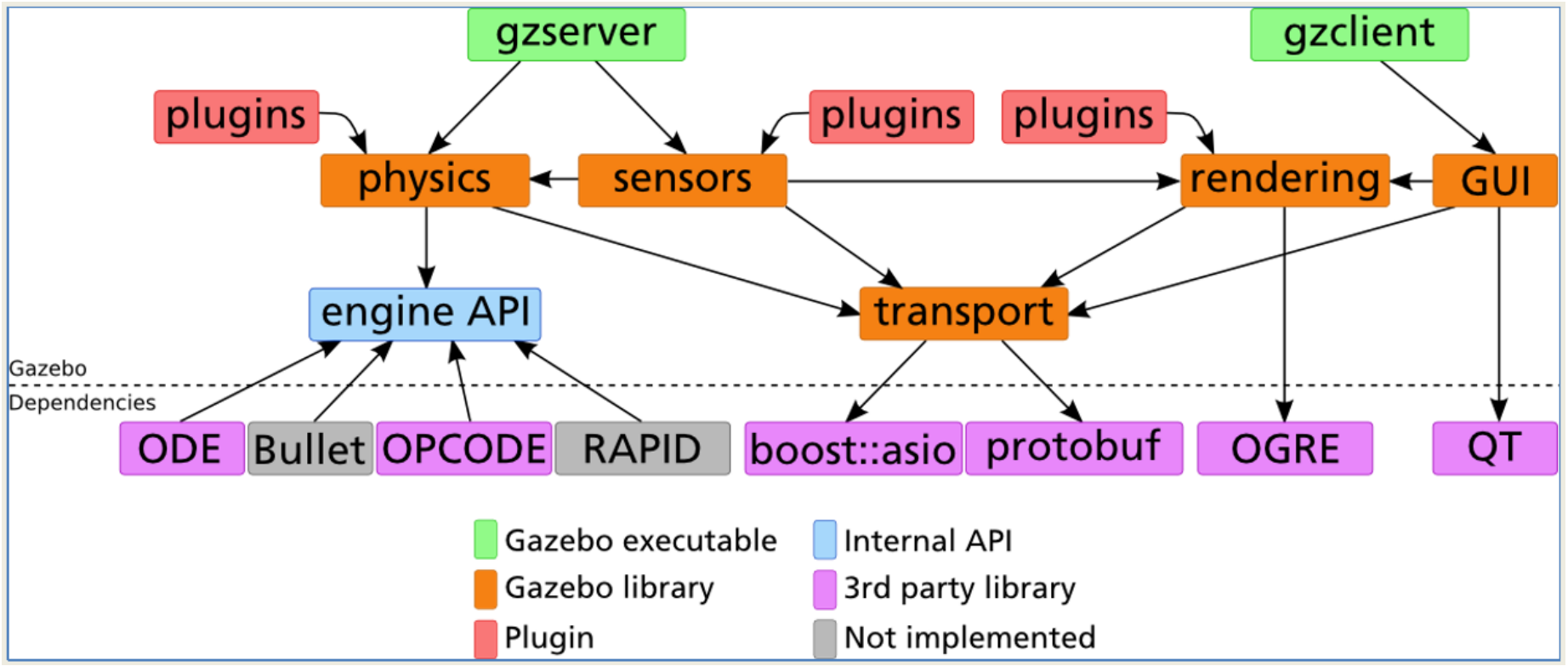
Middlesex University London

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

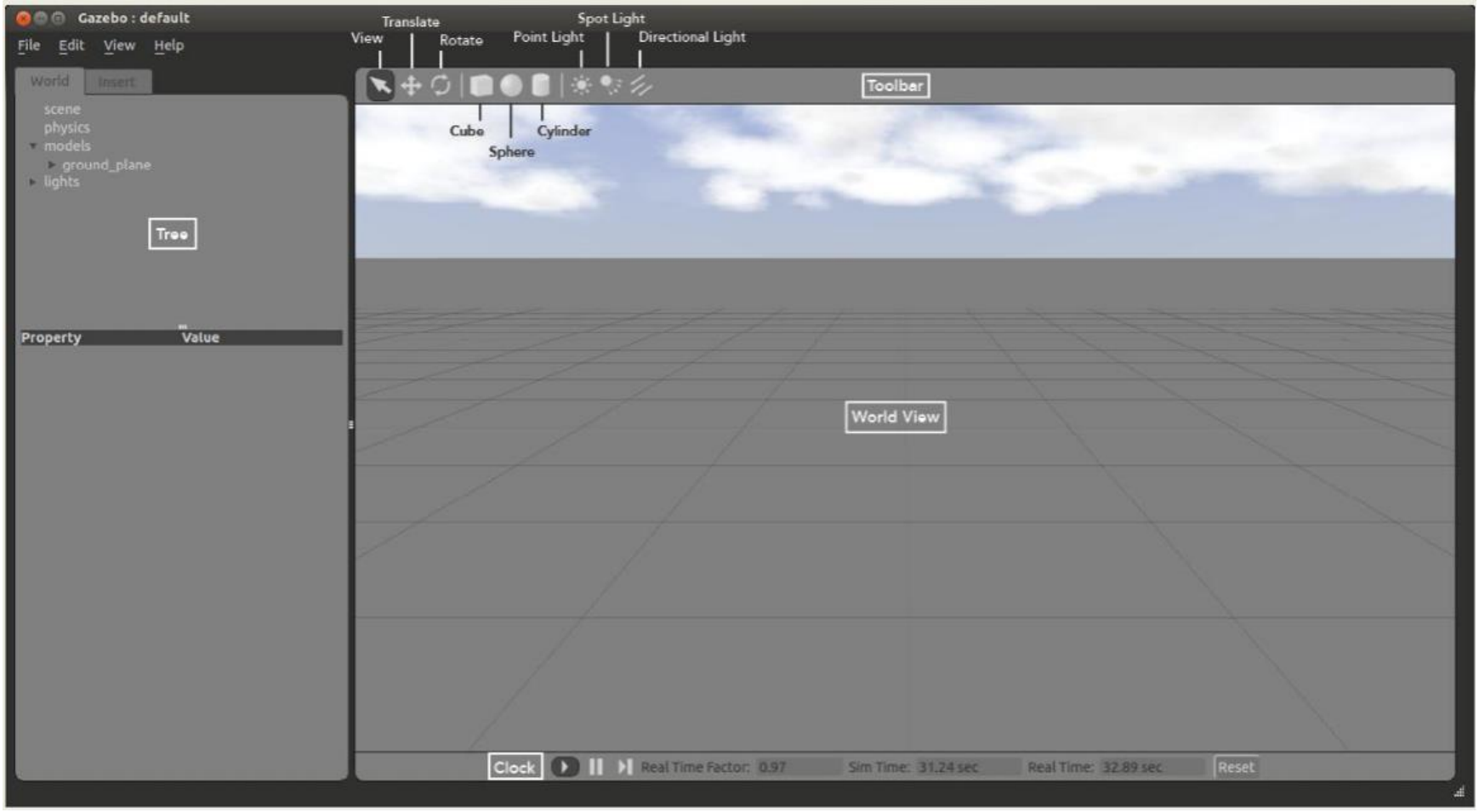
