Gazebo Simulator

Unit 5: Write Gazebo Plugins

- Summary -

Estimated time to completion: **6 hours**

Gazebo plugins add world behaviors, create animations for models, and create new robot sensors inside a simulation.

- End of Summary -

5.1 Preparing the Environment!

- Creating plugins package -

Gazebo plugins are created with C++. Therefore, it can be developed in an independent workspace. However, in this course, you will use a ROS workspace. So, create a new package called **writing_plugins**.

▶ Execute

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

In [ ]:
```
catkin_create_pkg writing_plugins gazebo gazebo_ros gazebo_plugins roscpp
```

Create a new file **~/catkin_ws/src/writing_plugins/src/my_gazebo_plugin.cc**

my_gazebo_plugin.cc

In [ ]:

```
#include <gazebo/gazebo.hh>
#include <gazebo/physics/physics.hh>
namespace gazebo {
    class MyGazeboPlugin : public WorldPlugin {
        public:
        MyGazeboPlugin() : WorldPlugin() {
            printf("Plugin constructor method!\n");
        }

        public:
        void Load(physics::WorldPtr _world, sdf::ElementPtr _sdf) {
            printf("Everything is awesome!\n");
        }
    };

    // Register plugin
    GZ_REGISTER_WORLD_PLUGIN(MyGazeboPlugin)
}
```

The above code defines a class **MyGazeboPlugin** that inherits from **WorldPlugin**, which is a plugin interface for Gazebo's simulation engine.

The constructor of **MyGazeboPlugin** prints a message to the console, and the **Load** method also prints a message.

The **GZ_REGISTER_WORLD_PLUGIN** is a macro call used to register the plugin with Gazebo.

Note: If you are unfamiliar with some of the terms mentioned above, that's okay. These concepts are covered in the Advanced C++ course.

This library must be compiled before being used by a Gazebo world. Add the following instructions to the **~/catkin_ws/src/writing_plugins/CMakeLists.txt** file:

CMakeLists.txt

In [ ]:
```
cmake_minimum_required(VERSION 3.0.2)
project(writing_plugins)

find_package(gazebo REQUIRED)

find_package(catkin REQUIRED COMPONENTS
  gazebo_plugins
  gazebo_ros
  roscpp
)

catkin_package()

include_directories(
# include
  ${catkin_INCLUDE_DIRS}
)

include_directories(${GAZEBO_INCLUDE_DIRS})
link_directories(${GAZEBO_LIBRARY_DIRS})
list(APPEND CMAKE_CXX_FLAGS "${GAZEBO_CXX_FLAGS}")

add_library(my_gazebo_plugin SHARED src/my_gazebo_plugin.cc)
target_link_libraries(my_gazebo_plugin ${GAZEBO_LIBRARIES})
```

And compile the workspace:

▶ Execute

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

In [ ]:
```
catkin_make
```

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

And the plugin is ready to be used.

Export the Gazebo variable below. This is how you tell Gazebo about the new plugin you have created! Do this in the shell whenever you want to launch the simulation, but just once! Your new lib is placed at `~/catkin_ws/devel/lib` .

▶ Execute

In [ ]:
```
export GAZEBO_PLUGIN_PATH=${GAZEBO_PLUGIN_PATH}:~/catkin_ws/devel/lib
```

Create a new world and launch files:

▶ Execute

In [ ]:
```
mkdir -p ~/catkin_ws/src/writing_plugins/launch
```

In [ ]:
```
touch ~/catkin_ws/src/writing_plugins/launch/gazebo.launch
```

In [ ]:
```
mkdir -p ~/catkin_ws/src/writing_plugins/worlds
```

In [ ]:
```
touch ~/catkin_ws/src/writing_plugins/worlds/gazebo.world
```

Paste this content in the launch file:

gazebo.launch

In [ ]:
```
<launch>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find writing_plugins)/worlds/gazebo.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>
```

And the one below for the world file:

gazebo.world

In [ ]:
```
<?xml version="1.0"?>
<sdf version="1.4">
  <world name="default">
    <plugin name="my_gazebo_plugin" filename="libmy_gazebo_plugin.so"/>
  </world>
</sdf>
```

Launch that new simulation and check the plugin working!

