
Developer's Master

Gazebo Simulator

Unit 4: Build a World

- Summary -

Estimated time to completion: **4 hours**

This unit presents concepts around creating a new world in Gazebo. You will learn to create one from scratch and put different models, ground, animated objects, and actors. In addition, you will learn how to use plugins and how to control worlds programmatically.

- End of Summary -

To follow this unit, create a new package in the `~/catkin_ws/src` folder.

► Execute

```
In [ ]: cd ~/catkin_ws/src
```



```
In [ ]: catkin_create_pkg studying_gazebo rospy urdf
```



4.1 Building a World

You will explore two different ways of creating a Gazebo world. Each one has its pros and cons.

- Building a world using the GUI -

The easiest way to build a world is by using the GUI. First, open Gazebo by using its CLI command:

► Execute

In []: gazebo



It must start a new Gazebo simulation with an empty world. Next, add the existing models, positioning, rotating, and scaling as you wish, as was taught in the first unit of this course. Check the example below, created using the GUI:



Then you want to use this world in your ROS applications. Save it in a file
~/catkin_ws/src/studying_gazebo/worlds/example-gui.world

Open that file in the Code Editor. You must have something with, more or less, 200 lines of code. This file saves everything about the current state of the simulation, including the camera position, which means the perspective you have at this moment of the 3D world.

Look for **model** tags inside this file. It is an **XML syntax**. You must find some of these tags, for example:

```
<model name='Construction Barrel'>
...
  <pose>-5.31619 -5.78841 0 0 -0 0</pose>
</model>
...
<model name='ambulance'>
...
  <pose>-6.63603 -1.32489 0 0 -0 0</pose>
</model>
...
```

Everything about the ambulance model is written there, including visuals, collision, surface friction, and much more. This file is self-contained. It does not need anything else to work in Gazebo.

- End of Building a world using the GUI -

- Building a world including models -

Now, open in one of the terminals the file `/usr/share/gazebo-11/worlds/empty.world`.

► Execute

```
In [ ]: vim /usr/share/gazebo-11/worlds/empty.world
```



Check that you do not have the whole definition for the models **sun** and **ground_plane**. This is because they are included. This is a way to define worlds in a lightweight file, using the models your installed Gazebo already has.

Create a new file `~/catkin_ws/src/studying_gazebo/worlds/example2.world` with the following content:

```
In [ ]: <?xml version="1.0" ?>
<sdf version="1.5">
  <world name="default">
    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
    </include>

    <!-- A ground plane -->
    <include>
      <uri>model://ground_plane</uri>
    </include>

    <!-- ambulance -->
    <include>
      <pose>-6.63603 -1.32489 0 0 -0 0</pose>
      <uri>model://ambulance</uri>
    </include>

    <!-- construction barrel -->
    <include>
      <pose>-5.31619 -5.78841 0 0 -0 0</pose>
      <uri>model://construction_barrel</uri>
    </include>
  </world>
</sdf>
```



Launch Gazebo and set up this world to be opened. You will get a similar result in the simulation. In terms of models and their positions, they will be the same. Camera positioning and any other changes using the GUI will not be included.

This is the way to create worlds by editing the files. It can be used to generate worlds programmatically, for example.

- End of Building a world including models -

4.2 Gazebo Models

- Structure and requirements -

Gazebo can load models in the simulation using the **world** file launched or the **GUI**. In addition, Gazebo has a model database repository, which is available on GitHub: https://github.com/osrf/gazebo_models (https://github.com/osrf/gazebo_models). These models are also downloaded/installed on the computer after Gazebo installation.

Launch an empty world to check the models available in the Gazebo environment. Create a new launch file in the package:

► Execute

```
In [ ]: mkdir ~/catkin_ws/src/studying_gazebo/launch/
```

```
In [ ]: mkdir ~/catkin_ws/src/studying_gazebo/worlds/
```

```
In [ ]: touch ~/catkin_ws/src/studying_gazebo/launch/empty_world.launch
```

```
In [ ]: vim ~/catkin_ws/src/studying_gazebo/launch/empty_world.launch
```

Then paste the content below to this new launch file:

► Copy and paste

In []:  `<launch>`
 `<arg name="model" default="burger" doc="model type [burger, waffle, waffle_p`
 `<arg name="x_pos" default="0.8"/>`
 `<arg name="y_pos" default="0.0"/>`
 `<arg name="z_pos" default="0.0"/>`

 `<include file="$(find gazebo_ros)/launch/empty_world.launch">`
 `<arg name="world_name" value="$(find gazebo_ros)/launch/empty_world.launch`
 `<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>`

In []:  Compile the workspace **and** launch the empty world:

