ROS Basics in 5 days

Unit 1   ROS Deconstruction

ROS in 5 days, I cannot believe it!

You have probably heard of ROS and how long it takes to learn to all students. It is very likely that you are thinking that learning ROS in 5 days is not possible... so let me tell you the news:

It is possible to learn ROS fast if you have the proper method!

How is it possible?

We have created a learning method that allows you to get most of ROS in the minimum amount of time. Our method has 4 parts:

1. **DECONSTRUCTION**: we have identified the important parts of ROS that you must master in order to understand 80% of ROS programs. You will concentrate on learning these parts very deep.
2. **REMOVING**: we have removed many things that are not needed and just add noise to your learning.
3. **LEARNING**: we guide you step by step in a progressive manner through all those important parts, starting always from a robot that does things.
4. **PRACTICING**: we make you practice a lot on every step, always on a robot using our simulated robots.

- Want to learn more about this method? Read this book: The First 20 Hours: How to Learn Anything . . . Fast! (https://www.amazon.com/gp/product/1591846943/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=1591846943&linkCode=as2&tag=theconsim-20&linkId=88feac086be2a46c28909cb9909a6860)
- Want to have scientific evidence of this method? Read this paper: Cognitive Skill Acquisition (https://pdfs.semanticscholar.org/6ce5/af5973ecec36eb9c699e7cae739743c8b4b5.pdf).

So... at this point, the main question is: which are the most important parts of ROS that you need to learn in order to be able to program any robot with ROS?

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

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.

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 important! 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 **Terminal number #1** in order to start moving the Kobuki robot.

Terminals are like Linux terminals, but on the web. They are located, by default, at the bottom right section of the screen. You can type Linux commands there.

▶ Execute in terminal #1

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

Whenever you want to stop moving the robot, and in order to stop the program, just press Ctrl + C in the Terminal.

NOTE: You will notice that, 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 **Terminal #2** and execute the following command in order to stop the Kobuki robot.

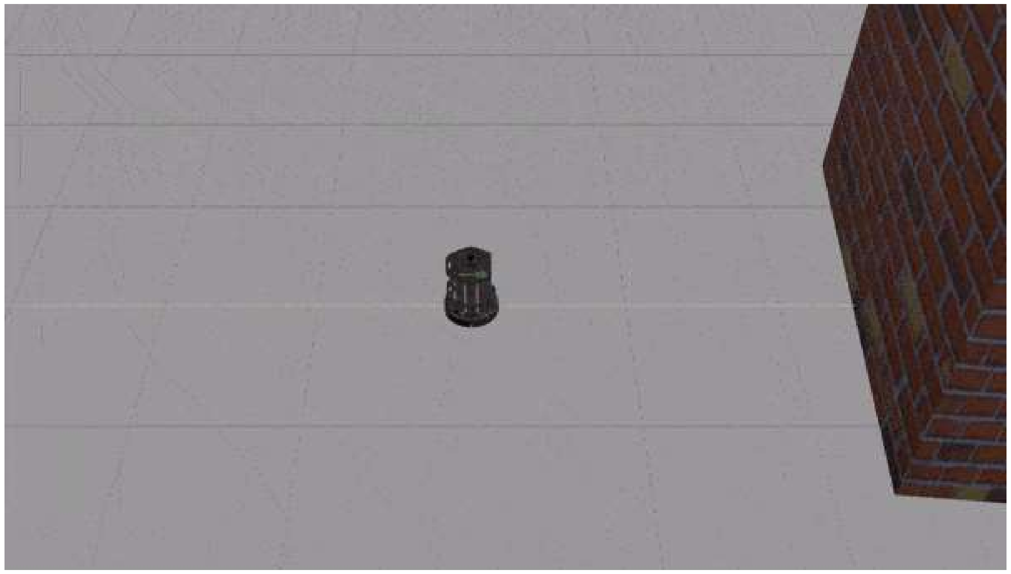
▶ Execute in terminal #2

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

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

- End of Example 1.1 -

- Expected Behavior for Example 1.1 -



- End of Expected Behavior -

- 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.

- 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 robot keeps listening to the last message that was published into the topic, 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!