Middlesex University London

- 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

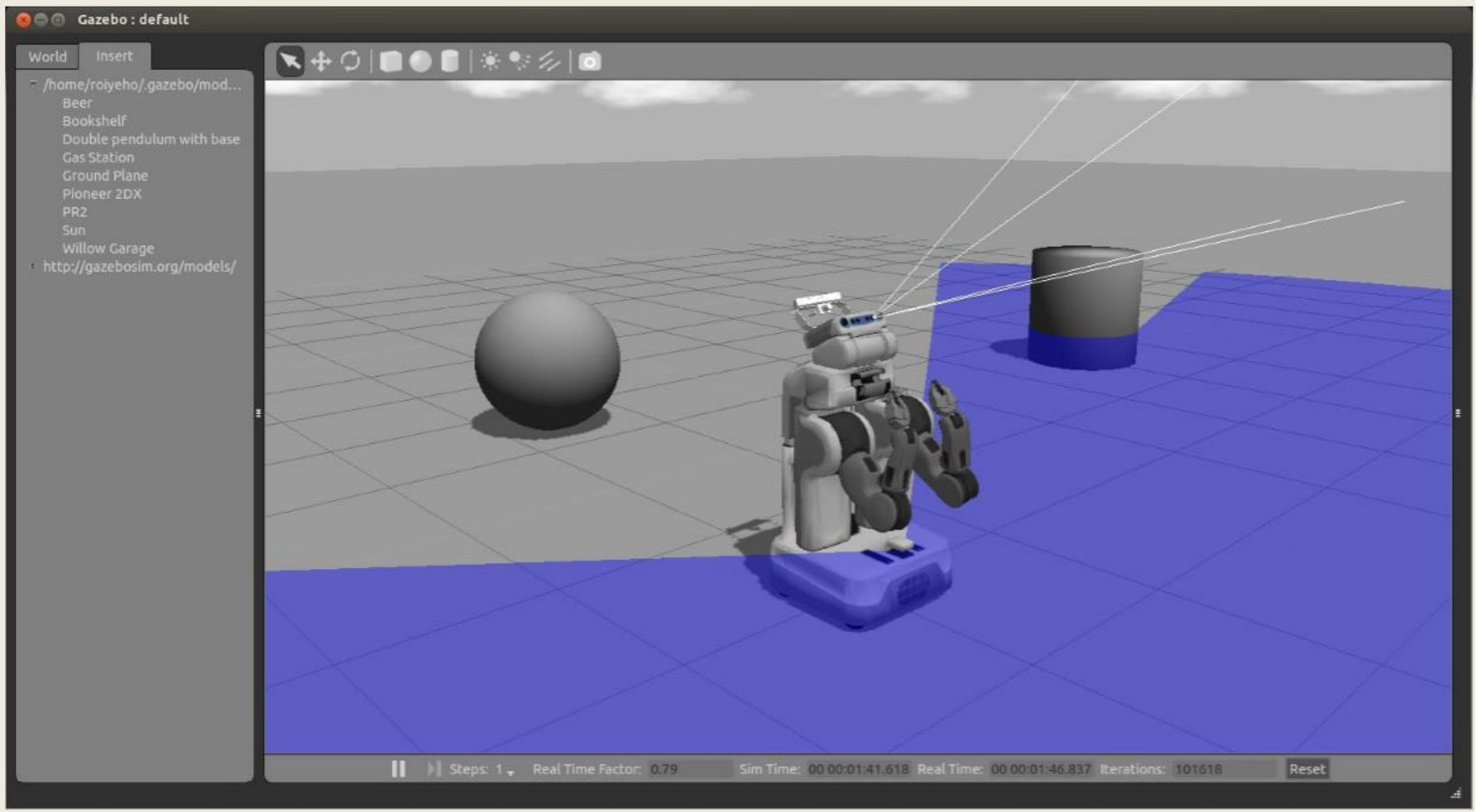
(Orbit & Zoom work in this mode, as well)