► Execute

In []:  `cd ~/catkin_ws`

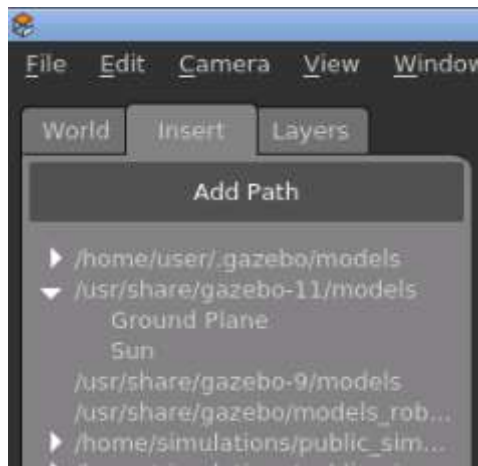
In []:  `catkin_make`

In []:  `source devel/setup.bash`

In []:  `roslaunch studying_gazebo empty_world.launch`

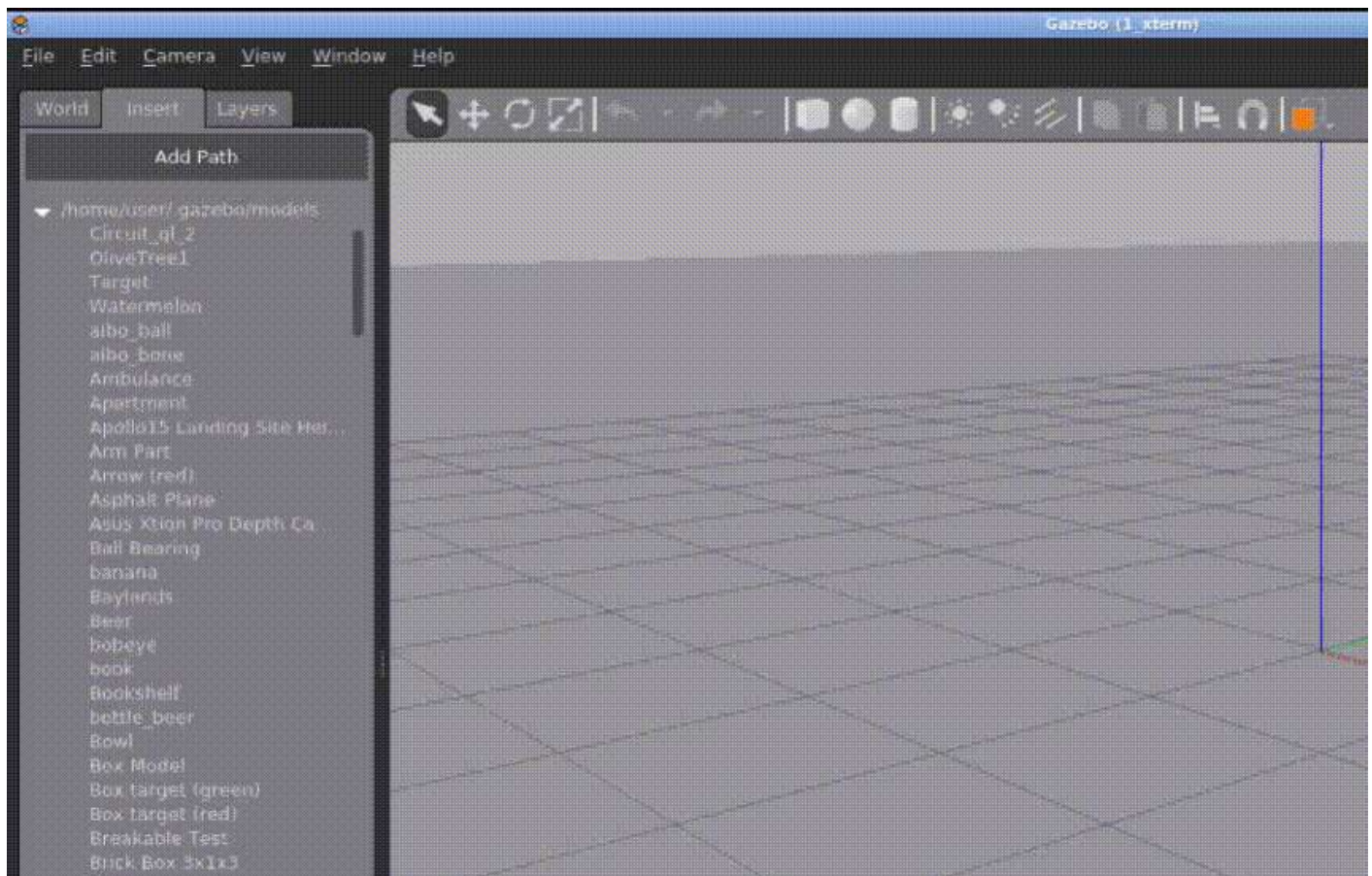
You should have Gazebo open. Check the left panel and click on the tab **Insert**. There are at least two valid paths:

- `/home/user/.gazebo/models`
- `/usr/share/gazebo-11/models`



In the `/usr/share/..` folder, you have only two models, ground and sun. However, in the first folder, in the user's home, there are many models. These are installed on this computer. Gazebo detects ROS workspaces with models and other folders in the `$GAZEBO_MODEL_PATH` variable.

Select one of the models by left-clicking and placing it in the simulation.



