ROS GAZEBO INTERACTION

ROBOTICS

POLITECNICO
MILANO 1863

CONNECTING ROS AND GAZEBO

What we already know

Gazebo

-Create a model

-Write a plugin to control model behaviour

-Show published gazebo data (topics)

ROS

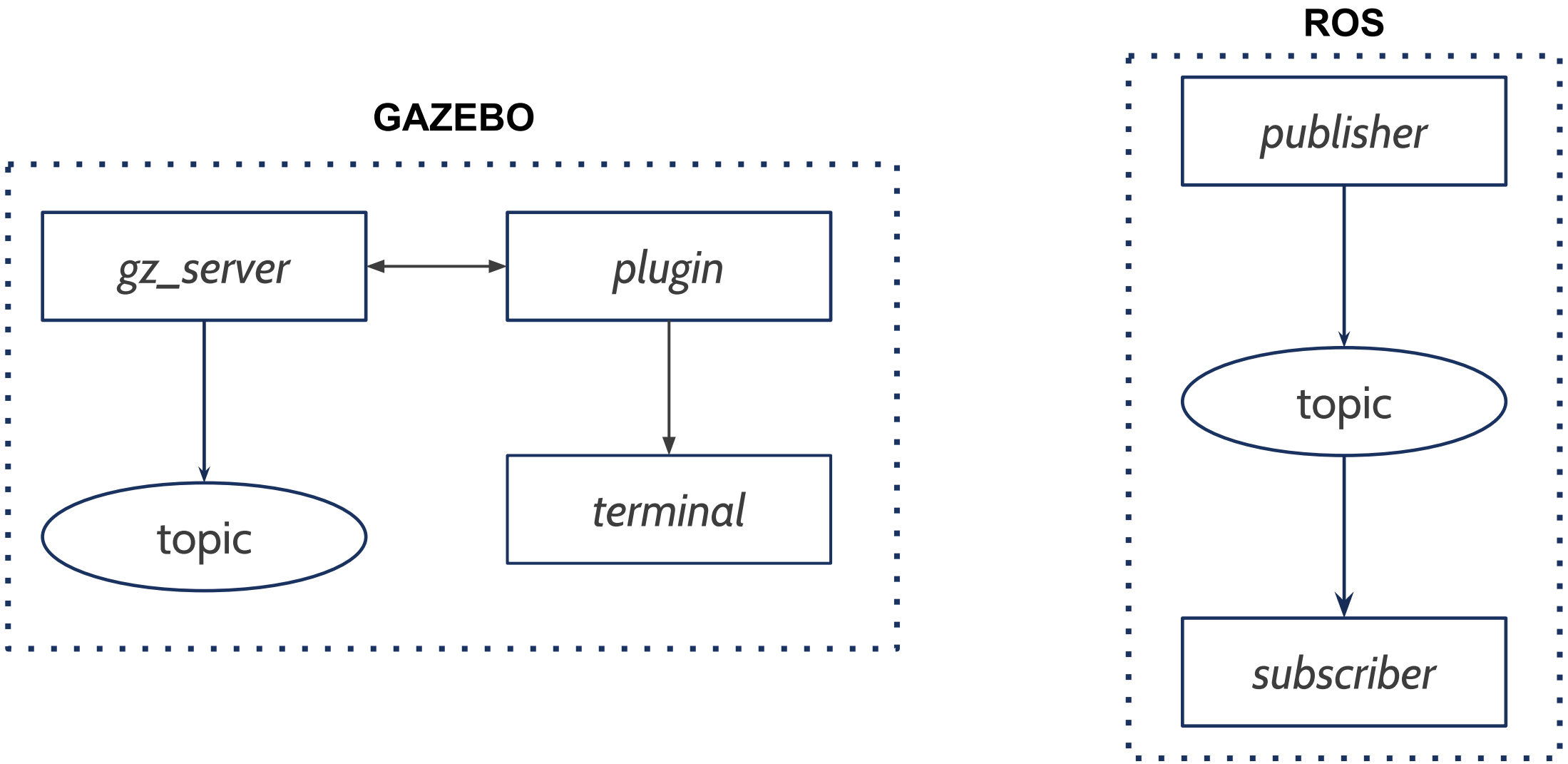
-Write a publisher/subscriber

-Create custom messages

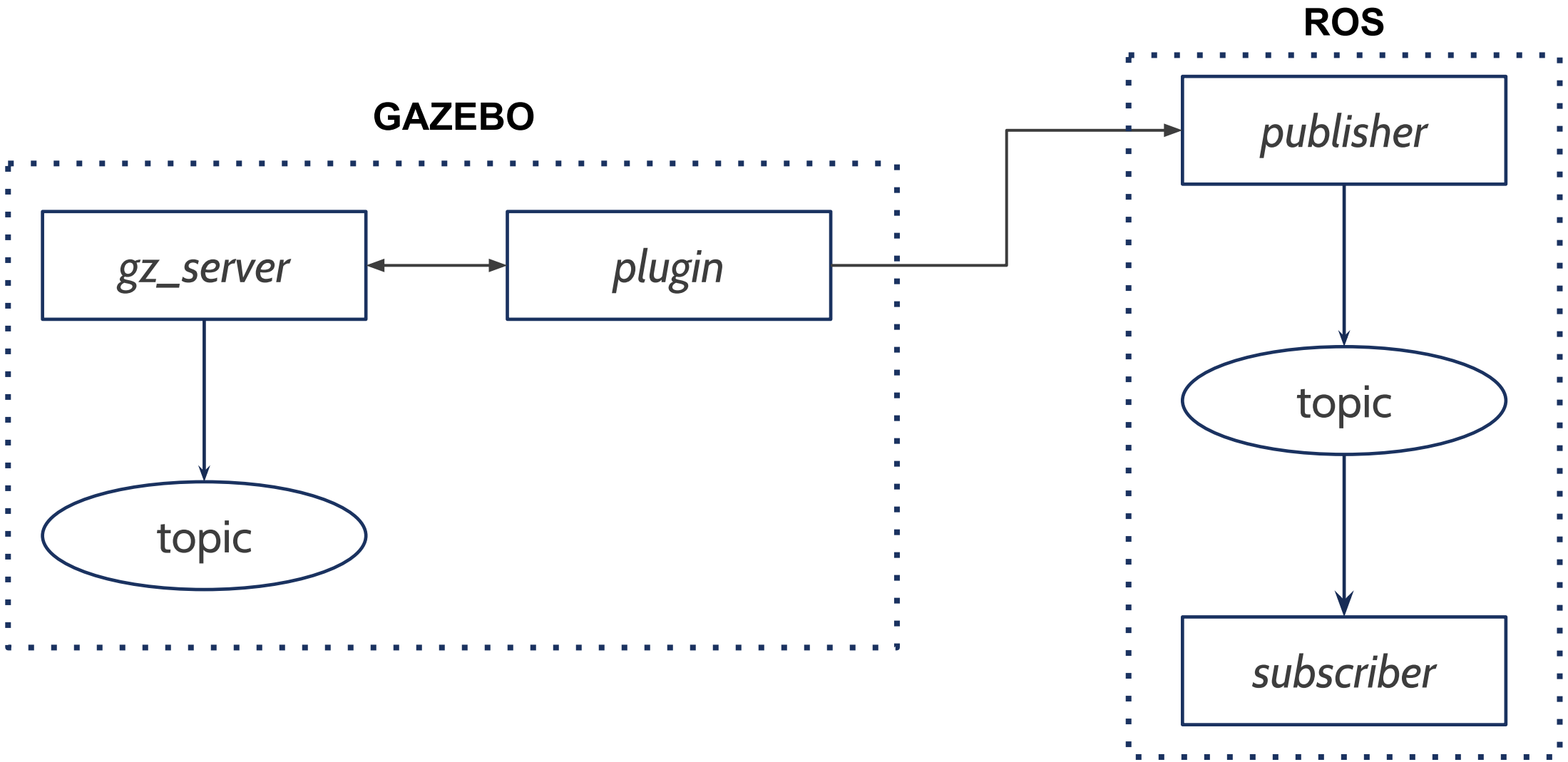
-Analyze data using ROS tools