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 functionality 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 Terminals (by pressing Ctrl+C) before starting this example. Also, make sure that the Kobuki robot is not moving.

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

▶ **Execute in terminal #1**

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

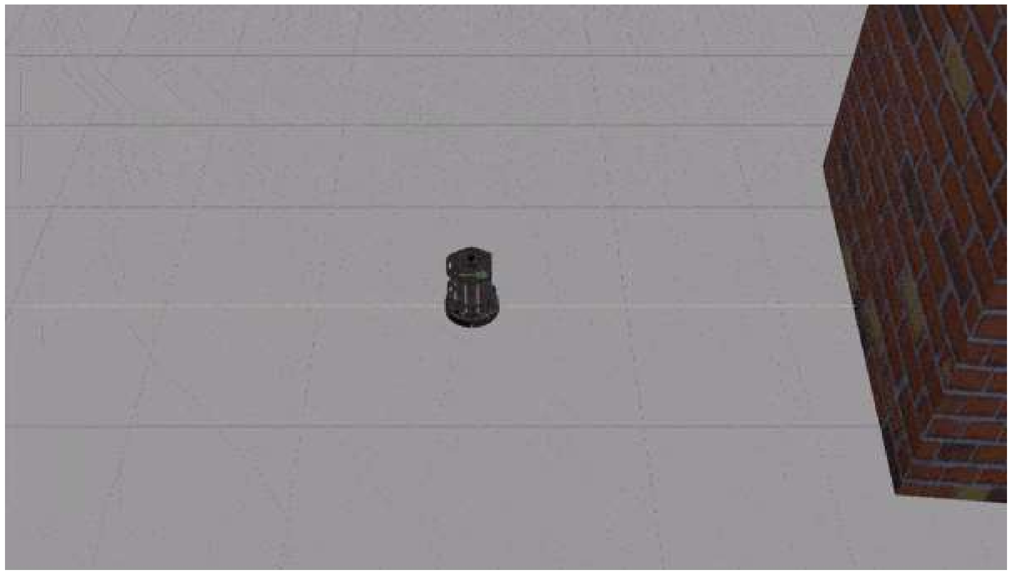
Execute the following command in Terminal number #2 in order to call the service.

▶ **Execute in terminal #2**

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

- End of Example 1.2 -

- Expected Behavior for Example 1.2 -



- End of Expected Behavior -

- 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 Unit (Chapters 5 and 6),** you will learn a lot more about services!

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 Terminals (by pressing Ctrl+C) before starting with this example.

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

▶ Execute in terminal #1

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

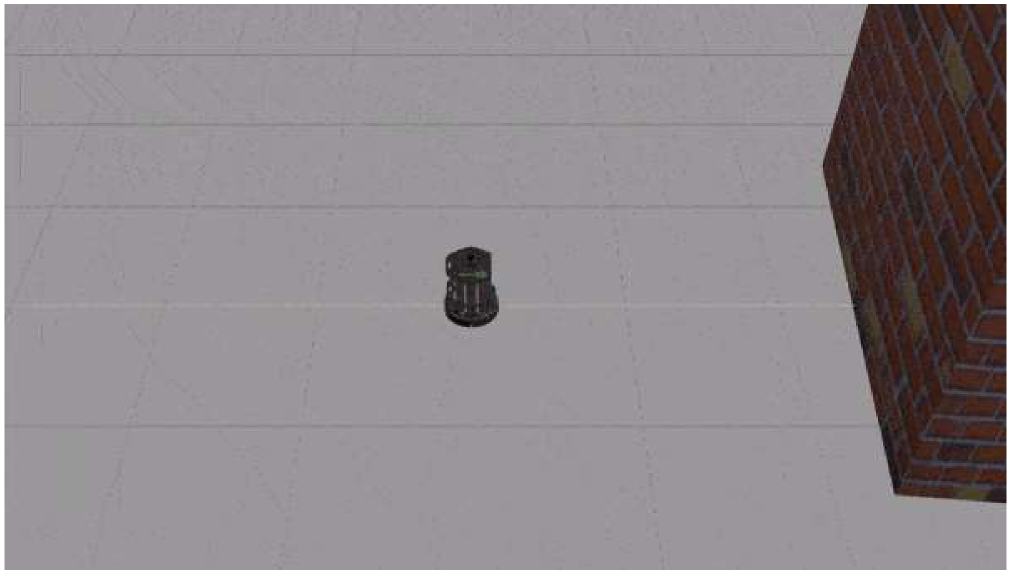
Execute the following command in Terminal number #2 in order to "call" the action.