A model must have a defined structure to be available. Go to the folder for one of the models, the ambulance, for example:

► Execute

```
In [ ]: cd /home/user/.gazebo/models/ambulance/
```

```
user:~/ .gazebo/models/ambulance$ ls -l
total 16
drwxr-xr-x 1 user user 4096 Dec  2  2020 materials
drwxr-xr-x 1 user user 4096 Dec  2  2020 meshes
-rw-r--r-- 1 user user  381 Dec  7  2020 model.config
-rw-r--r-- 1 user user  616 Dec  7  2020 model.sdf
```

It has two mandatory files:

- model.config
- model.sdf

SDF files describe the model, and **config** contains the meta-data used by Gazebo to display it in the menu.

The folders, **materials**, and **meshes** contain resources to render the model, images, and 3D files (i.e., DAE, MTL, OBJ, etc.)

See how they are used in the next section!

- End of Structure and requirements -

4.3 Creating Gazebo Models

- How to create a new model -

The first step is to have the basic folder and file structure for a new model. So use the new package and create a folder to contain your models:

► Execute

```
In [ ]: cd ~/catkin_ws/src/studying_gazebo
```

```
In [ ]: mkdir -p models/box_model
```

```
In [ ]: cd models/box_model
```

```
In [ ]: touch model.config
```

In []: touch model.sdf



Do not use materials or textures in this first moment. Instead, copy and paste the content below to the **model.config** file. Change author and other information accordingly:

► Copy and paste

```
In [ ]: <?xml version="1.0"?>

<model>
  <name>Box Model</name>
  <version>1.0</version>
  <sdf version="1.6">model.sdf</sdf>

  <author>
    <name>Author's name</name>
    <email>no-author@mail.com</email>
  </author>

  <description>
    A simple box model
  </description>
</model>
```



And the following to the **model.sdf** file:

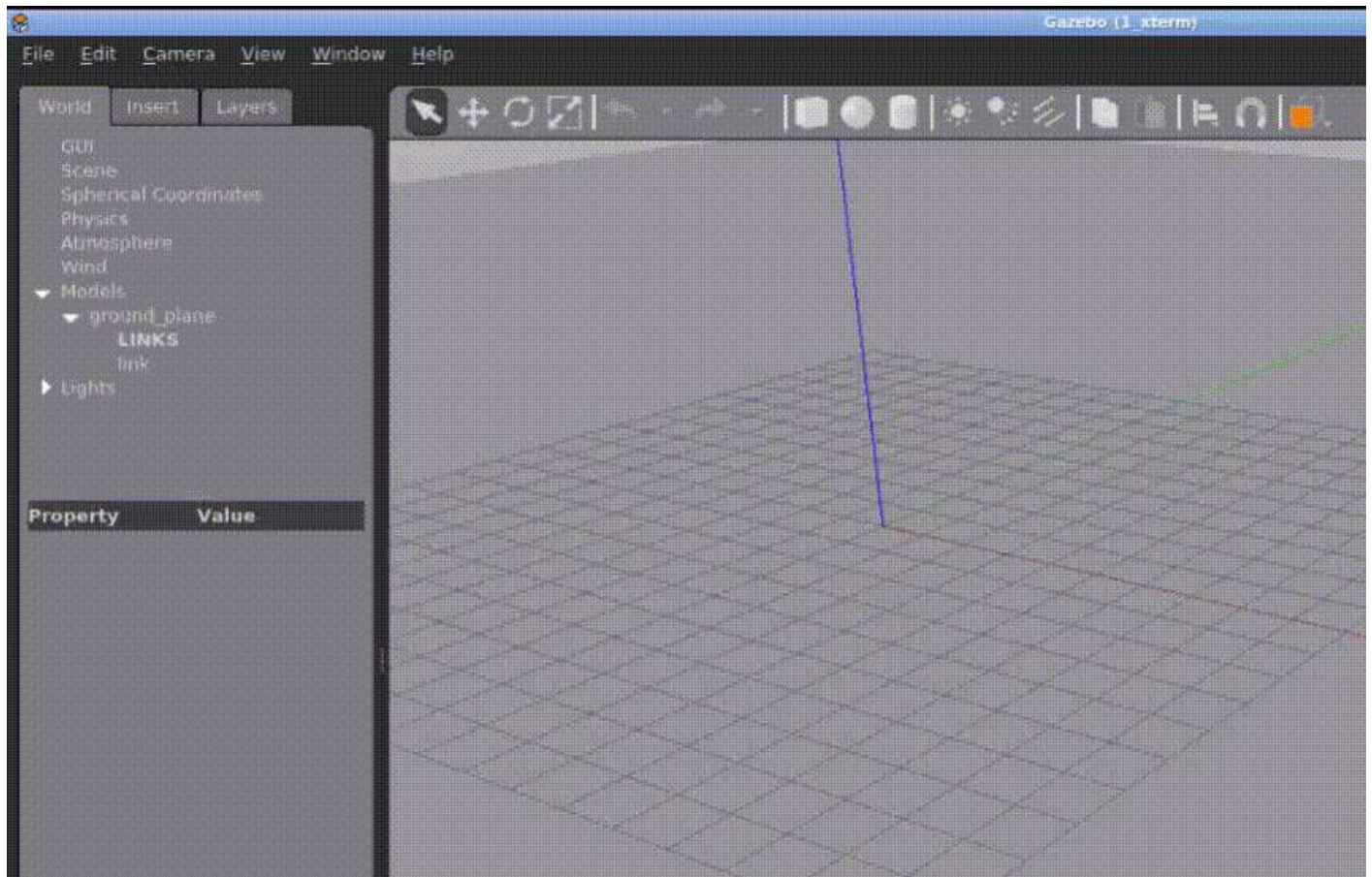
► Copy and paste

In []:

```
<?xml version='1.0'?>
<sdf version="1.4">
  <model name="box_model">
    <pose>0 0 0.5 0 0 0</pose>
    <static>false</static>
    <link name="link">
      <inertial>
        <mass>1.0</mass>
        <inertia>
          <ixx>0.083</ixx>
          <ixy>0.0</ixy>
          <ixz>0.0</ixz>
          <iyy>0.083</iyy>
          <iyz>0.0</iyz>
          <izz>0.083</izz>
        </inertia>
      </inertial>
      <collision name="collision">
        <geometry>
          <box>
            <size>1 1 1</size>
          </box>
        </geometry>
      </collision>
      <visual name="visual">
        <geometry>
          <box>
            <size>1 1 1</size>
          </box>
        </geometry>
      </visual>
    </link>
  </model>
</sdf>
```



Now you can use this model in Gazebo simulation. Relaunch the empty world. Go to **Insert** tab, add the new **models** folder to the path and insert the new Box Model created.



SDF file

Models SDF file are XML and must be defined at the beginning:

```
<?xml version='1.0'?>
<sdf version="1.4">
  <model name="box_model">
    </model>
    ...
</sdf>
```

Below the **model** tag, there are three attributes:

- pose
- static
- link

Pose is where the object will be placed in the world. Since the model is defined in a different file, this pose will not be used to spawn the model in the world. You choose this. Although this is a reference for the model definition, it is where the object dimensions will start! The arguments are **X, Y, Z, Roll, Pitch, and Yaw**.

```
<pose>X Y Z R P Y</pose>
```

Static is a boolean. If it equals "true," the object will not move or respond to any forces applied.

```
<static>true/false</static>
```

Link is the only part of the model you are defining. A model can have many links combined with **joints** (you will see it later). In that case, a single link defines it.

```
<link name="link_name">
  <inertial>...</inertial>
  <collision></collision>
  <visual></visual>
</link>
```

The **link** attribute itself has many other attributes:

- Inertial
- Collision
- Visual

Inertial is about the mass and inertia matrix. The physics engine uses them to calculate the object response during the simulation.

Collision is where you define the object's dimensions to be taken into account when the physics engine calculates collisions with other objects. You do not see the collision shape by default.

Visual is where you define the shape of what you will see in the simulation. In that simple case, collision and visual are the same shape and size. However, you may want to have different shapes when you have very complex models, for example, and do not want to waste computational power to calculate collisions.

It used simple **geometry** to define the shape of the object. There are many other options, like a **cylinder** or **spheres**. Check the official docs here: <http://sdformat.org/spec?elem=geometry> (<http://sdformat.org/spec?elem=geometry>).

- End of How to create a new model -

- Exercise 4.3 -

In this exercise, create a new model called **custom_shape**. This model will have two different visual geometries.