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

Middlesex University London

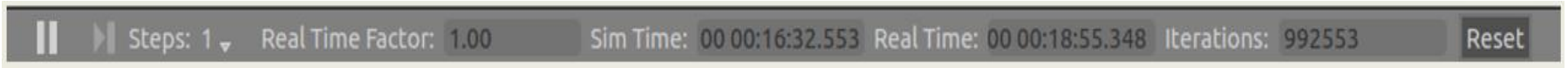
- The models item in the world tab contains a list of all models and their links

- 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

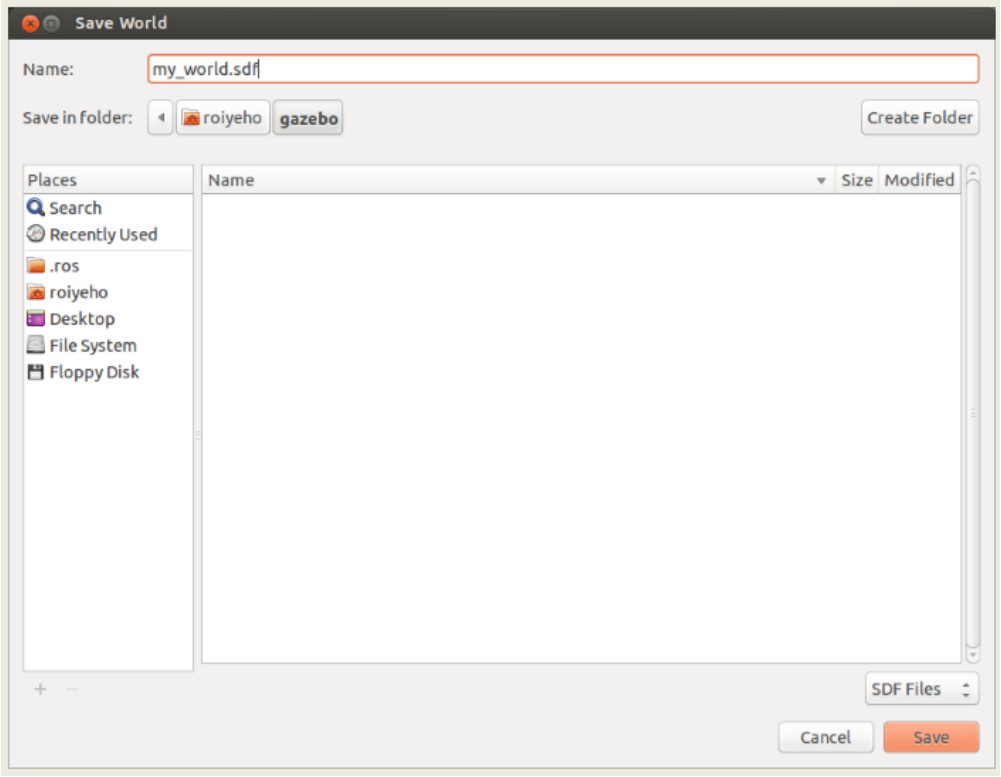
- You can start, pause and step through the simulation with the clock
- It is located at the bottom of the World View



- **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

Middlesex University London

- 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)