▶ Execute in terminal #2

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

- End of Example 1.3 -

- Expected Behavior for Example 1.3 -



- End of Expected Behavior -

- 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!

Basically, you've done two things in the previous example. First, you **started the Action Server** by issuing the first command. Then, you **called that 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 get a better understanding of Actions when you go through the **Actions Unit (Chapters 7 and 8)**.

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 to 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 Terminals (by pressing Ctrl+C) before starting with this example.

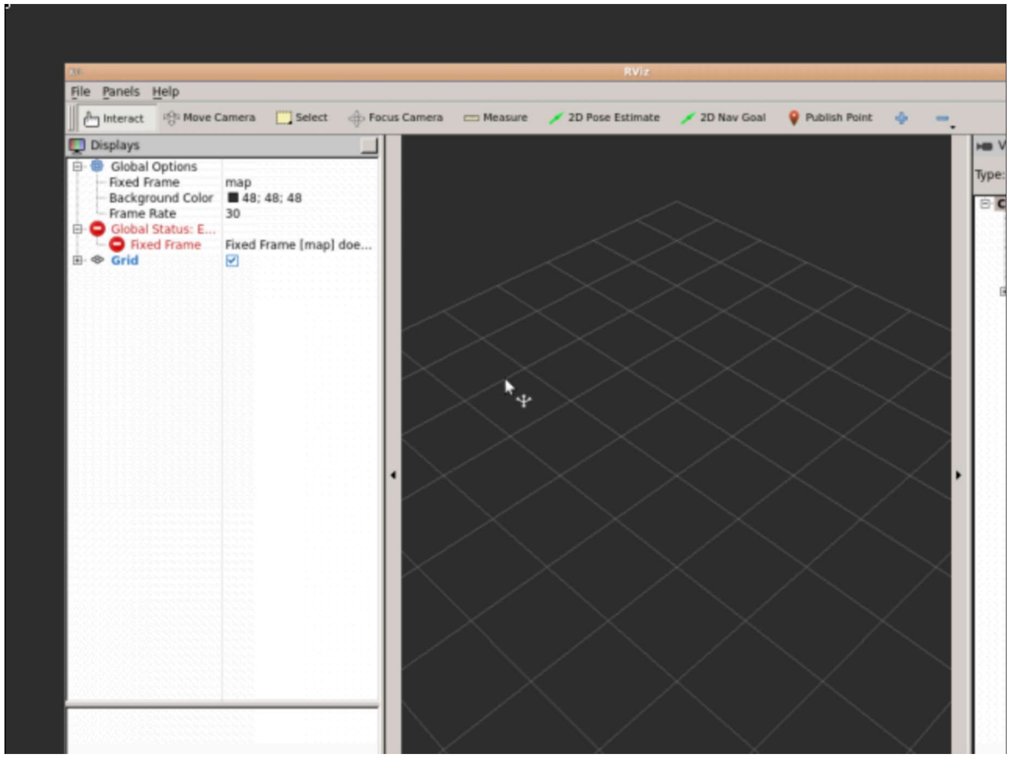
Execute the following command in Terminal number #1 in order to start RViz.

▶ Execute in terminal #1

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

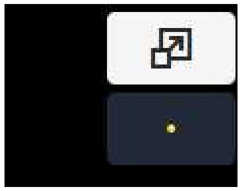
You should see a black canvas appear on top of everything else on the screen. Wait a couple of seconds until Rviz has loaded.

NOTE: Tap the minimize icon to hide the graphical tools window to the taskbar. Hit the icon with a screen on it to get it back.