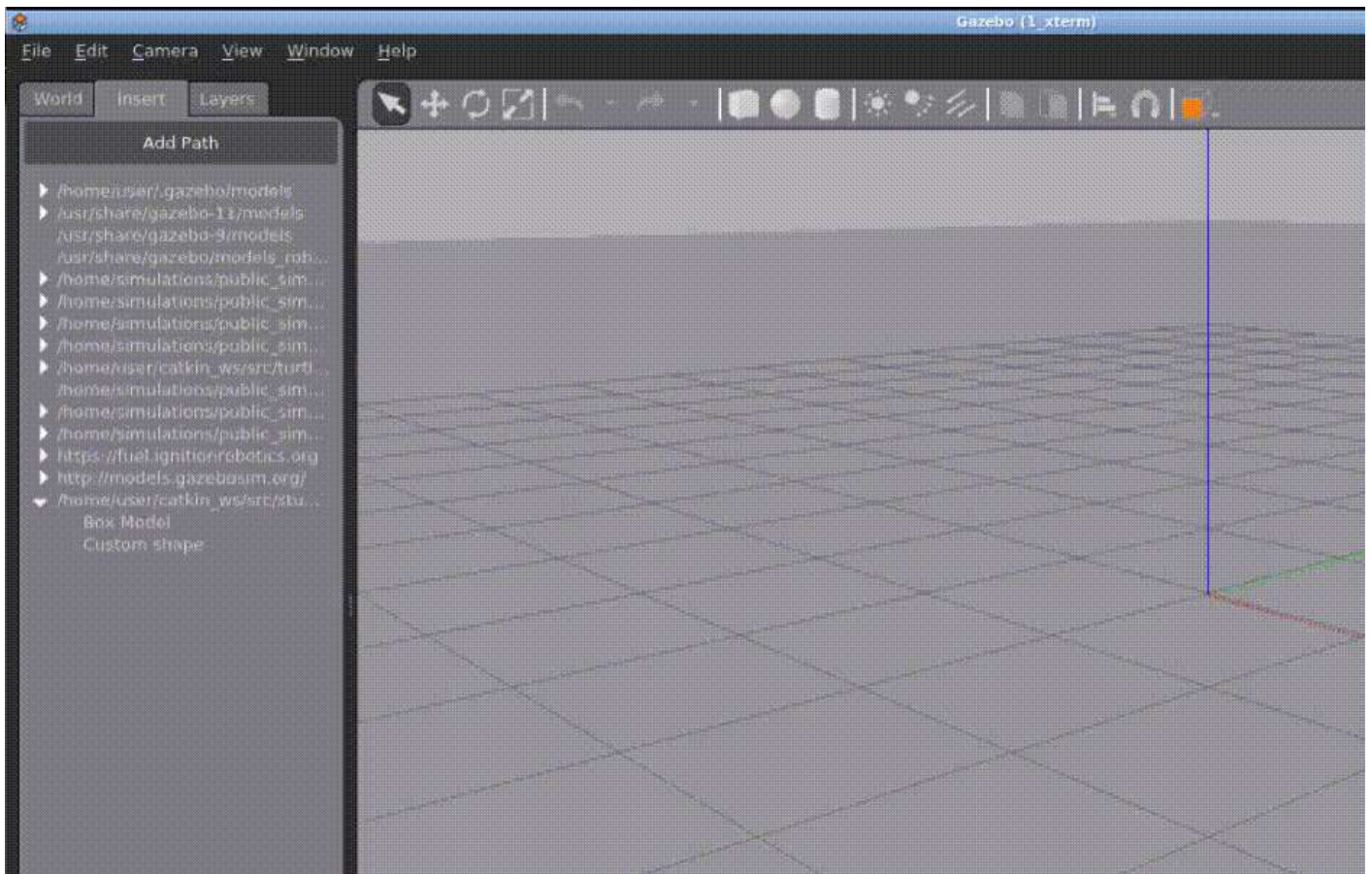
- Use 2 **<visual>** tags, one for **box** geometry and another for **cylinder** geometry. They must have different **name** attributes.
- Use a **<pose>** tag in the second **<visual>** tag to displace it from the object's center.

E.g:

```
<visual name="visual2">
  <pose>0 0 1 0 0 0</pose>
  <geometry>
    <cylinder>
      <radius>0.25</radius>
      <length>1.5</length>
    </cylinder>
  </geometry>
</visual>
```

Check how the new visual part does NOT change the collision!

- Insert the object in the world and check the collision view. The expected result is the following:



- End of Exercise 4.3 -

- Solution 4.3 -

- [model.sdf \(unit-02/model.sdf.txt\)](#)

- End of Solution 4.3 -

4.4 Digital Elevation Models

- Using DEM files to create worlds -

Digital Elevation Model (DEM) is a graphical representation of the bare ground surface. These files are generated using depth images from cameras or LIDAR sensors. Gazebo supports such files to create worlds and simulate more realistic environments.

Create a new world called **dem.world**. You will also need a **DEM** file. Download the file below and uncompress it to the folder:

► Execute

```
In [ ]: touch ~/catkin_ws/src/studying_gazebo/worlds/dem.world
```

```
In [ ]: cd ~/catkin_ws/src/studying_gazebo/worlds
```

```
In [ ]: wget https://github.com/osrf/gazebo_tutorials/raw/master/dem/files/mtsthelens
```

```
In [ ]: unzip mtsthelens_before.zip
```

```
In [ ]: rm mtsthelens_before.zip
```

Now that you have a valid **dem** file, use it in your new **world** file. Paste the following code into the new **dem.world** file:

 Copy and Paste

In []:

```
<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
    </include>

    <model name="heightmap">
      <static>true</static>
      <link name="link">
        <collision name="collision">
          <geometry>
            <heightmap>
              <uri>/home/user/catkin_ws/src/studying_gazebo/worlds/30.1.1.1282
              <size>75 75 25</size>
              <pos>0 0 0</pos>
            </heightmap>
          </geometry>
        </collision>

        <visual name="visual_abcedf">
          <geometry>
            <heightmap>
              <texture>
                <diffuse>file://media/materials/textures/dirt_diffusespecular.
                <normal>file://media/materials/textures/flat_normal.png</norma
                <size>1</size>
              </texture>
              <texture>
                <diffuse>file://media/materials/textures/grass_diffusespecular
                <normal>file://media/materials/textures/flat_normal.png</norma
                <size>1</size>
              </texture>
              <texture>
                <diffuse>file://media/materials/textures/fungus_diffusespecula
                <normal>file://media/materials/textures/flat_normal.png</norma
                <size>1</size>
              </texture>
              <blend>
                <min_height>2</min_height>
                <fade_dist>5</fade_dist>
              </blend>
              <blend>
                <min_height>4</min_height>
```



```


        <fade_dist>5</fade_dist>
    </blend>
    <uri>/home/user/catkin_ws/src/studying_gazebo/worlds/30.1.1.1282
    <size>75 75 25</size>
    <pos>0 0 0</pos>
</heightmap>
</geometry>
</visual>

</link>
</model>

</world>
</sdf>

