ROS Basics in 5 days (C++)

Unit 0 Course Preview

- Summary -

Estimated time of completion: **10 minutes**

This Unit is a Preview of the ROS in 5 Days Course. You'll get a glimpse of what you'll learn during the Course with an introduction to the different parts, and also by seeing some simple examples of each part.

- End of Summary -

So you need to learn ROS, and you want to do it fast, right? I assume that's why you are here, in the ROS in 5 Days Course. Don't worry, I assure you that by the end of the course you'll feel like you can do anything with your robots! But, it won't be easy, and you'll have to do your best!

In this Course Preview, you'll get a very short introduction to the different units that you will be covering during the course. This unit is not meant to teach you ROS in 30 minutes. It's not that easy! You'll just have an overview of the main things that you'll learn during the course. It will be by completing the different chapters of this course that you'll be able to learn how to program any robot.

But what will you need to learn in order to be able to program any robot? Have a look at this unit and you'll see!

0.1 What do you need to learn to program a robot with ROS?

0.1.1 First, you'll need to learn some Basic Concepts

Before starting to learn anything, you always need to first be introduced to some basic concepts. For instance, **you can't learn to run before you know how to walk**. And you can't learn to sing before you know how to talk. Right? So, with this course, it won't be any different.

In the first unit of this course (the Basic Concepts chapter), you'll be introduced to some ROS Basic Concepts that you need to learn before you start working on more complex concepts.

0.1.2 Second, you'll need to know about Topics

The very first thing you need to know in order to learn ROS is how to work with **topics**. Topics are, probably, the most important part of ROS, so you will need to learn them carefully.

ROS handles almost all its communications through topics. Even more complex communication systems, such as services or actions, rely, at the end, on topics. That's why they are so impoartant! Through ROS topics, you will, for instance, be able to communicate with your robot in order to make it move, to read your robot's sensor readings, and more.

In the top right corner of your screen, you have the simulation window. In this window, you will meet different robots during the course (depending on the chapter you are on) that will help you in the process of learning ROS. In this first chapter, (Course Preview), you'll be working with the lovely Kobuki. I'm sure you'll get along great!

So now that we've made the proper introductions... let's start with the example!

- Example 1.1 -

Execute the following command in WebShell number #1 in order to start moving the Kobuki robot. WebShells are like Linux shells, but on the web. They are located at the bottom right. You can type Linux commands there.

▶ Execute in Shell #1

In [ ]:
```
roslaunch publisher_example move.launch
```

NOTE 1: In order to stop the program, just press Ctrl + C in the WebShell.

NOTE 2: Even after stopping the program, the robot will still keep moving. In order to stop it, you will have to execute another command.

Now select WebShell #2 and execute the following command in order to stop the Kobuki robot.

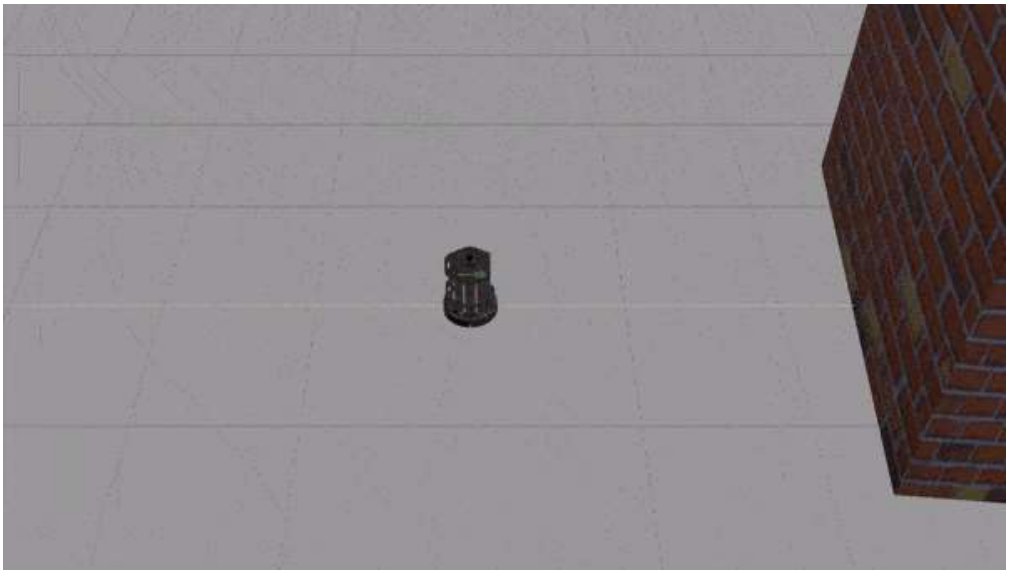
▶ Execute in Shell #2

In [ ]:
```
roslaunch publisher_example stop.launch
```

IMPORTANT: At the end of each example, you'll find some notes that will help you with each example. Please take a look at them!