-Write a launch file

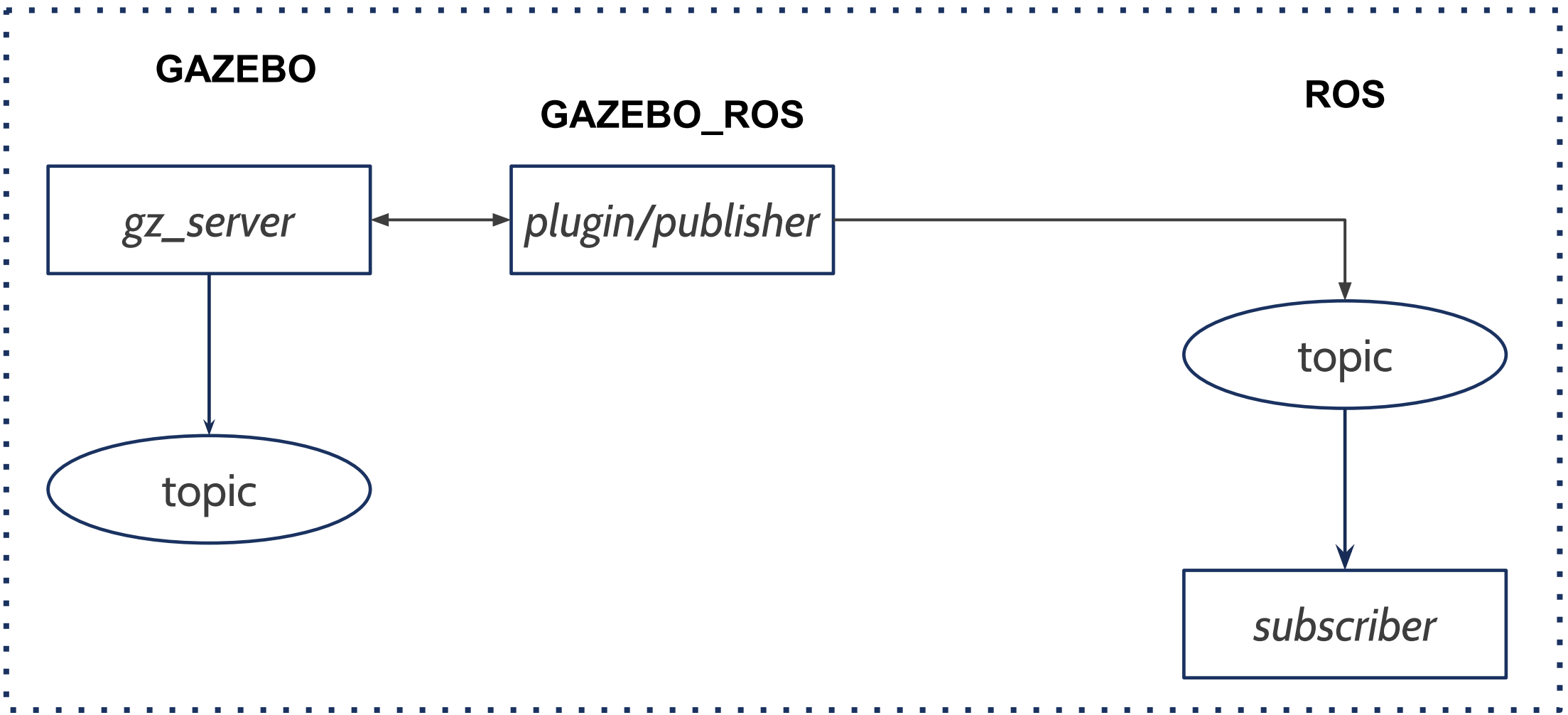
OUR ARCHITECTURE



IMPROVMENT


ROS
GAZEBO
publisher
gz_server
plugin
topic
topic
subscriber

IMPROVMENT


GAZEBO
GAZEBO_ROS
ROS
gz_server
plugin/publisher
topic
topic
subscriber

ROS GAZEBO INTERACTION

Ros offers lots of plugin to interact with the standard sensor of gazebo (like diffdrive)