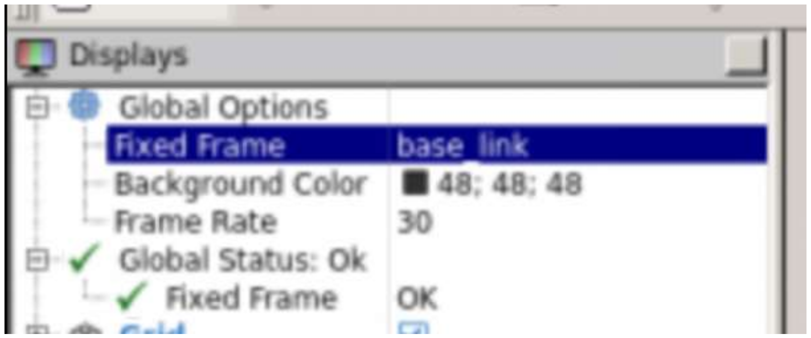
Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
Global Status: E...	
Fixed Frame	Fixed Frame [map] doe...
Grid	✓

NOTE: You can maximize the RViz window by double-clicking on its top bar. If you can't access the top bar, just select the RViz window and then press the icon to the right in order to readjust the current window. This is the icon:</b>.



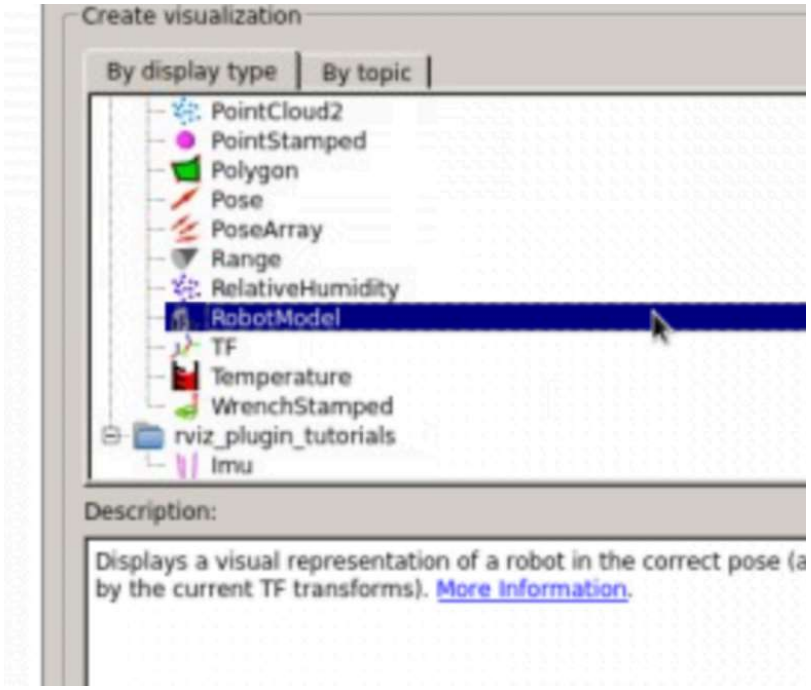
Now, follow the next steps:

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



Global Options
Fixed Frame base_link
Background Color ■ 48; 48; 48
Frame Rate 30
Global Status: Ok
Fixed Frame OK

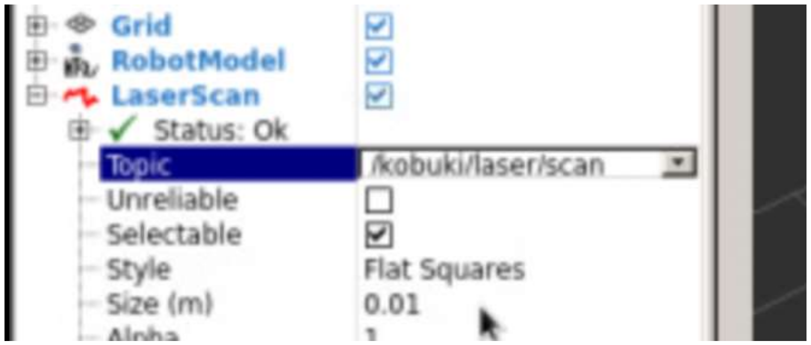
b) Click the "Add" button below the Displays section. Select the RobotModel option.