- End of Example 1.1 -

- Expected Result for Example 1.1 -



- End of Expected Result -

- Notes for Example 1.1 -

Check the following Notes in order to complete the Example:

Note 1: Even after terminating the command with Ctrl+C, the robot will still keep moving. This is because the robot will keep listening to the last message that you published on the topic. You will understand what this means later.</i>

- End of Notes -

Throughout the course, you will have to do ROS programs and test them on the simulated robot.

Awesome, right? You'll probably have a lot of questions about how this whole process works... What you've actually done is to launch a Publisher, which is a ROS program that writes a message into a topic, in this case into the **/cmd_vel** topic. This topic is the one used to send velocity commands to the base of the robot. So, by sending a message through this topic, you've made the robot start moving.

But... why did the robot keep moving, even after I stopped the program? Well, that's because the topic keeps listening to the last message that was published, which told it to move the robot. So, in order to stop it, you've just launched another Publisher, which, in this case, was publishing a message that told the robot to stop.

But wait, remember that this is just a Course Preview. You'll be able to learn more about topics in the **Topics Unit (Chapters 3 and 4)**. Just be patient!

0.1.3 Third, you'll need to know about Services

So now, let's move on. You've seen that topics are a very important part of ROS, and that they let you communicate with your robot. But is this the only way?

As you can imagine, the answer is NO. Of course! ROS also provides **services**. Services allow you to code a specific functionaility for your robot, and then provide for anyone to call it. For instance, you could create a service that makes your robot move for a specific period of time, and then stop.

Services are a little bit more complex than topics since they are structured in two parts. On one side, you have the Service Server, which provides the functionality to anyone who wants to use it (call it). On the other side, you have the Service Client, which is the one who calls/requests the service functionality.

Let's see an example of how you could call a previously created service, which makes the Kobuki robot move in a circle for 4 seconds.

- Example 1.2 -

IMPORTANT: Make sure to stop all the previous programs running in your WebShells (by pressing Ctrl+C) before starting this example. Also, make sure that the Kobuki robot is not moving.

Execute the following command in WebShell number #1 in order to start the service.

▶ Execute in Shell #1

In [ ]:
```
roslaunch service_demo service_launch.launch
```

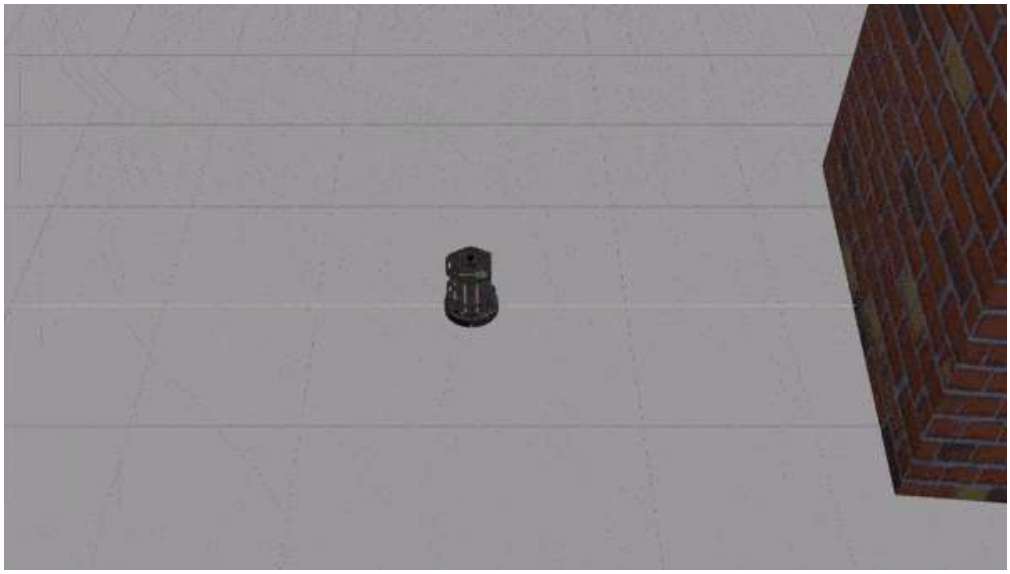
Execute the following command in WebShell number #2 in order to call the service.