▶ Execute

In [ ]:
```
roslaunch writing_plugins gazebo.launch
```

Expected Output

```
Plugin constructor method!
Everything is awesome!
```

Great. You are inside Gazebo! In the following sections, you will learn two different types of plugins and what you can do in each one!

- End of Creating plugins package -

5.2 World Plugin

- Writing a world plugin -

You have just created a world plugin, but so far, you have just printed a simple message in the shell. Now, you see something that can be done on this kind of plugin that can be seen in the simulation itself!

Yet in the file **my_gazebo_plugin.cc**, replace the **Load** method by the following:

+ my_gazebo_plugin.cc

In [ ]:
```
void Load(physics::WorldPtr _world, sdf::ElementPtr _sdf) {
    // set a node
    transport::NodePtr node(new transport::Node());
    node->Init(_world->Name());

    // set publisher
    transport::PublisherPtr publisher =
        node->Advertise<msgs::Factory>("~/factory");

    // create msg obj
    msgs::Factory msg;

    // model to use
    msg.set_sdf_filename("model://jersey_barrier");

    // Send the message
    publisher->Publish(msg);
}
```

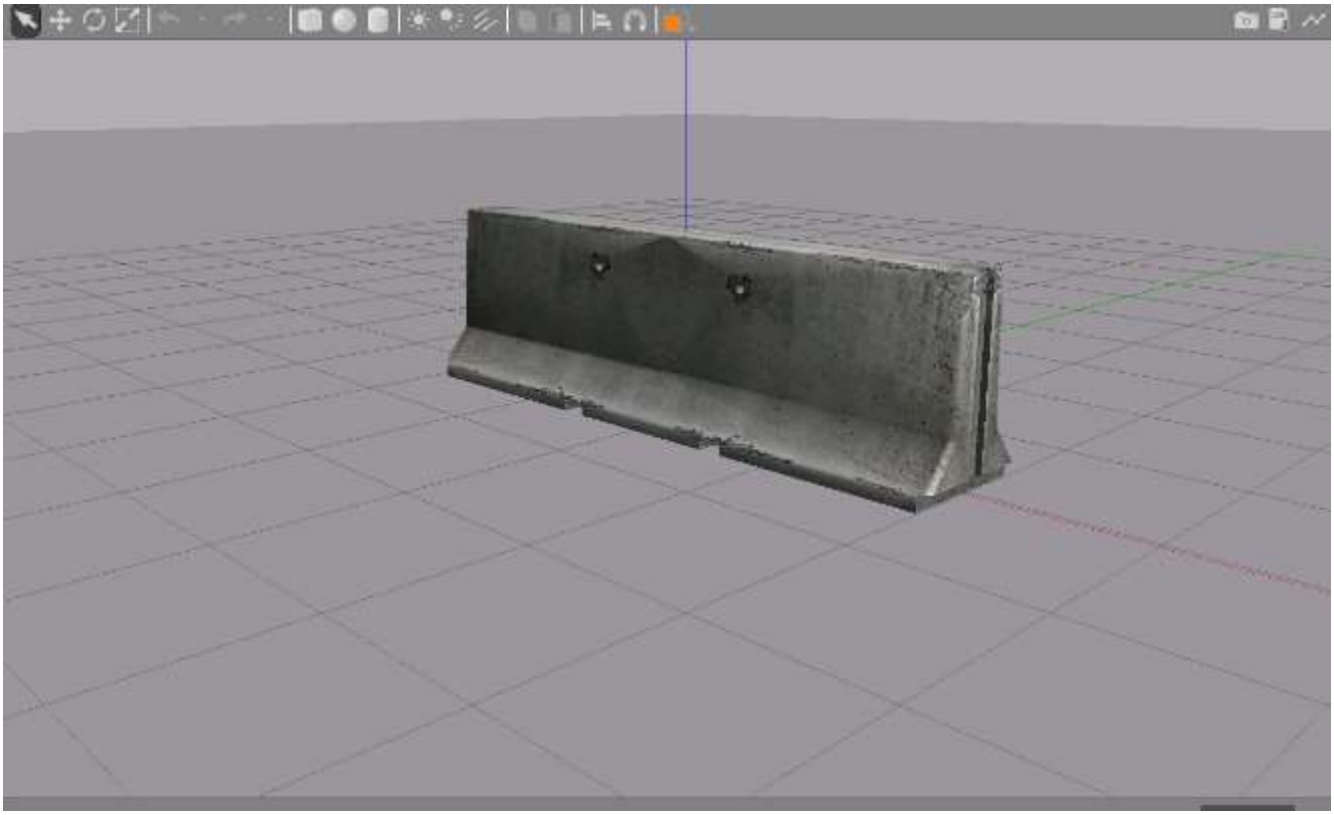
Re-compile the workspace and restart the simulation:

▶ Execute

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

In [ ]:
```
catkin_make
```

In [ ]:
```
roslaunch writing_plugins gazebo.launch
```



- End of Writing a world plugin -

- Exercise 5.2.1. -

Create a new plugin called **world_plugin**.

- Create the file **world_plugin.cc**
- Add the necessary instructions in **CMakeLists.txt**
- Replace the plugin being used in the world file with this new one

You will know this plugin is working because you will add this instruction in the **Load** method, just before `publisher->Publish(msg);` :

```
    // setup new position
    msgs::Set(msg.mutable_pose(),
        ignition::math::Pose3d(
            ignition::math::Vector3d(5, 4, 0),
            ignition::math::Quaterniond(0, 0, 0)
        )
    );
```

This plugin must not only insert the model, but put it in a specific place.

- End of Exercise 5.2.1 -

- Solution to Exercise 5.2.1 -

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!

- world_plugin.cc (unit-05/ex05-1/world_plugin.cpp.txt)
- CMakeLists.txt (unit-05/ex05-1/CMakeLists-txt.txt)
- gazebo.world (unit-05/ex05-1/gazebo.world.txt)

Note: You might need to zoom out a bit so you can see the entire scene.

- End of Solution -

5.3 Model Plugin

- Writing a model plugin -

Creating plugins for models is very similar to world plugins, though you are inheriting from a different class, that will provide you with other methods and objects to work with.

So start creating your first model plugin:

▶ Execute

In [ ]:
```
touch ~/catkin_ws/src/writing_plugins/worlds/model.world
```

In [ ]:
```
touch ~/catkin_ws/src/writing_plugins/src/my_model_plugin.cc
```

In [ ]:
```
vim ~/catkin_ws/src/writing_plugins/src/my_model_plugin.cc
```

my_model_plugin.cc

In [ ]:

```
#include <functional>
#include <gazebo/common/common.hh>
#include <gazebo/gazebo.hh>
#include <gazebo/physics/physics.hh>
#include <ignition/math/Vector3.hh>

namespace gazebo {
class MyModelPlugin : public ModelPlugin {
public:
  void Load(physics::ModelPtr _parent, sdf::ElementPtr /*_sdf*/) {
    // Store the pointer to the model
    this->model = _parent;

    // Listen to the update event. This event is broadcast every
    // simulation iteration.
    this->updateConnection = event::Events::ConnectWorldUpdateBegin(
        std::bind(&MyModelPlugin::OnUpdate, this));
  }

  // Called by the world update start event
public:
  void OnUpdate() {
    // Apply a small linear velocity to the model.

    if (this->counter < 10000) {
      this->model->SetLinearVel(ignition::math::Vector3d(0, 0, 0.4));
    //   this->model->SetLinearAccel(ignition::math::Vector3d(0, 0, 0));
    }

    this->counter++;
  }

  // Pointer to the model
private:
  physics::ModelPtr model;

private:
  int counter;

  // Pointer to the update event connection
private:
  event::ConnectionPtr updateConnection;
};

// Register this plugin with the simulator
```

```
GZ_REGISTER_MODEL_PLUGIN(MyModelPlugin)
} // namespace gazebo
```

And for the **model.world** file:

model.world

In [ ]:

```
<?xml version="1.0"?>
<sdf version="1.4">
  <world name="default">

    <!-- Ground Plane -->
    <include>
      <uri>model://ground_plane</uri>
    </include>

    <include>
      <uri>model://sun</uri>
    </include>

    <model name="box">
      <pose>0 0 0.5 0 0 0</pose>
      <link name="link">
        <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>

      <plugin name="my_model_plugin" filename="libmy_model_plugin.so"/>
    </model>
  </world>
</sdf>
```