Middlesex University London

• **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

Middlesex University London

• **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

Middlesex University London

- 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

Middlesex University London

• 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 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
```

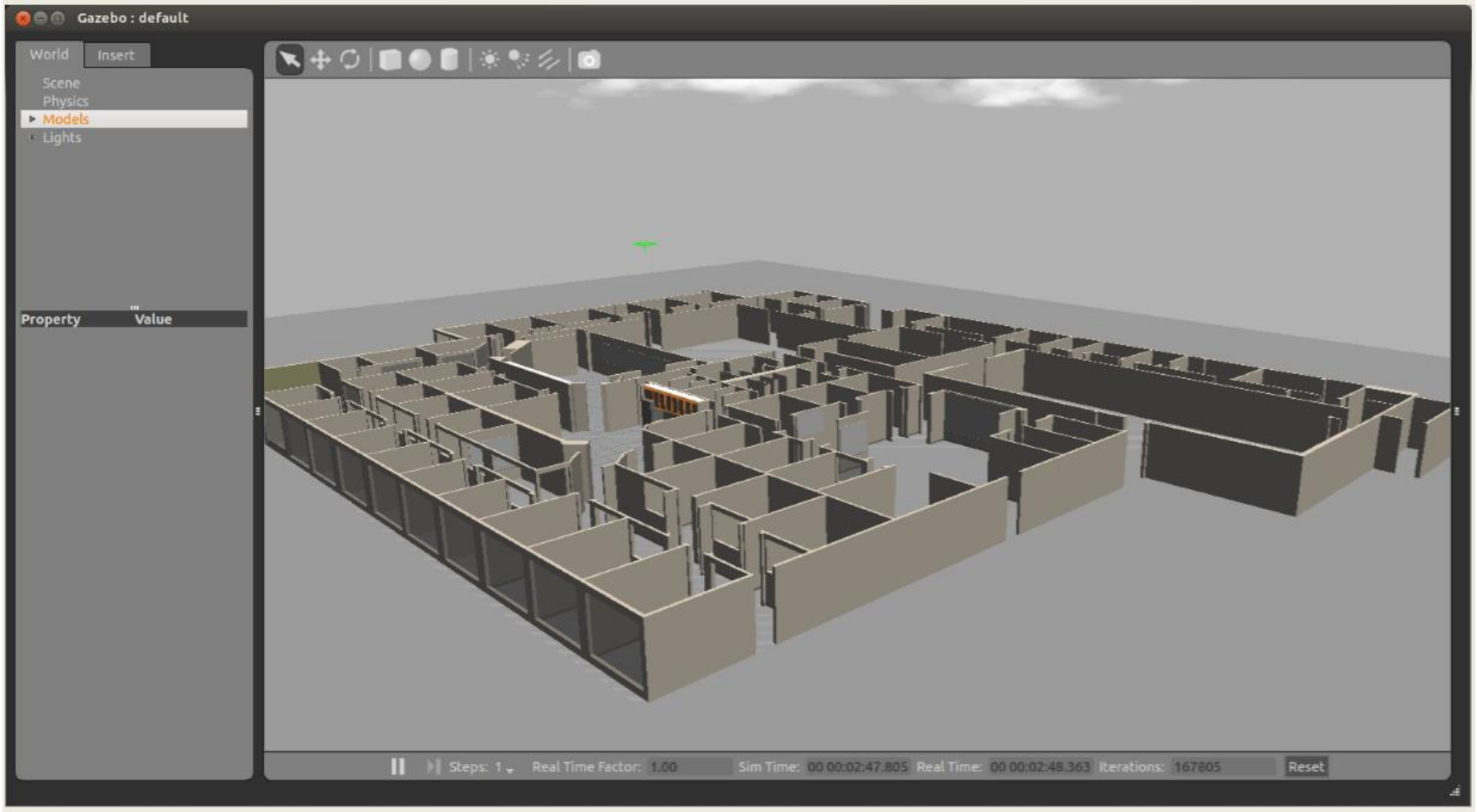
willowgarage_world.launch