▶ Execute in Shell #2

In [ ]:
```
rosservice call /service_demo "{}"
```

- End of Example 1.2 -

- Expected Result for Example 1.2 -



- End of Expected Result -

- Notes for Example 1.2 -

Check the following notes in order to complete the example:

Note 1: The service must be up and running before you can call it. So make sure that you have launched the service before calling it.

Note 2: Bear in mind that your robot will start moving from the point you stopped it in the previous example. So, it may not coincide with the gif shown.

- End of Notes -

I bet you're starting to get thrilled about this whole thing, right? But I'm sure you still have more questions.

In the above example, you've done two things. First, you **started** the service in order to make it available for anyone who wants to call it. Then, you executed a command that **called** that service. The purpose of this service is to move the Kobuki robot in a circle for 4 seconds, and then stop it. So, when you called the service, the robot started performing this movement.

But remember, this is just the Preview Unit. In the **Services Units (Chapters 5 and 6),** you will learn a lot more about services!

0.1.4 Fourth, you'll need to know about Actions

Thought you were done? Not at all! You still need a couple of things in order to be able to work with ROS. For now, you've already had a glimpse of **topics** and **services**, and you've seen how they allow you to communicate with your robot.

But, that's not all. ROS also provides **actions**. Actions are similar to services, in the sense that they also allow you to code a functionality for your robot, and then provide it so that anyone can call it. The main difference between actions and services is that when you call a service, the robot has to wait until the service has ended before doing something else. On the other hand, when you call an action, your robot can still keep doing something else while performing the action.

There are other differences, such as an action allowing you to provide feedback while the action is being performed. But you'll see all of that during the course. For now, let's just try a quick example to work with an action.

- Example 1.3 -

IMPORTANT: Make sure to stop all of the programs running in your WebShells (by pressing Ctrl+C) before starting with this example.

Execute the following command in WebShell number #1 in order to start the action.

▶ Execute in Shell #1

In [ ]:
```
roslaunch action_demo action_launch.launch
```

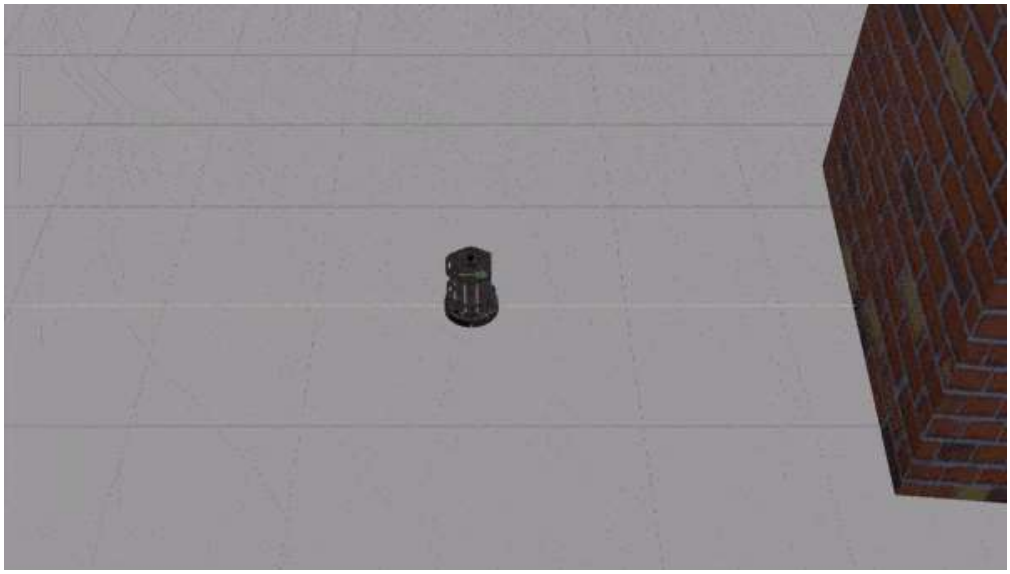
Execute the following command in WebShell number #2 in order to "call" the action.