Most of the time if you are interested in subscribing to a topic published by a gazebo sensor it’s only needed to properly configure a standard plugin

We will test a depth camera with the gazebo_ros plugin

Create inside your gazebo model folder a new folder named streo_cam

Then create a model.sdf file

ROS GAZEBO INTERACTION

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <model name="stereo_cam">
    <pose>0 0 0.036 0 0 0</pose>
    <link name="link">
      <inertial>
        <mass>0.1</mass>
      </inertial>
      <collision name="collision">
        <geometry>
          <box>
            <size>0.073000 0.276000 0.072000</size>
          </box>
        </geometry>
      </collision>
      <visual name="visual">
        <geometry>
        <box>
            <size>0.073000 0.276000 0.072000</size>
          </box>
        </geometry>
      </visual>
```

ROS GAZEBO INTERACTION

```
<sensor name="camera" type="depth">
        <update_rate>20</update_rate>
        <camera>
          <horizontal_fov>1.047198</horizontal_fov>
          <image>
            <width>640</width>
            <height>480</height>
            <format>R8G8B8</format>
          </image>
          <clip>
            <near>0.05</near>
            <far>3</far>
          </clip>
        </camera>
        </sensor>
        </link>
<static>true</static>
  </model>
</sdf>
```

ROS GAZEBO INTERACTION

This was standard SDF, we simply create a box and associated a sensor of type camera to it.

Now we will connect to the camera the openni_kinect plugin; despite the name this plugin is suitable for all depth cameras, simply you need to properly configure all the parameters

To get any information simply check the source code:

http://wiki.ros.org/gazebo_plugins

ROS GAZEBO INTERACTION

```
<plugin name="camera_plugin" filename="libgazebo_ros_openni_kinect.so">

    <baseline>0.2</baseline>
    <alwaysOn>true</alwaysOn>
    <updateRate>0.0</updateRate>
    <cameraName>camera_ir</cameraName>
    <imageTopicName>/camera/depth/image_raw</imageTopicName>
    <cameraInfoTopicName>/camera/depth/camera_info</cameraInfoTopicName>
    <depthImageTopicName>/camera/depth/image_raw</depthImageTopicName>
    <depthImageInfoTopicName>/camera/depth/camera_info</depthImageInfoTopicName>
    <pointCloudTopicName>/camera/depth/points</pointCloudTopicName>
    <frameName>camera_link</frameName>

  </plugin>
```

ROS GAZEBO INTERACTION

And the model.config:

```
<?xml version="1.0" ?>
<model>
    <name>stereo_cam</name>
    <version>1.0</version>
    <sdf version="1.6">model.sdf</sdf>
    <author>
        <name></name>
        <email></email>
    </author>
    <description></description>
</model>
```

ROS GAZEBO INTERACTION

Now we can start gazebo as a ROS node, using command:

$ rosrun gazebo_ros gazebo

or, the two separate command:

$ rosrun gazebo_ros gzserver

$ rosrun gazebo_ros gzclient

sometimes the compact version won’t start while the two oder command will

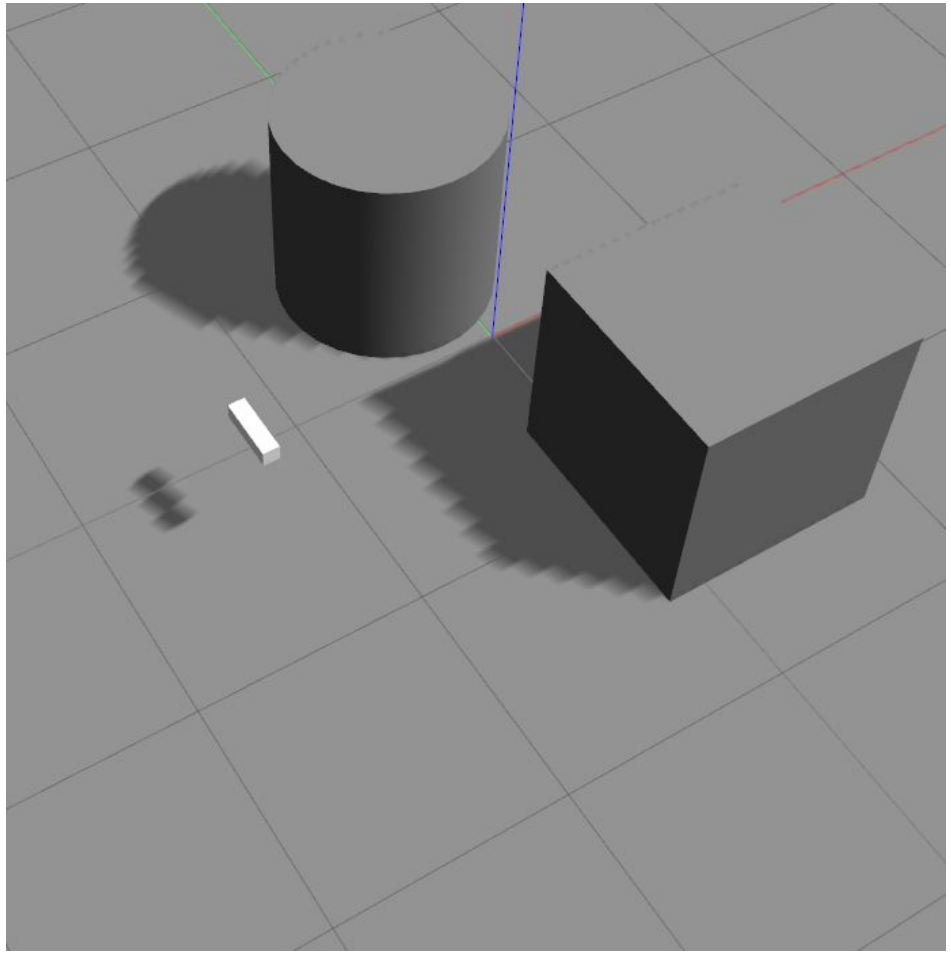
ROS GAZEBO INTERACTION

When you have both the server and the client running insert the camera object and put some obstacles around

then open a new terminal and call:

$ rviz

rviz is a powerful topic viewer shipped with ros desktop

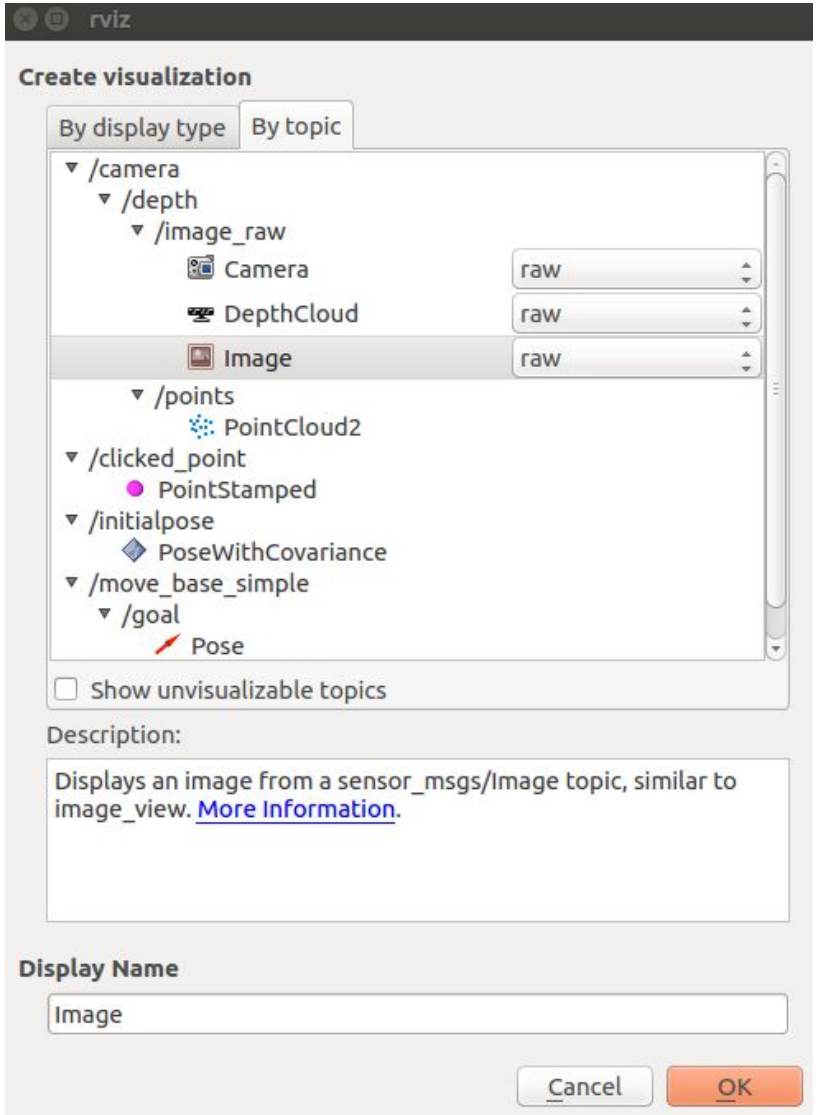


ROS GAZEBO INTERACTION

To visualize the camera topic go to the bottom left of the GUI and use the add button.

Then open the “by topic” tap to see all the published topics

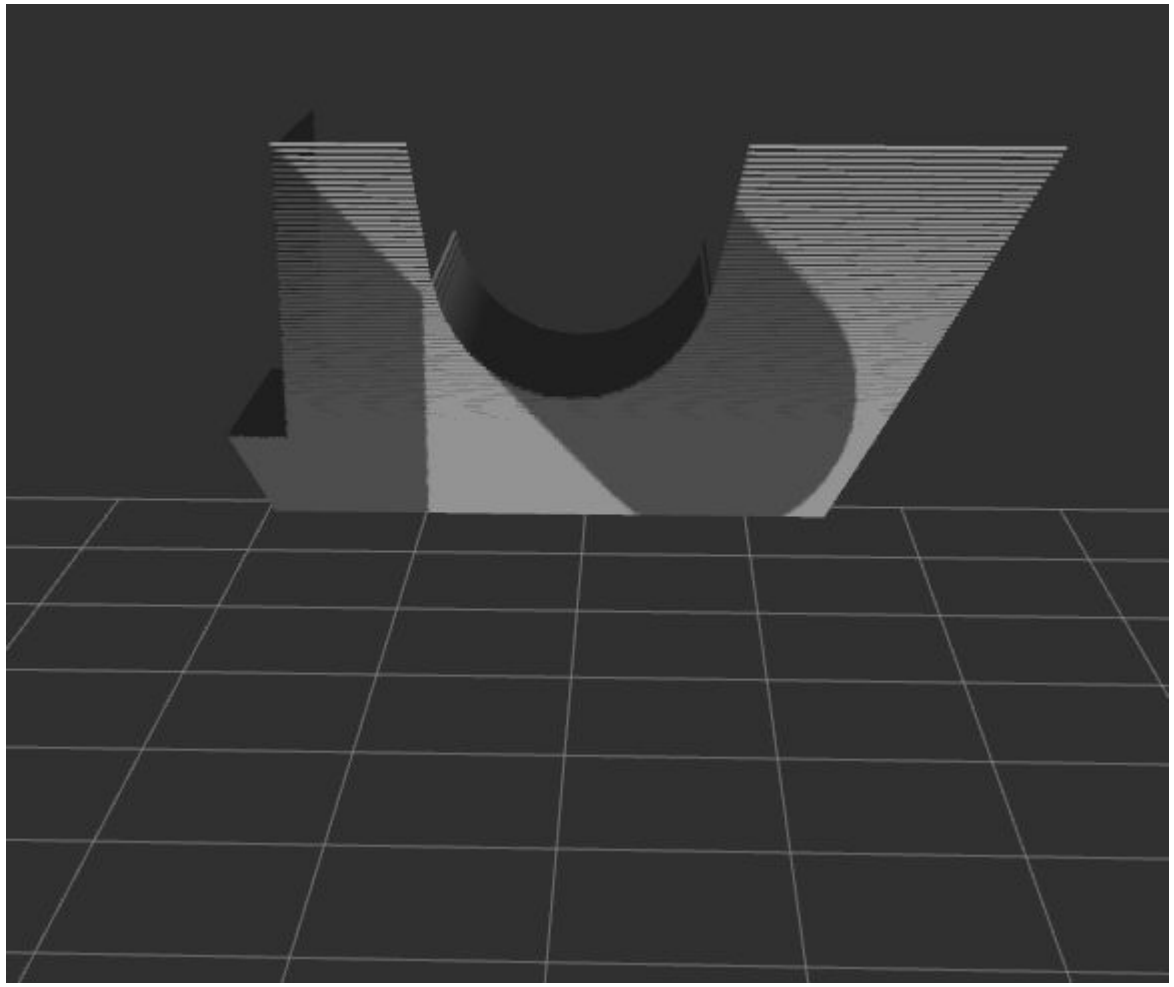
To show the image topic select the “image_raw” topic, then select Image



ROS GAZEBO INTERACTION

To show the point cloud select Points then PointCloud2

To properly visualize it in the user interface we have to specify in the top left of the GUI, under Global Options the Fixed Frame, as wrote in the plugin configuration, in our case “camera_link”



ROS GAZEBO INTERACTION

If you can’t use standard plugin you need to write your own plugin:

-We will write a **gazebo plugin** which contains some **ROS code**

-We write the code inside the ROS workspace

-We will also put the gazebo model inside the ROS workspace

-Start both ROS and Gazebo with a launch file

-We will use catkin to compile everything

ROS GAZEBO INTERACTION

Inside you catkin_enviroment/src create a package called gazebo_hello using:

$ catkin_create_pkg gazebo_hello gazebo_ros roscpp

We already know we are going to write a gazebo plugin, so we add gazebo_ros and roscpp as dependencies

Inside the src folder create a file called: hello.cpp

ROS GAZEBO INTERACTION

We are going to write an hybrid code between ROS and gazebo, so we include:

```
#include <gazebo/common/Plugin.hh>
#include <ros/ros.h>
```

But we are still writing a plugin, so after the include we write:

```
namespace gazebo
{
class WorldPluginHello : public WorldPlugin
{};}
```

ROS GAZEBO INTERACTION

Then we start writing out plugin function, a standard constructor:

```
public:
  WorldPluginHello() : WorldPlugin()
  {
  }
```

And the load function

```
void Load(physics::WorldPtr _world, sdf::ElementPtr _sdf)
  {}
```

ROS GAZEBO INTERACTION

The ROS code will be written inside this load function, first we check if ROS has been initialized:

```
if (!ros::isInitialized())
    {
ROS_ERROR("ROS not initialized");
return;
}
```

ROS GAZEBO INTERACTION

If ROS has been initialized we can print the hello world, simply writing:

```
ROS_INFO("Hello World!");
```

Last, outside the WorldPluginHello class, we have to register out plugin:

```
GZ_REGISTER_WORLD_PLUGIN(WorldPluginHello)
```

THE CODE

```
#include <gazebo/common/Plugin.hh>
#include <ros/ros.h>

namespace gazebo
{
class WorldPluginHello : public WorldPlugin
{
public:
  WorldPluginHello() : WorldPlugin()
  {
  }

  void Load(physics::WorldPtr _world, sdf::ElementPtr _sdf)
  {
    // Make sure the ROS node for Gazebo has already been initialized
    if (!ros::isInitialized())
    {
      ROS_ERROR("ROS not initialized");
      return;
    }

    ROS_INFO("Hello World!");
  }

};
GZ_REGISTER_WORLD_PLUGIN(WorldPluginHello)
}
```

ROS GAZEBO INTERACTION

Now that we have the plugin code we have to make some changes in the CMakeLists.txt file:

First we have to uncomment the line:

```
add_compile_options(-std=c++11)
```

To compile the code using c++11, which is required for gazebo-ros plugin

When we created the package we added the dependencies to gazebo_ros and roscpp, so they are already listed inside the catkin components for the package

ROS GAZEBO INTERACTION

But we have to add these dependencies also to the system dependencies, so we have to uncomment and add:

```
catkin_package(
  DEPENDS
    roscpp
    gazebo_ros
)
```

ROS GAZEBO INTERACTION

This time we are also using some gazebo file, so we also have to add the gazebo package, adding after the command find_package (catkin REQUIRED… the command:

```
find_package(gazebo REQUIRED)
```

Then we specify the path for the linker:

```
link_directories(${GAZEBO_LIBRARY_DIRS})
```

ROS GAZEBO INTERACTION

Now we have to specify the directories for the include file, catkin has already added it's own folder, looking inside the code we can find:

```
include_directories(
# include
  ${catkin_INCLUDE_DIRS}
)
```

ROS GAZEBO INTERACTION

So we simply add:

${GAZEBO_INCLUDE_DIRS}

To also include Gazebo

ROS GAZEBO INTERACTION

Last we have to add at the end of the file the two standard line we always used when we wrote gazebo plugin:

```
add_library(${PROJECT_NAME} src/hello.cpp)
target_link_libraries(${PROJECT_NAME} ${catkin_LIBRARIES}
${GAZEBO_LIBRARIES})
```

ROS GAZEBO INTERACTION

Now that we have modified the CMakeLists.txt we also have to add the line:

```
<gazebo_ros plugin_path="${prefix}/lib" gazebo_media_path="${prefix}" />
```

Inside the package.xml, at the end, between the export tag, to allow gazebo to find out compiled plugin once we start the simulation

ROS GAZEBO INTERACTION

Now we can compile the plugin, as we compiled all the previous ROS packages, cd to the root of you catkin environment and call catkin_make.

Now that we have the plugin we need a world to test it.

Create a **worlds** folder inside the gazebo_hello package and add the file hello.world

ROS GAZEBO INTERACTION

The hello.world file is a simple empty world that call the hello plugin

```
<?xml version="1.0" ?>
<sdf version="1.6">
  <world name="default">
    <include>
      <uri>model://ground_plane</uri>
    </include>

    <!-- reference to your plugin -->
    <plugin name="gazebo_hello" filename="libgazebo_hello.so"/>
  </world>
</sdf>
```

ROS GAZEBO INTERACTION

Now we have to create a launch file to start both ROS and gazebo, using the created world file.

The gazebo library of ros offers a standard launcher which takes as argument a world file. The launcher handles all the ROS initialization for us, so it’s sufficient to call this launcher inside our launch file.

Create a launch folder and the hello.launch file

ROS GAZEBO INTERACTION

Inside the launch file write:

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

to call the empty_world.launch file

ROS GAZEBO INTERACTION

Now to start the simulation we can use the command:

roslaunch gazebo_hello hello.launch

In the terminal we can read the “Hello World!” print.

And with the command rosnode list we can check if gazebo is properly working

SOMETHING MORE INTERESTING

Now that we know how to write ROS code inside a gazebo plugin we can try something more interesting:

-Write a plugin that publish using a ROS topic the minimum distance measured by a laser scanner

-We won't write the model, you can download from the drive folder:

https://goo.gl/GonArW

Inside the folder Gazebo_Ros and copy it in your model folder

GAZEBO ROS INTERACTION

What we will do is write the plugin

-Create a package inside you catkin_ws/src called gazebo_ros:

$ catkin_create_pkg gazebo_ros_plugin gazebo_ros roscpp gazebo_plugins

we start writing some dependencies that are standard when creating gazebo-ros plugins

now cd to the src folder of the new package and create a publisher.h file

GAZEBO ROS INTERACTION

First we write some guard and include:

```
#ifndef PUB_HH
#define PUB_HH

#include "gazebo/common/Plugin.hh"
#include "gazebo/sensors/sensors.hh"
#include "gazebo/common/Events.hh"
#include "gazebo/math/gzmath.hh"
#include <ros/ros.h>
#include <gazebo/sensors/Sensor.hh>
#include <gazebo/sensors/SensorTypes.hh>

#include <std_msgs/Float64.h>
```

GAZEBO ROS INTERACTION

Then we write the prototype of the function we will use, inside the gazebo namespace. We are writing a plugin which interact with a laser sensor, so we inherits from the SensorPlugin class.

```
namespace gazebo {
    class GAZEBO_VISIBLE distPublisher : public SensorPlugin {
        public: distPublisher();
        public: virtual ~distPublisher();
        public: void Load(sensors::SensorPtr _parent, sdf::ElementPtr _sdf);
        protected: virtual void OnUpdate(sensors::RaySensorPtr _sensor);
        protected: sensors::RaySensorPtr parentSensor;
        private: event::ConnectionPtr connection;
```

GAZEBO ROS INTERACTION

Last we create some for the ROS communication and we close the guard

```
        ros::NodeHandle nh;
      ros::Publisher publisher;

    };
}
#endif
```

THE CODE

```
#ifndef PUB_HH
#define PUB_HH

#include "gazebo/common/Plugin.hh"
#include "gazebo/sensors/sensors.hh"
#include "gazebo/common/Events.hh"
#include "gazebo/math/gzmath.hh"
#include <ros/ros.h>
#include <gazebo/sensors/Sensor.hh>
#include <gazebo/sensors/SensorTypes.hh>
#include <std_msgs/Float64.h>

namespace gazebo {
   class GAZEBO_VISIBLE distPublisher : public SensorPlugin {
      public: distPublisher();
      public: virtual ~distPublisher();
      public: void Load(sensors::SensorPtr _parent, sdf::ElementPtr _sdf);
      protected: virtual void OnUpdate(sensors::RaySensorPtr _sensor);
      protected: sensors::RaySensorPtr parentSensor;
      private: event::ConnectionPtr connection;

      ros::NodeHandle nh;
        ros::Publisher publisher;

   };
}
#endif
```

GAZEBO ROS INTERACTION

Now we start writing the cpp file; create a file called publisher.cpp
First we include the header file:

```
#include "publisher.h"
```

Then we specify the namespace and register the plugin:

```
using namespace gazebo;

GZ_REGISTER_SENSOR_PLUGIN(distPublisher)
```

GAZEBO ROS INTERACTION

Next we write the constructor, and initialize our publisher:

```
distPublisher::distPublisher() {
     publisher = nh.advertise<std_msgs::Float64>("distance", 1);
 }
```

And the destructor, which will disconnect from the sensor:

```
distPublisher::~distPublisher() {
    this->parentSensor->DisconnectUpdated(this->connection);
    this->parentSensor.reset();
}
```

GAZEBO ROS INTERACTION

Now we can write the load function:

```
void distPublisher::Load(sensors::SensorPtr _parent, sdf::ElementPtr _sdf) {
  this->parentSensor = std::dynamic_pointer_cast<sensors::RaySensor>(_parent);

  this->connection = this->parentSensor->ConnectUpdated(
            std::bind(&distPublisher::OnUpdate, this, this->parentSensor));
}
```

Which will call the OnUpdate function every time a new data is received from the laser sensor

GAZEBO ROS INTERACTION

The last function to write is the OnUpdate callback where we receive the array with all the measures from the laser scanner.