```

Finally, create a new launch file for this world (**dem.launch**):

 Copy and Paste

In []:

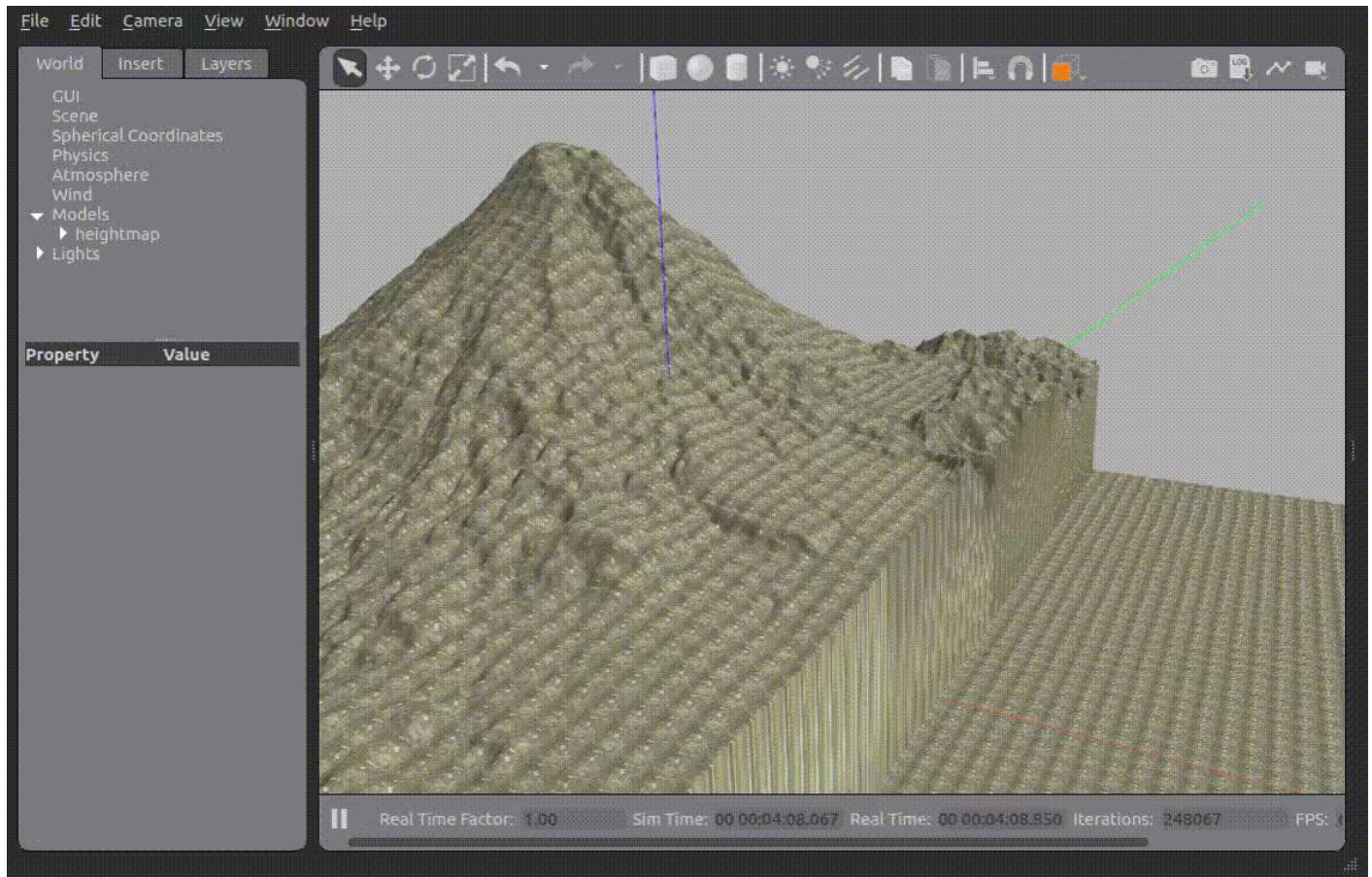
```

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

```



Launch the new world and test the environment. Notice that you defined both **visual** and **collision** using the same heightmap geometry. This is very important so the physics will respect the shape of the ground, which implies in a higher CPU consumption, because you are not simplifying the collisions, like you have done for the robot chassis or even for the model created in this unit.



Reference: http://gazebosim.org/tutorials?tut=dem&cat=build_world (http://gazebosim.org/tutorials?tut=dem&cat=build_world).