▶ Execute in Shell #2

In [ ]:
```
roslaunch action_demo_client client_launch.launch
```

- End of Example 1.3 -

- Expected Result for Example 1.3 -



- End of Expected Result -

- Notes for Example 1.3 -

Check the following notes in order to complete the example:

Note 1: Make sure you've started the Action Server by executing the first command. Otherwise, you won't be able to call it.

Note 2: Bear in mind that your robot will start moving from the point you stopped it at in the previous example. So, it may not coincide with the gif shown.

- End of Notes -

I'm sure that, at this point, you're starting to get a little bit confused with what you're doing. Don't worry, that's all right!

Bascially, you've done two things in the previous example. First, you started the Action Server by issuing the first command. Then, you "called" the Action by issuing the second command. You may be thinking that this is the same as the previous example with services, but it is not. There are some important differences between them, though you may not be able to see it yet.

But, you don't have to worry about that now. As I've told you before, this is just a Course Preview, so you will more fully understand Actions when you go through the **Actions Unit (Chapters 7 and 8)**.

0.1.5 Finally, you'll need to know how to use the Debugging Tools

If you are going to start working with robots, you need to know this one thing: **you will have to deal with errors**.

Fortunately, ROS provides lots of tools that will help you detect what's going on. In the following example, you will be introduced to what is likely to be the most important tool of all of them: **RViz**. Follow the below example to have a quick look at this amazing tool!

- Example 1.4 -

IMPORTANT: Make sure to stop all of the programs running in your WebShells (by pressing Ctrl+C) before starting with this example.

Execute the following command in WebShell number #1 in order to start RViz.

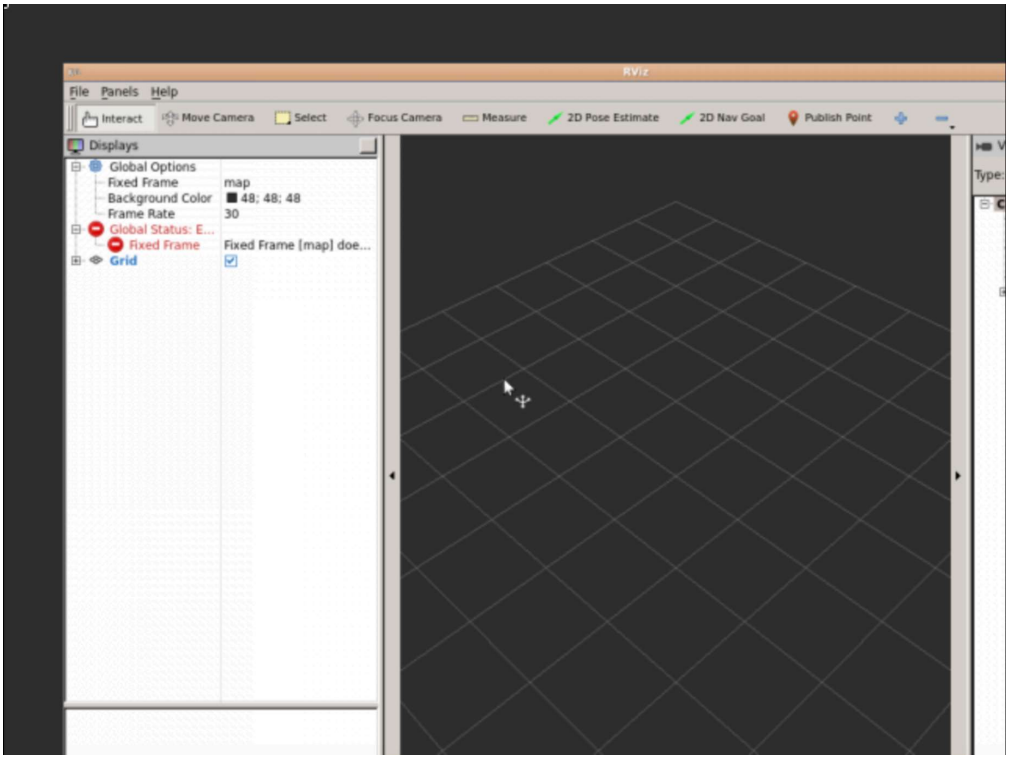
▶ Execute in Shell #1

In [ ]:
```
rosrun rviz rviz
```

Now, hit the icon with a screen on it, located in the top-right corner of the IDE window,

