⬅ Back (/tutorials?cat=)

Intermediate: Connect to ROS

Recap

Our Velodyne sensor is fully functional, but we don't have any hooks to a robot middleware, like ROS. One of the benefits of using Gazebo with ROS is that it's easy to switch between the real-world and the simulated. In order to make this happen, we need to have our sensor play nicely with the ROS ecosystem.

Add ROS transport

We will modify our current plugin to include the ROS transport mechanism, in a similar manner to how we add the Gazebo transport mechanism in the previous tutorial.

We are assuming that ROS is currently installed (http://ros.org/install) on your system.

1. Add a header files to the `velodyne_plugin.cc` file.

```
#include <thread>
#include "ros/ros.h"
#include "ros/callback_queue.h"
#include "ros/subscribe_options.h"
#include "std_msgs/Float32.h"
```

2. Add a few member variables to the plugin.

```
/// \brief A node use for ROS transport
private: std::unique_ptr<ros::NodeHandle> rosNode;

/// \brief A ROS subscriber
private: ros::Subscriber rosSub;

/// \brief A ROS callbackqueue that helps process messages
private: ros::CallbackQueue rosQueue;

/// \brief A thread the keeps running the rosQueue
private: std::thread rosQueueThread;
```

3. At the end of the `Load` function, add the following.

```
// Initialize ros, if it has not already bee initialized.
if (!ros::isInitialized())
{
  int argc = 0;
  char **argv = NULL;
  ros::init(argc, argv, "gazebo_client",
      ros::init_options::NoSigintHandler);
}

// Create our ROS node. This acts in a similar manner to
// the Gazebo node
this->rosNode.reset(new ros::NodeHandle("gazebo_client"));

// Create a named topic, and subscribe to it.
ros::SubscribeOptions so =
  ros::SubscribeOptions::create<std_msgs::Float32>(
      "/" + this->model->GetName() + "/vel_cmd",
      1,
      boost::bind(&VelodynePlugin::OnRosMsg, this, _1),
      ros::VoidPtr(), &this->rosQueue);
this->rosSub = this->rosNode->subscribe(so);

// Spin up the queue helper thread.
this->rosQueueThread =
  std::thread(std::bind(&VelodynePlugin::QueueThread, this));
```

4. If you read through the code, you'll notice that we need two new functions: **OnRosMsg** and **QueueThread**. Let's add those now.

```
/// \brief Handle an incoming message from ROS
/// \param[in] _msg A float value that is used to set the velocity
/// of the Velodyne.
public: void OnRosMsg(const std_msgs::Float32ConstPtr &_msg)
{
  this->SetVelocity(_msg->data);
}

/// \brief ROS helper function that processes messages
private: void QueueThread()
{
  static const double timeout = 0.01;
  while (this->rosNode->ok())
  {
    this->rosQueue.callAvailable(ros::WallDuration(timeout));
  }
}
```

5. The last item to take care of is the cmake build.

    1. Open **CMakeLists.**txt .
    2. Modify the top portion of the file to look like the following.

```
cmake_minimum_required(VERSION 2.8 FATAL_ERROR)

find_package(roscpp REQUIRED)
find_package(std_msgs REQUIRED)
include_directories(${roscpp_INCLUDE_DIRS})
include_directories(${std_msgs_INCLUDE_DIRS})
```

3. Modify the plugin's target link libraries.

```
target_link_libraries(velodyne_plugin ${GAZEBO_LIBRARIES} ${roscpp_LIBRARIES})
```

4. Your `CMakeLists.txt` should now look like this.

```
cmake_minimum_required(VERSION 2.8 FATAL_ERROR)

find_package(roscpp REQUIRED)
find_package(std_msgs REQUIRED)
include_directories(${roscpp_INCLUDE_DIRS})
include_directories(${std_msgs_INCLUDE_DIRS})

# Find Gazebo
find_package(gazebo REQUIRED)
include_directories(${GAZEBO_INCLUDE_DIRS})
link_directories(${GAZEBO_LIBRARY_DIRS})
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${GAZEBO_CXX_FLAGS}")

# Build our plugin
add_library(velodyne_plugin SHARED velodyne_plugin.cc)
target_link_libraries(velodyne_plugin ${GAZEBO_LIBRARIES} ${roscpp_LIBRARIES})

# Build the stand-alone test program
add_executable(vel vel.cc)

if (${gazebo_VERSION_MAJOR} LESS 6)
  include(FindBoost)
  find_package(Boost ${MIN_BOOST_VERSION} REQUIRED system filesystem regex)
  target_link_libraries(vel ${GAZEBO_LIBRARIES} ${Boost_LIBRARIES})
else()
  target_link_libraries(vel ${GAZEBO_LIBRARIES})
endif()
```

6. Make sure you've sourced ROS:

```
source /opt/ros/<DISTRO>/setup.bash
```

7. Recompile the plugin.

```
cd ~/velodyne_plugin/build
cmake ../
make
```

Control Velodyne from ROS

We can now load the Gazebo plugin as usual, and it will listen on ROS topic for incoming float messages. These messages will then be used to set the Velodyne's rotational speed.

1. Start `roscore`

```
source /opt/ros/<DISTRO>/setup.bash
roscore
```

2. In a new terminal, start Gazebo

```
cd ~/velodyne_plugin/build
source /opt/ros/<DISTRO>/setup.bash
gazebo ../velodyne.world
```

3. In a new terminal, use `rostopic` to send a velocity message.

```
source /opt/ros/<DISTRO>/setup.bash
rostopic pub /my_velodyne/vel_cmd std_msgs/Float32 1.0
```

4. Change the last number of the above command to set different velocities.

Conclusion

Congratulations, you now have the tools to build custom models, share your models, and generate a public API. Have fun, and happy simulating!