- End of Using DEM files to create worlds -

4.5 Population of Models

- Population of models -

The population of models can be used to generate worlds without placing the objects one by one. You can put models randomly or distribute them in a given pattern. See how to do it.

Create two new files, a **world**, and a **launch** file.

► Execute

```
In [ ]: touch ~/catkin_ws/src/studying_gazebo/worlds/population.world
```



In []: touch ~/catkin_ws/src/studying_gazebo/launch/population.launch



And fill the files with the content below:

 Copy and Paste

population.launch

In []:

```
<launch>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find studying_gazebo)/worlds/population.wc
    <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>
```



population.world

In []:

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <world name="default">

    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
    </include>

    <!-- A ground plane -->
    <include>
      <uri>model://ground_plane</uri>
    </include>

    <!-- Testing the automatic population of objects -->
    <population name="barriers_population">
      <model name="jersey_barrier_1">
        <include>
          <static>false</static>
          <uri>model://jersey_barrier</uri>
        </include>
      </model>
      <pose>0 0 0.3 0 0 0</pose>
      <box>
        <size>12 12 0.01</size>
      </box>
      <model_count>3</model_count>
      <distribution>
        <type>linear-x</type>
      </distribution>
    </population>

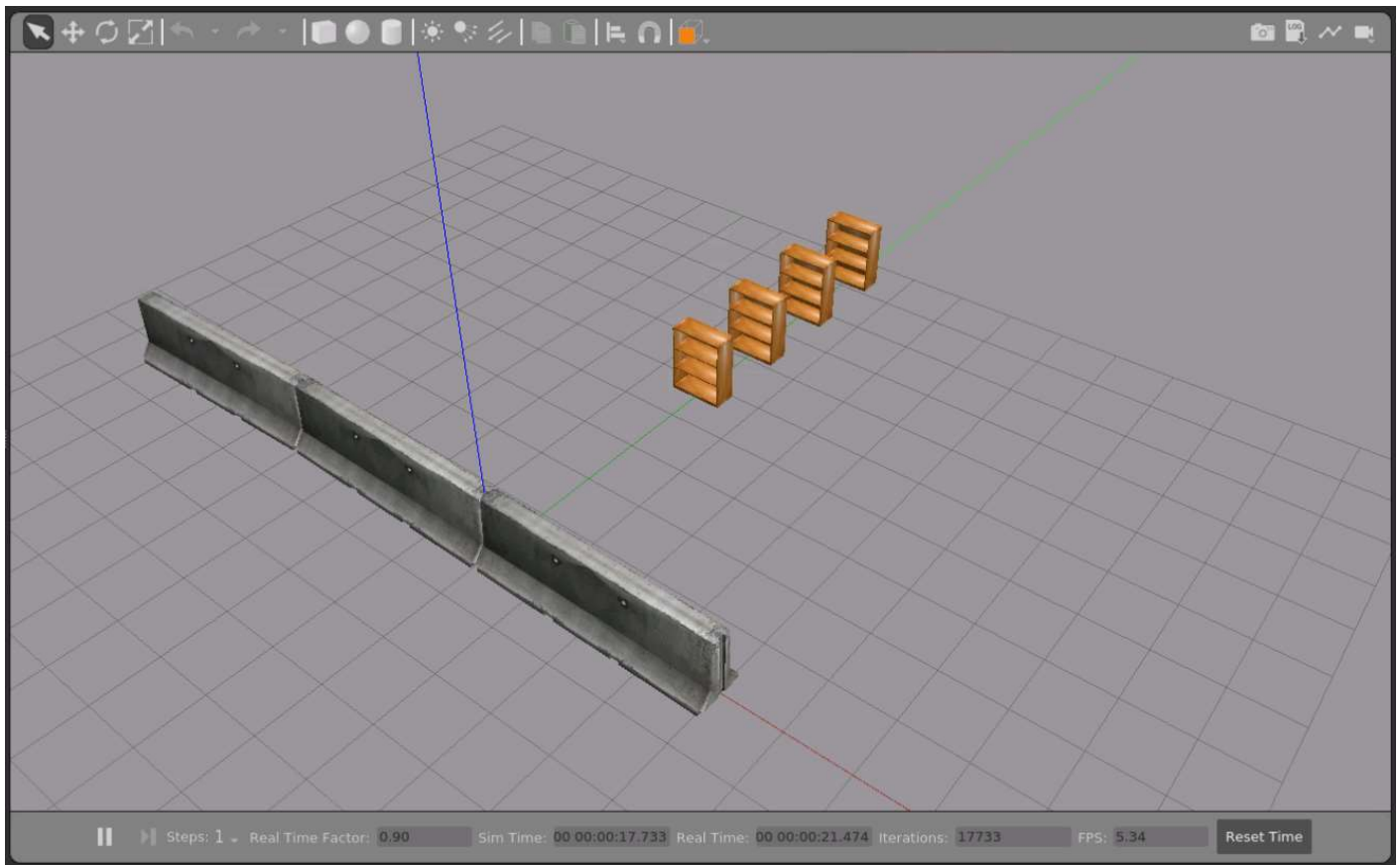
    <!-- Testing the automatic population of objects -->
    <population name="bookshelf_population">
      <model name="bookshelf">
        <include>
          <static>false</static>
          <uri>model://bookshelf</uri>
        </include>
      </model>
      <pose>0 8 0.3 0 0 0</pose>
      <box>
        <size>6 6 0.01</size>
      </box>
      <model_count>4</model_count>
```