```
void distPublisher::OnUpdate(sensors::RaySensorPtr _sensor) {

}
```

We want to save the minimum distance to then publish it, so we create a variable min initialized to the max sensor value:

```
 double min = _sensor->RangeMax();
```

GAZEBO ROS INTERACTION

Then we cycle on the received buffer looking for the min value.
To avoid errors during this process it's good practice to turn off the sensor, look for the min value and then turn it on again:

```
_sensor->SetActive(false);
 for (int i=0;i<_sensor->RangeCount();i++){
    if (_sensor->Range(i)<min)
         min=_sensor->Range(i);



 }
   _sensor->SetActive(true);
```

GAZEBO ROS INTERACTION

Last we create a message, in our case a float will be enough, and we publish it:

```
std_msgs::Float64 msg;
msg.data = min;
publisher.publish(msg);
```

THE CODE

```
#include "publisher.h"
using namespace gazebo;

GZ_REGISTER_SENSOR_PLUGIN(distPublisher)
distPublisher::distPublisher()//:
        //nh("robot")
 {
        publisher = nh.advertise<std_msgs::Float64>("distance", 1);
 }

distPublisher::~distPublisher() {
   this->parentSensor->DisconnectUpdated(this->connection);
   this->parentSensor.reset();
}
void distPublisher::Load(sensors::SensorPtr _parent, sdf::ElementPtr _sdf) {
  this->parentSensor = std::dynamic_pointer_cast<sensors::RaySensor>(_parent);

  this->connection = this->parentSensor->ConnectUpdated(
          std::bind(&distPublisher::OnUpdate, this, this->parentSensor));
}
void distPublisher::OnUpdate(sensors::RaySensorPtr _sensor) {

  double min = _sensor->RangeMax();
  _sensor->SetActive(false);
  for (int i=0;i<_sensor->RangeCount();i++){
        if (_sensor->Range(i)<min)
                min=_sensor->Range(i);
  }
    _sensor->SetActive(true);

    std_msgs::Float64 msg;
    msg.data = min;
    publisher.publish(msg);
```

GAZEBO ROS INTERACTION

Now that we have written the plugin code we have to edit the package.xml file.

As we did in the previous tutoria we have to add the line:

```
<gazebo_ros plugin_path="${prefix}/lib" gazebo_media_path="${prefix}" />
```

at the end of the file between the <export> tags

GAZEBO ROS INTERACTION

The CMakeLists.txt file requires more changes; first uncomment the c++11 option:

```
add_compile_options(-std=c++11)
```

The Catkin required components are already satisfied because we specified them when we created the package, but we have to add all the gazebo libraries:

```
find_package(gazebo REQUIRED)
include_directories(${GAZEBO_INCLUDE_DIRS})
link_directories(${GAZEBO_LIBRARY_DIRS})
list(APPEND CMAKE_CXX_FLAGS "${GAZEBO_CXX_FLAGS}")
```

Simply add those line after the find_package call

GAZEBO ROS INTERACTION

Next we have to specify the configuration for dependent project, so we have to add inside the catkin_package command:

```
catkin_package(
  INCLUDE_DIRS include
  CATKIN_DEPENDS gazebo_plugins gazebo_ros roscpp
)
```

GAZEBO ROS INTERACTION

And specify the include directories:

```
include_directories( ${catkin_INCLUDE_DIRS}
                     ${Boost_INCLUDE_DIR}
                     ${GAZEBO_INCLUDE_DIRS}
)
```

GAZEBO ROS INTERACTION

Last we add the plugin source file

```
add_library(${PROJECT_NAME} src/publisher.cpp)
target_link_libraries( ${PROJECT_NAME} ${catkin_LIBRARIES}
${GAZEBO_LIBRARIES} RayPlugin )
```

Now we can cd to the root of the workspace and compile the plugin

GAZEBO ROS INTERACTION

The model file already include this plugin (if you didn't change his name):

<plugin name="distPublisher" filename="libgazebo_ros_plugin.so"/>

So we can start gazebo as a ros node

$ rosrun gazebo_ros gzserver
$ rosrun gazebo_ros gzclient

GAZEBO ROS INTERACTION

Insert the model and put a random object in front of it.

Now opening a terminal type:

$ rostopic echo /distance

and you should see the minimum distance measured by the laser scanner

GAZEBO ROS INTERACTION

Writing the subscriber node

The Gazebo node already publish and subscribe to some topics.

To add a new subscriber we have to properly configure the plugin avoiding conflict with the existing routine of gazebo_ros

We can’t simply create a subscriber object and call a callback.

Create a new package:

catkin_create_pkg gazebo_sub gazebo_ros roscpp

GAZEBO ROS INTERACTION

Now we create the plugin file inside the src folder, called sub.cpp. As usual some include:

```
#include <gazebo/common/Plugin.hh>
#include <ros/ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Float32.h>
#include "ros/callback_queue.h"
#include "ros/subscribe_options.h"
#include <thread>
```

GAZEBO ROS INTERACTION

Next some standard gazebo plugin code:

```
namespace gazebo
{
    class Subscriber : public WorldPlugin{

        public:
        Subscriber() : WorldPlugin(){
        }


    };
}
```

GAZEBO ROS INTERACTION

Now we create some variables which will be used to initialize the ros node, the subscriber node and handle all the data