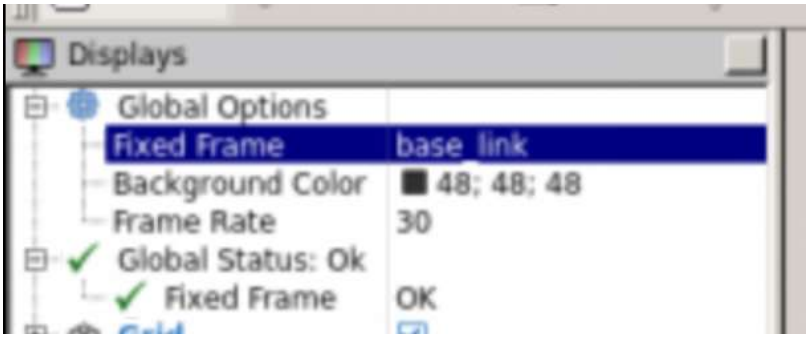
in order to open the Graphic Interface.

This will open the Graphic Interface in a new tab. You should see RViz here.

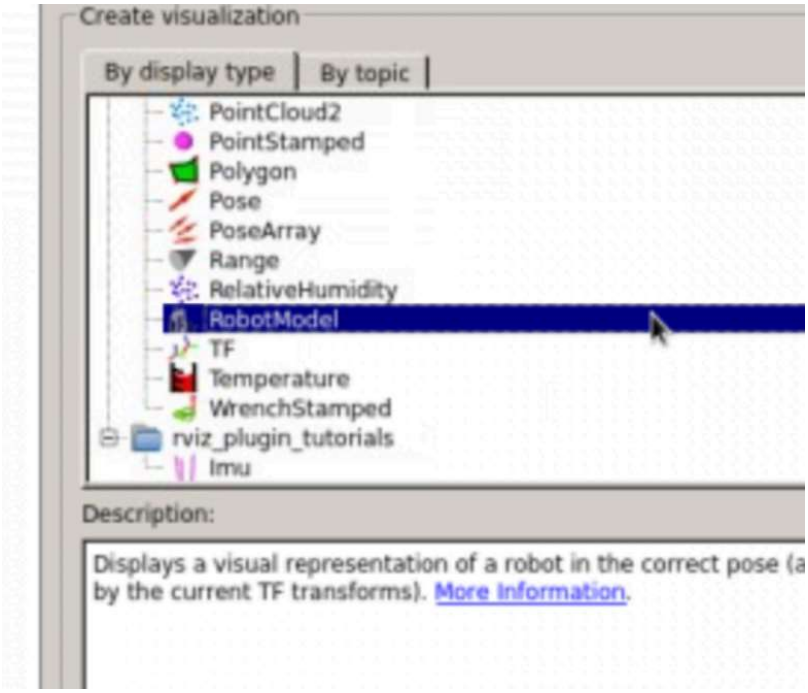


Now, follow the next steps:

a) Change the Fixed Frame for the "base_link."

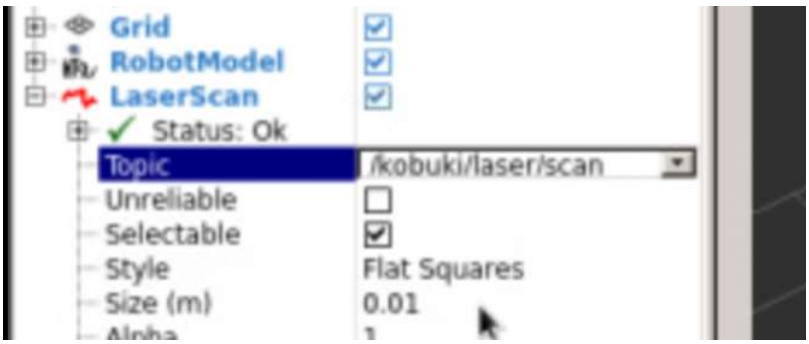


b) Click the "Add" button below the Displays section. Select the RobotModel option.