```
<distribution>
  <type>linear-y</type>
</distribution>
</population>

</world>
</sdf>
```

Launch that simulation. You must have the following scene:



The syntax is basically:

```
<population name="">
  <!-- The model to be used -->
  <model>
    <include>
    </include>
  </model>
  <!-- Where to start from -->
  <pose></pose>
  <!-- The area to spread the models - could be box or cylinder -->
  <box></box>
  <!-- Amount of objects -->
  <model_count></model_count>
  <distribution>
    <!--
      values:
      random, uniform, grid, linear-x, linear-y, and linear-z
    -->
    <type></type>
  </distribution>
</population>
```

- End of Population of models -

- Practice -

Increase the number of objects and the area of the population box. Change the distribution type to each one of the possibilities:

- random
- uniform
- grid
- linear-x
- linear-y
- linear-z

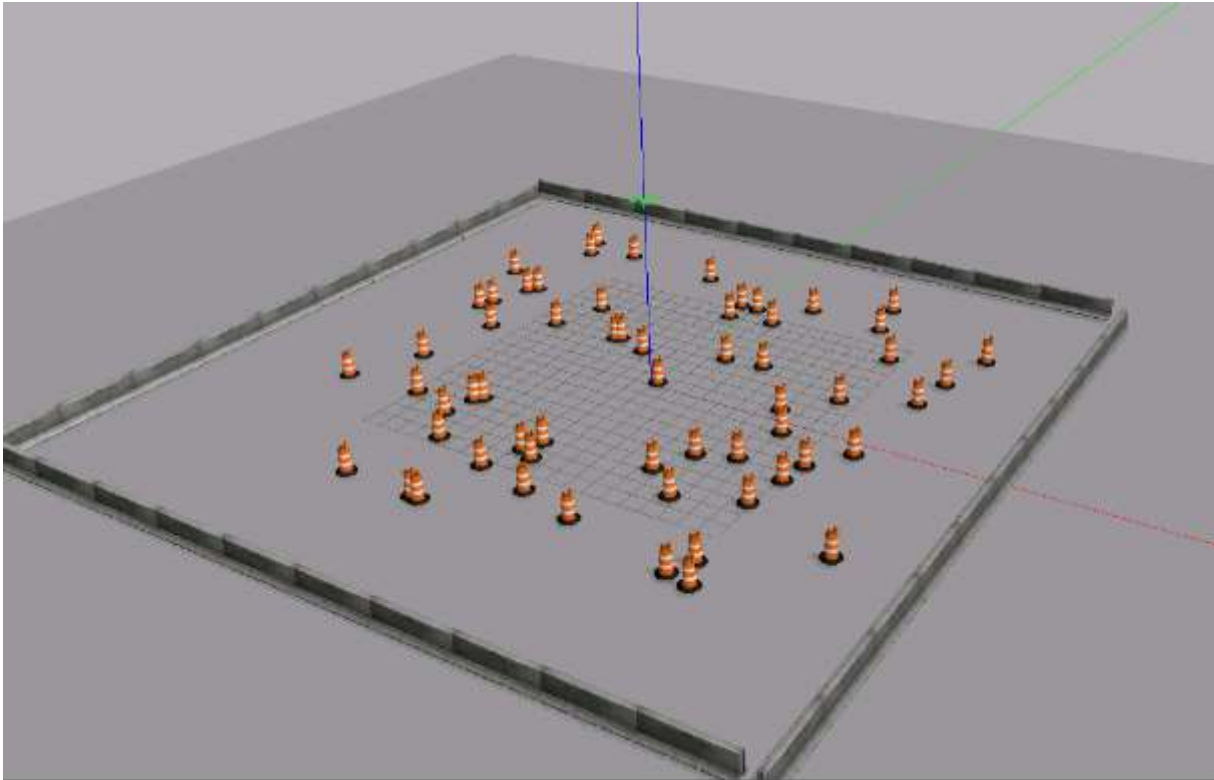
So you will learn and see how they work. You will need that for the next exercise!

- End of Practice -

- Exercise -

Create a scene to help us test a new robot prototype. The simulation must be surrounded by barriers to prevent the mobile robot from going away. It must be in a square of **40 meters x 40 meters**.

In the middle of the square, place some **construction barrels** (model is construction_barrel). It must have 60 objects in random positions. Use a limit of 30x30 meters for these objects, so there is space between the barrels and barriers.



- End of Exercise -

- Solution -

Do not check the solution before trying to complete the exercise on your own! Instead, review the solution when you have finished the exercise. It is an excellent resource to compare with your solution.

If you have problems solving the exercise, use our forum support!

[population.world \(unit-04/population.world.txt\)](#)

- End of Solution -