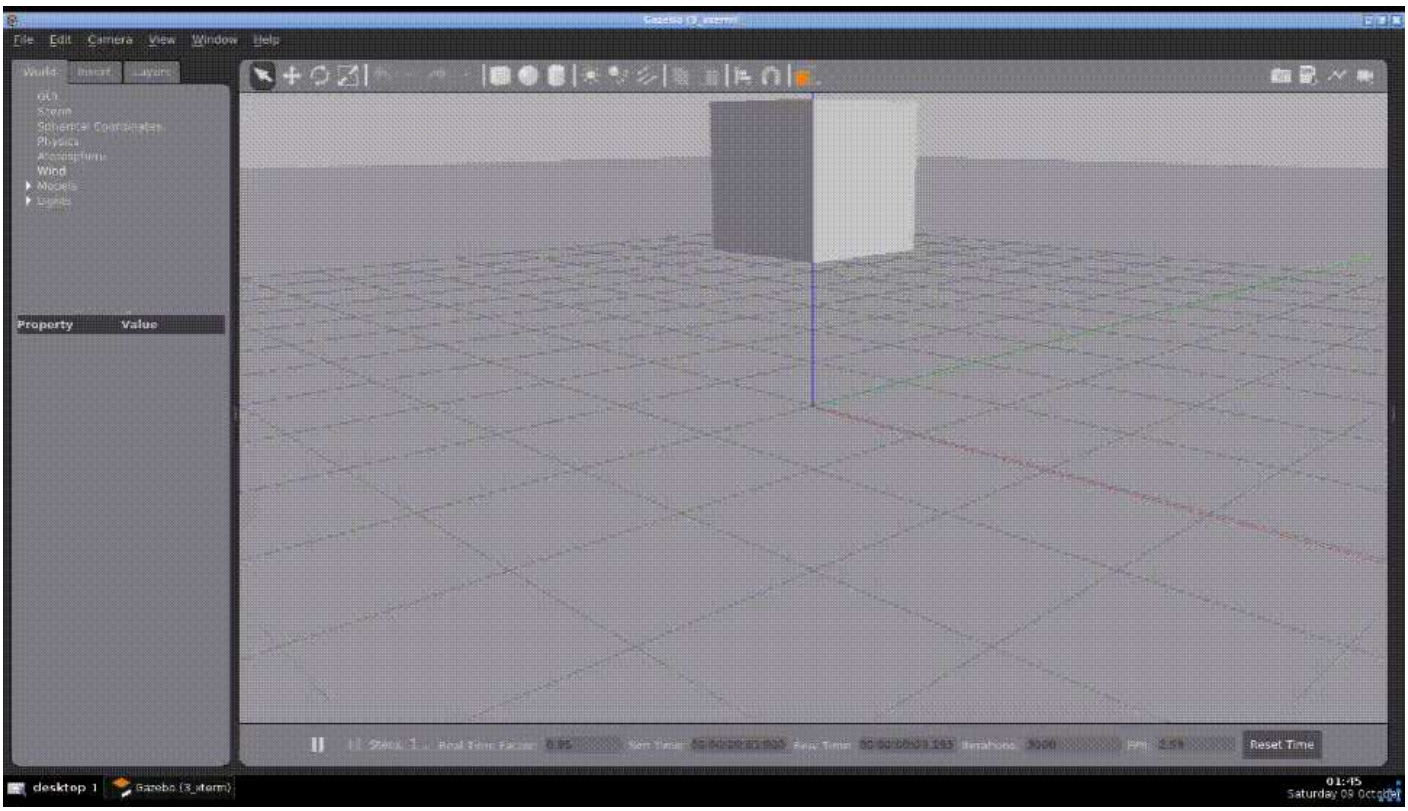
Change the **gazebo.launch** file and restart the simulation:

gazebo.launch

In [ ]:

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

The expected result is:



The first argument of the **Load** method is **_parent**, which is a pointer to the model you spawned.

Then you are using the method **void OnUpdate() {**, which is called for every iteration of the simulation engine. It is only called because you define it as a callback for the iteration cycle:

```
    this->updateConnection = event::Events::ConnectWorldUpdateBegin(
            std::bind(&MyModelPlugin::OnUpdate, this));
```

Check the bottom bar for the number of iterations. You need to be careful with this method because it is called in a higher rate frequency.