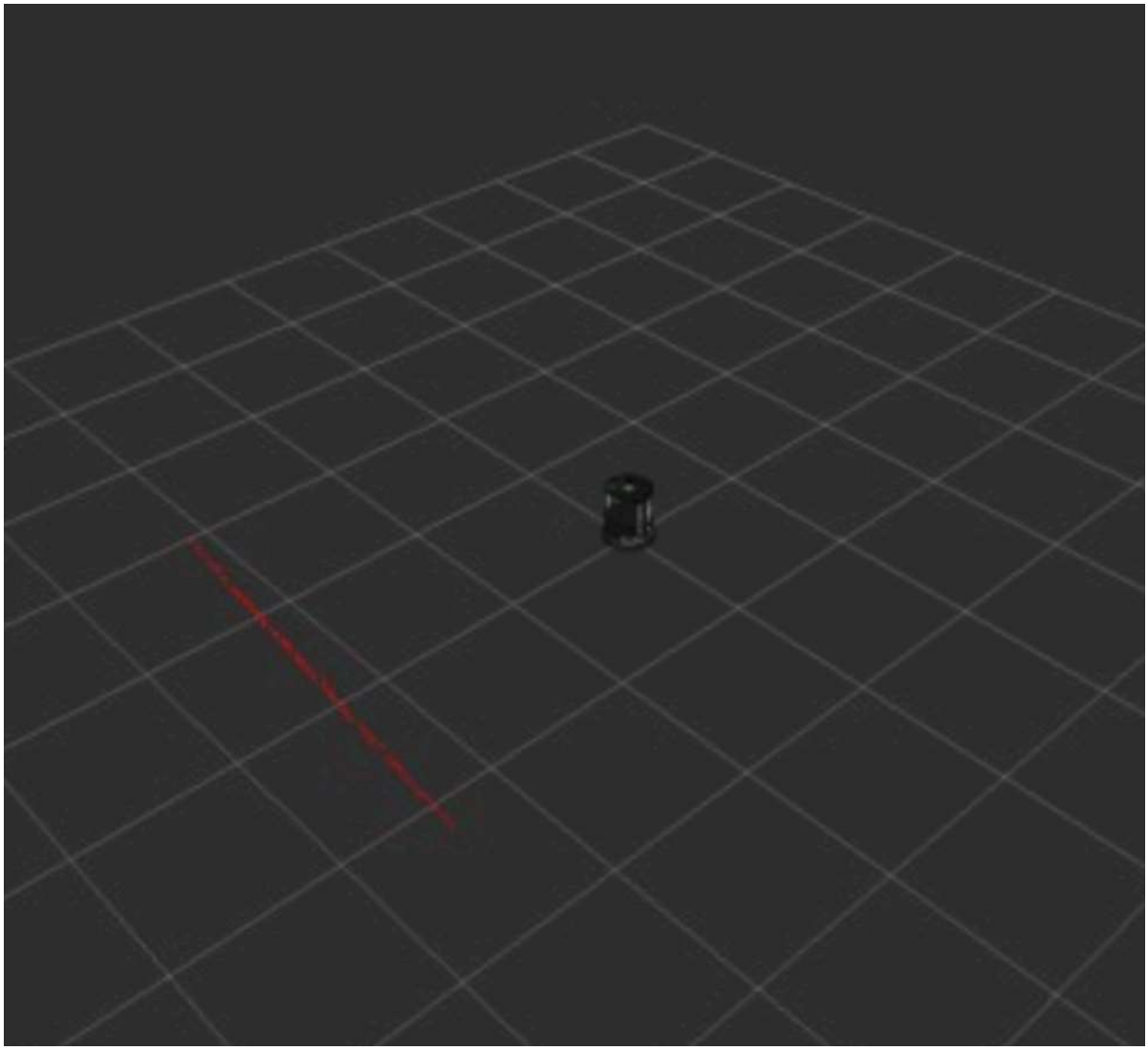
c) Now, repeat the process and select the LaserScan option.

d) Select the "/kobuki/laser/scan" topic from the LaserScan options.



- End of Example 1.4 -

- Expected Result for Example 1.4 -



- End of Expected Result -

- Notes for Example 1.4 -

Check the following notes in order to complete the Example:

Note 1: You can maximize the RViz window by double-clicking on its top bar.

- End of Notes -

At this point, I'm sure you have a lot of questions and doubts... but stay calm!

What you are seeing in RViz right now is your Kobuki robot. That's obvious, right? But what is that red line that appears in front of Kobuki? Do you know?

Well, that's the laser readings that the Kobuki robot is getting. So, as you may have already guessed, that red line represents the laser beams that are detecting that there's a wall in front of the robot.

Amazing, right? But this is still just a very small part of what you can achieve with RViz. You'll have to check the **Debugging Tools Unit (Chapter 9) to see more of RViz!**

So, what do you say? Want to really learn how all of this works? Want to become a Robotics master? Then let's get started!