```
<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

Middlesex University London



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

Middlesex University London

```
<?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

Middlesex University London

- 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

Middlesex University London

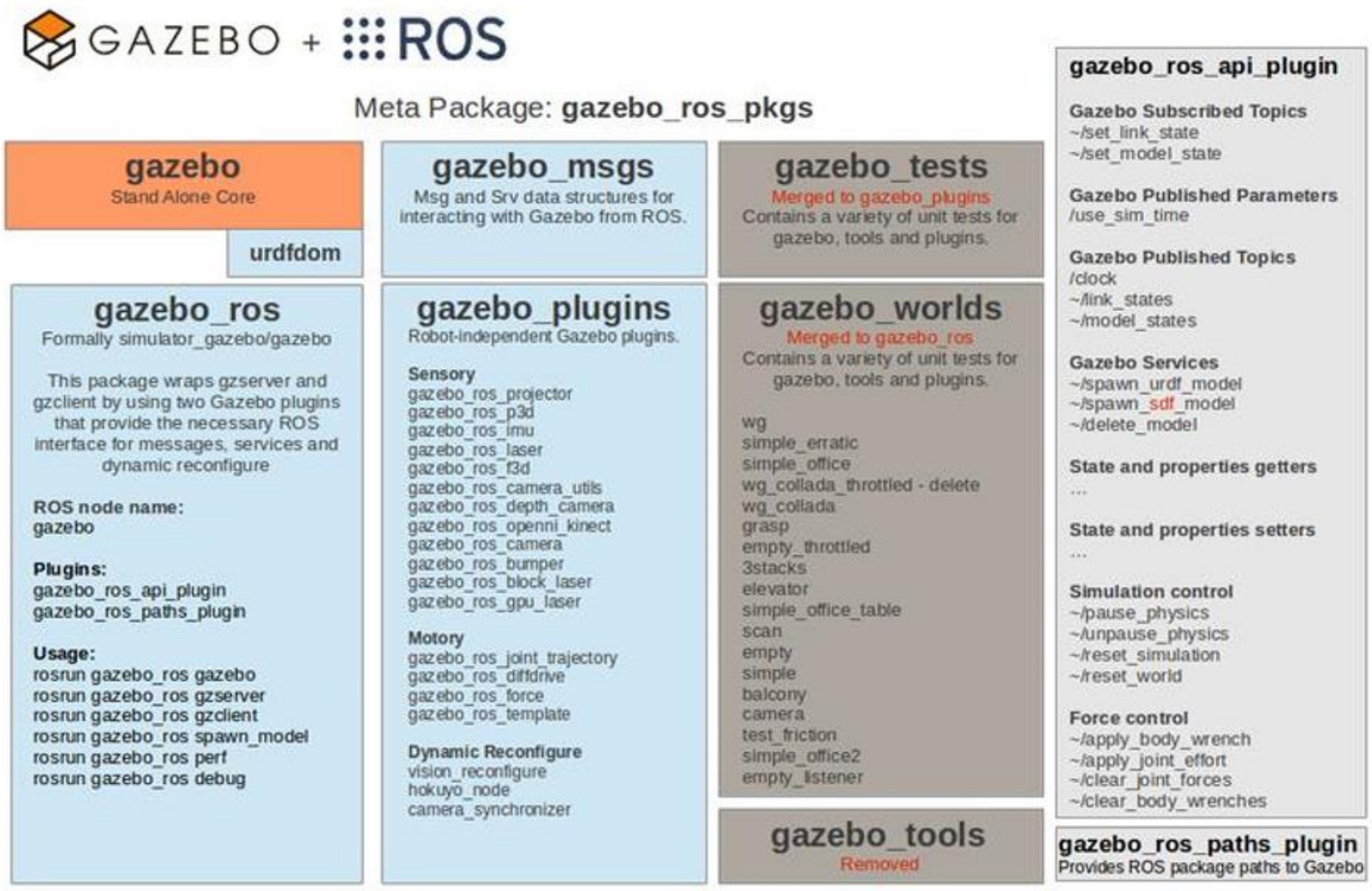
- ROS integrates closely with Gazebo through the **gazebo_ros** package

- 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



Gazebo ROS Services

Middlesex University London

```
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

Middlesex University London

- 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

Middlesex University London

- 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

Middlesex University London

- To install Turtlebot simulation stack type:

```
$ sudo apt install ros-melodic-turtlebot3-gazebo ros-melodic-turtlebot3-applications*
```

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

```
$ export TURTLEBOT3_MODEL=burger
```

```
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

turtlebot3_world.launch

```
<launch>
  <!-- <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/> -->
  <arg name="model" default="waffle"/>
  <arg name="x_pos" default="-2.0"/>
  <arg name="y_pos" default="-0.5"/>
  <arg name="z_pos" default="0.0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/turtlebot3_world.world"/>
    <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>

  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot3_description)/urdf/
turtlebot3_$(arg model).urdf.xacro" />