```
// A node use for ROS transport
private: std::unique_ptr<ros::NodeHandle> rosNode;
// A ROS subscriber and publisher
private: ros::Subscriber rosSub;
// A ROS callbackqueue that helps process messages
private: ros::CallbackQueue rosQueue;
// A thread the keeps running the rosQueue
private: std::thread rosQueueThread;
```

GAZEBO ROS INTERACTION

Next we write the Load function, first we check if Ros is initialized:

```
void Load(physics::WorldPtr _world, sdf::ElementPtr _sdf){
        if (!ros::isInitialized()){
            ROS_ERROR("ROS not initialized");
            return;
        }
```

GAZEBO ROS INTERACTION

Next we create a pointer the node handle:

```
rosNode.reset(new ros::NodeHandle("gazebo_client"));
```

And we create a subscriber option:

```
ros::SubscribeOptions so =
ros::SubscribeOptions::create<std_msgs::String>("/publisher",
1,boost::bind(&Subscriber::OnRosMsg, this, _1), ros::VoidPtr(),&rosQueue);
```

GAZEBO ROS INTERACTION

Now we can subscribe using:

```
rosSub = rosNode->subscribe(so);
```

e chiamare la funzione di gestione della coda:

```
rosQueueThread =std::thread(std::bind(&Subscriber::QueueThread, this));
```

GAZEBO ROS INTERACTION

Now we have to write both the callback function and the queue handler, starting with the callback:

```
public: void OnRosMsg(const std_msgs::StringConstPtr &msg) {
            ROS_INFO ("Received data: %s", msg->data.c_str());
        }
```

GAZEBO ROS INTERACTION

And the queue handler:

```
private: void QueueThread() {
            static const double timeout = 0.1;
            while (rosNode->ok()) {
                rosQueue.callAvailable(ros::WallDuration(timeout));
            }
        }
```

Last remember to register the plugin:

```
GZ_REGISTER_WORLD_PLUGIN(Subscriber)
```

THE CODE

```
#include <gazebo/common/Plugin.hh>
#include <ros/ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Float32.h>
#include "ros/callback_queue.h"
#include "ros/subscribe_options.h"
#include <thread>
namespace gazebo
{
	class Subscriber : public WorldPlugin{

		public:
		Subscriber() : WorldPlugin(){
		}
		private: std::unique_ptr<ros::NodeHandle> rosNode;
		private: ros::Subscriber rosSub;
		private: ros::CallbackQueue rosQueue;
		private: std::thread rosQueueThread;
		void Load(physics::WorldPtr _world, sdf::ElementPtr _sdf){
			if (!ros::isInitialized()){
				ROS_ERROR("ROS not initialized");
				return;
			}
			rosNode.reset(new ros::NodeHandle("gazebo_client"));

			ros::SubscribeOptions so = ros::SubscribeOptions::create<std_msgs::String>("/publisher", 1,boost::bind(&Subscriber::OnRosMsg, this, _1), ros::VoidPtr(),&rosQueue);
			rosSub = rosNode->subscribe(so);
			rosQueueThread =std::thread(std::bind(&Subscriber::QueueThread, this));

		}
		public: void OnRosMsg(const std_msgs::StringConstPtr &msg) {
			ROS_INFO ("Received data: %s", msg->data.c_str());
		}

		// ROS helper function that processes messages
		private: void QueueThread() {
			static const double timeout = 0.1;
			while (rosNode->ok()) {
				rosQueue.callAvailable(ros::WallDuration(timeout));
			}
		}


	};
GZ_REGISTER_WORLD_PLUGIN(Subscriber)

}
```

GAZEBO ROS INTERACTION

Now we have to write our publisher node, we won't write it as a plugin, so the code is identical to the first publisher example:

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>
int main(int argc, char **argv){
  ros::init(argc, argv, "publisher");
  ros::NodeHandle n;
  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("publisher", 1000);
  ros::Rate loop_rate(10);
  while (ros::ok())
  {
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world ";
    msg.data = ss.str();
    ROS_INFO("%s", msg.data.c_str());
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();

  }


  return 0;
}
```

GAZEBO ROS INTERACTION

As usual we have to make some changes to the CMakeLists.txt file:

-Uncomment the c++11 line
add_compile_options(-std=c++11)

-Add the gazebo dependencies

find_package(gazebo REQUIRED)
link_directories(${GAZEBO_LIBRARY_DIRS})

GAZEBO ROS INTERACTION

Specify the include directories:

```
include_directories(
  ${catkin_INCLUDE_DIRS}
  ${GAZEBO_INCLUDE_DIRS}
)
```

and add both files:

```
add_library(${PROJECT_NAME} src/sub.cpp)
target_link_libraries(${PROJECT_NAME} ${catkin_LIBRARIES}
${GAZEBO_LIBRARIES})
add_executable(pub src/pub.cpp)
target_link_libraries(pub ${catkin_LIBRARIES})
add_dependencies(pub gazebo_sub_generate_messages_cpp)
```

GAZEBO ROS INTERACTION

We also have to change the package.xml adding the line:

<gazebo_ros plugin_path="${prefix}/lib" gazebo_media_path="${prefix}" />

between the <export> tags

GAZEBO ROS INTERACTION

As we did with the hello world example we will create a custom world to start the plugin and then launch it with a launch file, create a worlds folder and the file sub.world:

```
<?xml version="1.0" ?>
<sdf version="1.6">
  <world name="default">
    <include>
      <uri>model://ground_plane</uri>
    </include>

    <!-- reference to your plugin -->
    <plugin name="gazebo_hello" filename="libgazebo_sub.so"/>
  </world>
</sdf>
```

GAZEBO ROS INTERACTION

Then create the launch folder and the sub.launch file

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

GAZEBO ROS INTERACTION

Now that we have created all the files we can cd to the root of the catkin workspace and compile it

Than call the launchfile to start the world

$ roslaunch gazebo_sub sub.launch

And then run the publisher node

$ rosrun gazebo_sub pub