Define a **counter** and apply a linear Z velocity to the model, while the counter is below 10.000. Notice that, even if you restart the world (and iterations restarts), the counter does not restart. Take this into account!

Check the methods available for the model object here: [ModelRef API reference (https://osrf-distributions.s3.amazonaws.com/gazebo/api/dev/classgazebo_1_1physics_1_1Model.html#afbeccc662db81edbeb](https://osrf-distributions.s3.amazonaws.com/gazebo/api/dev/classgazebo_1_1physics_1_1Model.html#afbeccc662db81edbeb)

- End of Writing a model plugin -

- Exercise 5.3.1 -

For this exercise, you need to create a new **AModelPlugin** in the file **a_model_plugin.cc**. In order to test your plugin, change the world **model.world** using the template below:

model.world

In [ ]:

```
<?xml version="1.0"?>
<sdf version="1.4">
    <world name="default">
        <!-- Ground Plane -->
        <include>
            <uri>model://ground_plane</uri>
        </include>
        <!-- Sun -->
        <include>
            <uri>model://sun</uri>
        </include>
        <!-- Model -->
        <model name="a_barrier">
            <pose>1 1 0 0 0 0</pose>
            <static>false</static>
            <include>
                <uri>model://construction_barrel</uri>
            </include>
            <plugin name="a_model_plugin" filename="liba_model_plugin.so">
                <linear_vel>0.2</linear_vel>
                <iterations>15000</iterations>
            </plugin>
        </model>
    </world>
</sdf>
```

This new plugin must:

- Make the model to perform a square
- Use the callback OnUpdate and count the iterations to set the direction of the linear velocity
- It must perform a square in a loop, not just once

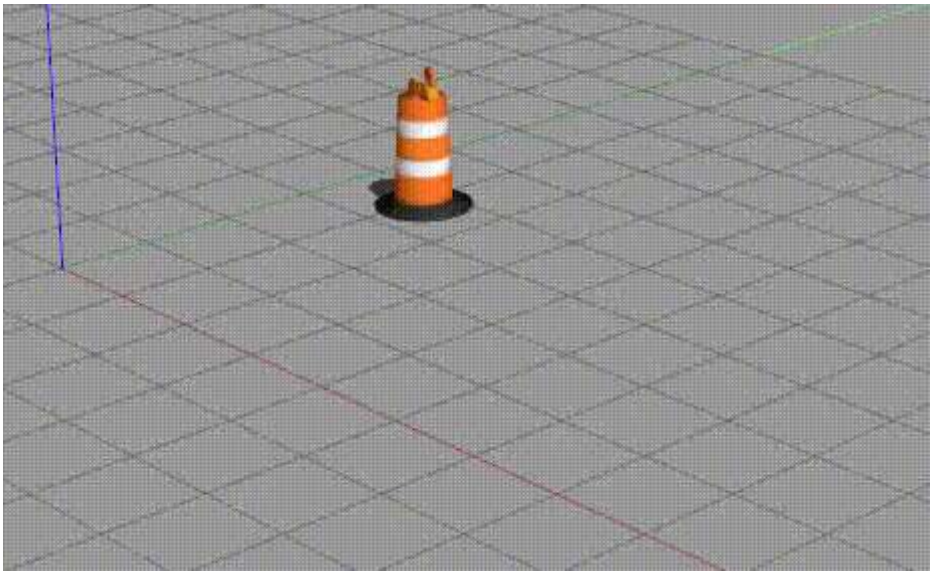
Read the parameters in the world file

In order to do that, use the template below in the **Load** method:

In [ ]:

```
this->iterations = 10 * 1000;
if (_sdf->HasElement("iterations")) {
    this->iterations = _sdf->Get<int>("iterations");
}
```

And use this parameter instead of the `10000` static value. Do the same for **linear_vel**, so you can adjust the same plugin to move many objects with different values and different time duration.



You can read more about parameters in the official API reference of SDF: Reference (http://sdformat.org/tutorials?tut=custom_elements_attributes_proposal)

- End of Exercise 5.3.1. -

- Solution to Exercise 5.3.1 -

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!

a_model_plugin.cc (unit-05/a_model_plugin.cc.txt)

- End of Solution -