  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf"  args="-urdf -model turtlebot3_$(arg model) -x $
(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -param robot_description" />
</launch>
```

Spawning URDF Robots

Middlesex University London

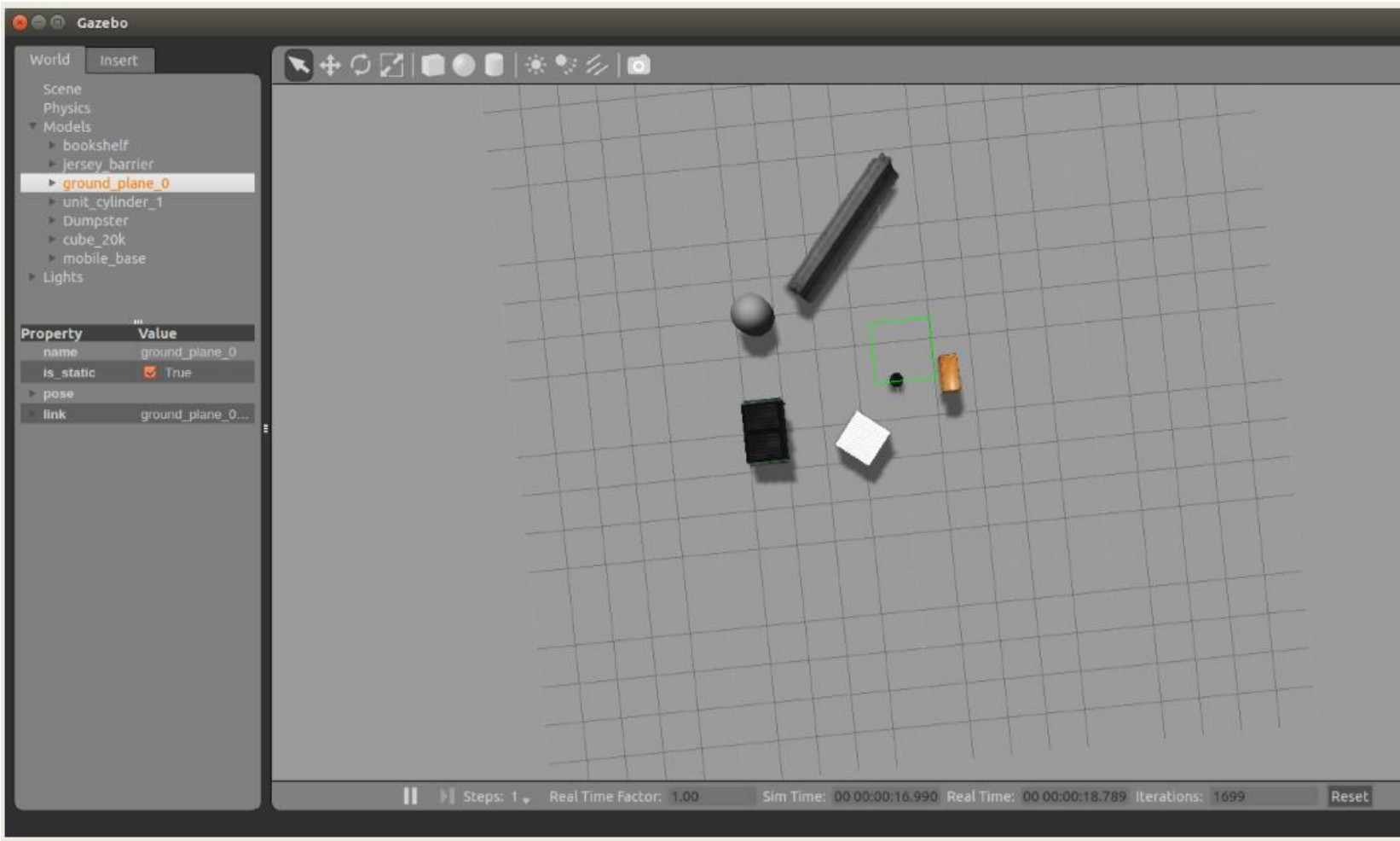
- The spawn_model node in gazebo_ros package makes a service call request to the gazebo ROS node in order to add a custom URDF into Gazebo
- You can use this script in the following way:

```
$ rosrun gazebo_ros gazebo
$ rosrun gazebo_ros spawn_model -file /opt/ros/melodic/share/turtlebot3_description/urdf/turtlebot3_burger.urdf.xacro -urdf -x 2.4 -y 1.2 -z 0 -model turtlebot3_burger
```

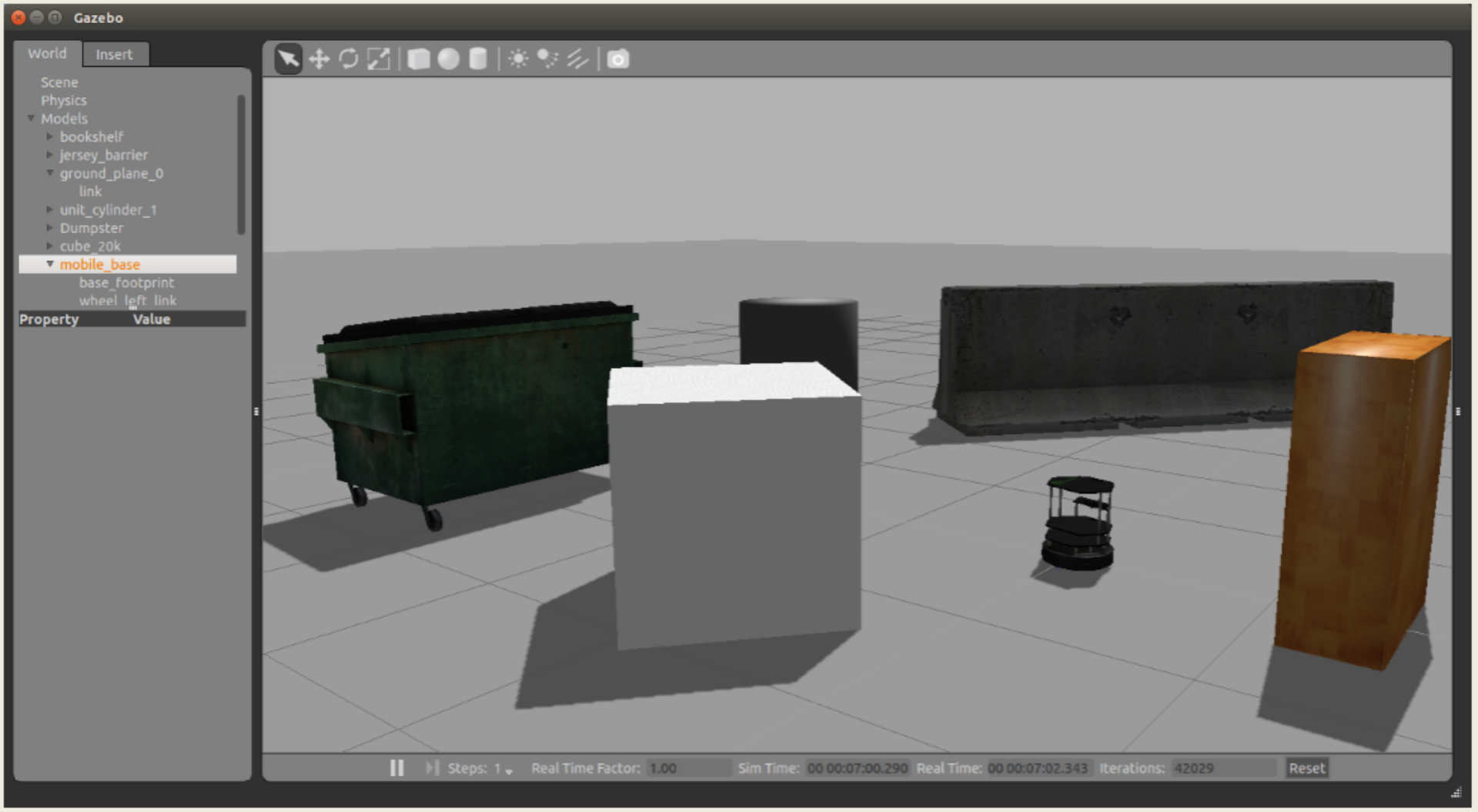
- The x,y,z arguments indicate the initial location of the robot

Turtlebot Simulation

Middlesex University London



Turtlebot Simulation

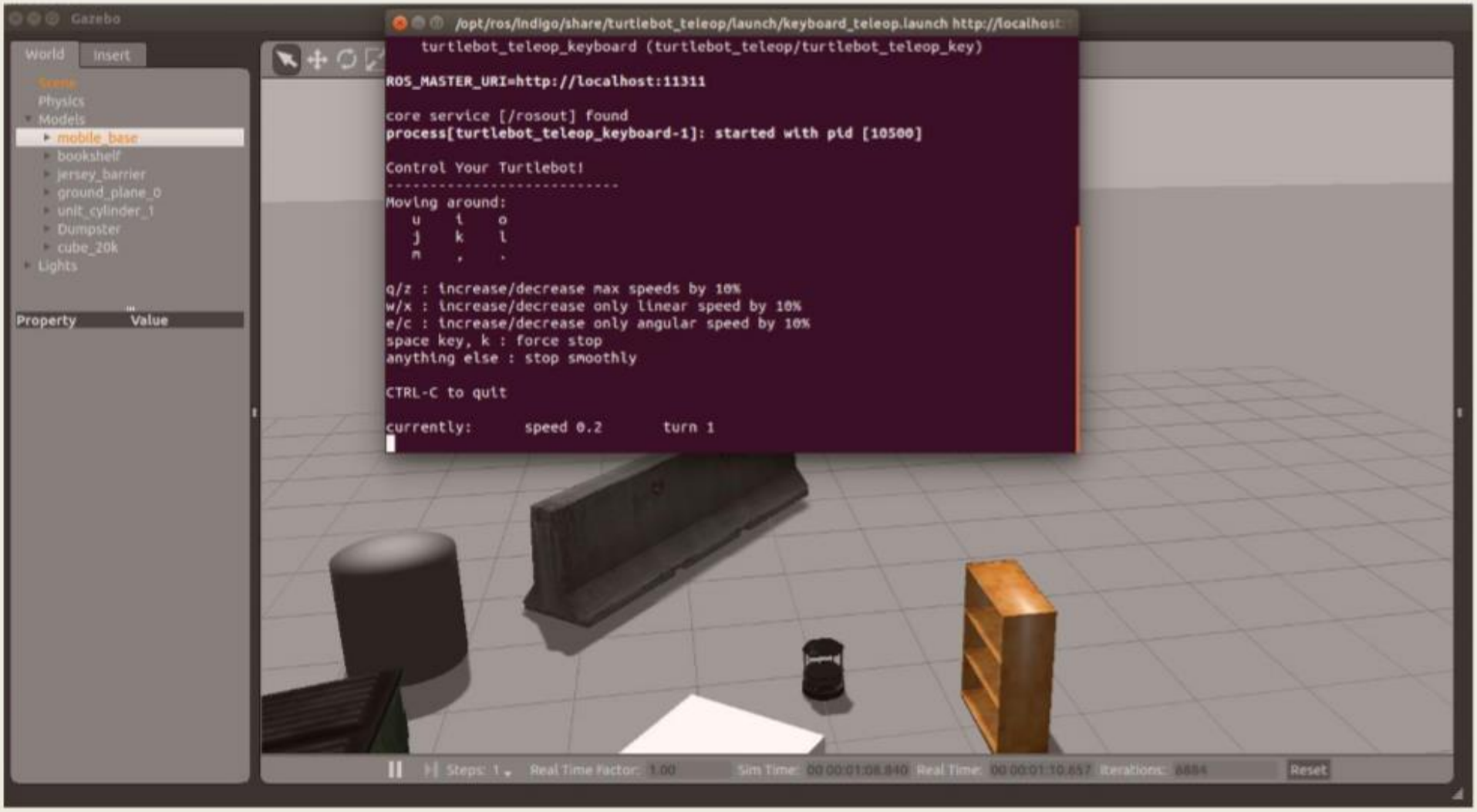


Move the Turtlebot with Teleop

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

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_gazebo turtlebot3_stage_4.launch
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Moving Turtlebot with Teleop



Moving Turtlebot for a Map

Middlesex University London

- We will now try to build a map of the environment. Run:

$ roslaunch turtlebot3_gazebo turtlebot3_world.launch

$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping

Note: if you find ERROR: cannot launch node of type [gmapping/slam_gmapping] , then run:
$ sudo apt install ros-melodic-slam-gmapping

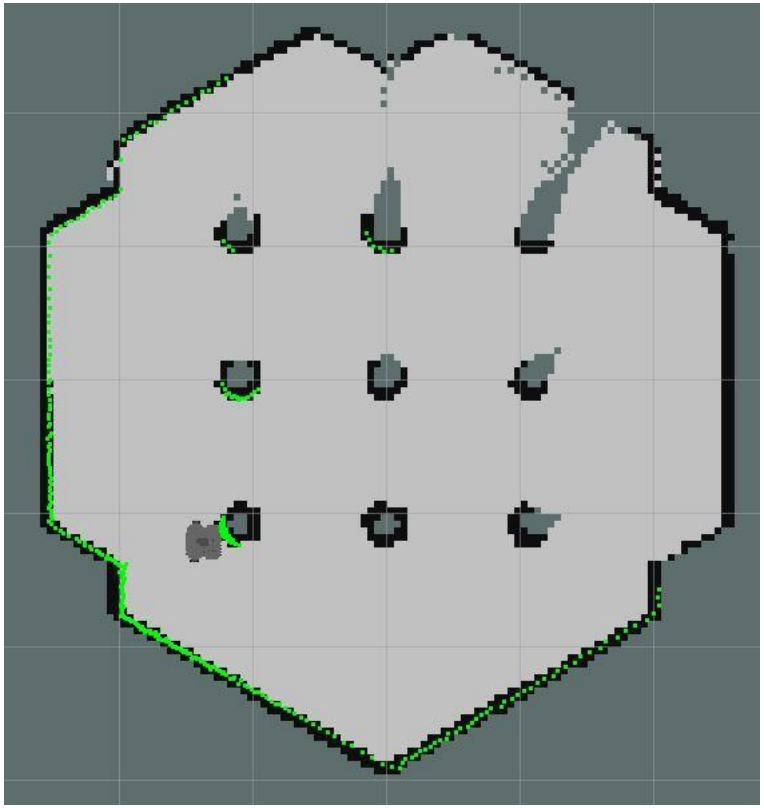
$ roslaunch turtlebot3_gazebo turtlebot3_simulation.launch

Note: this launch file start turtlebot3_drive node, but the /cmd_vel topic may carry 0 in speed, run:

$ rostopic pub -1 /cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'

You will see the map is building with the robot walking. You can look around by using, e.g., rqt_graph, rostopic list, etc.

Moving Turtlebot for a Map



You can choose to save the map by running:
$ rosrun map_server map_saver -f ~/map