Displays a visual representation of a robot in the correct pose (a
by the current TF transforms). More Information.

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

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



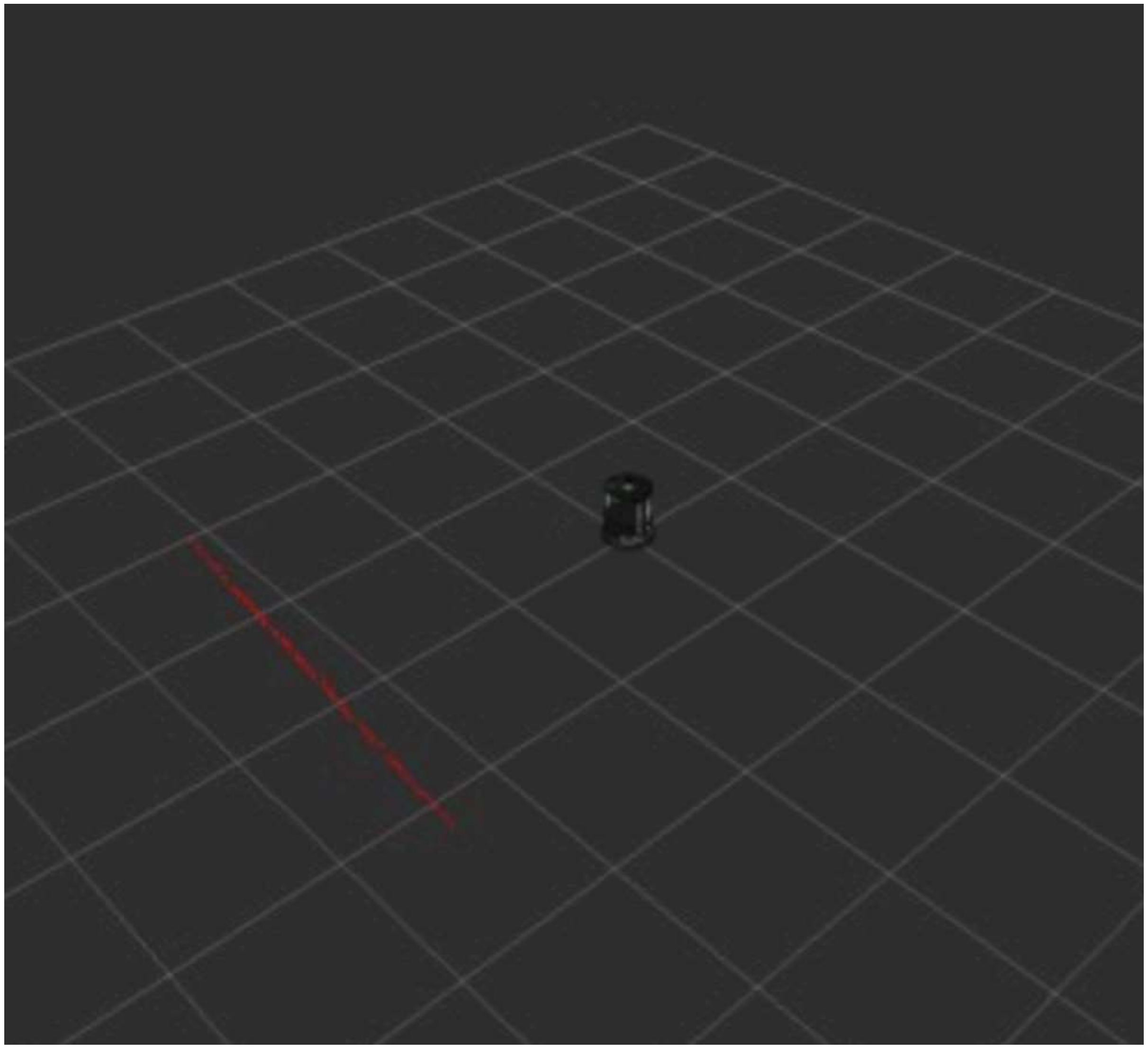
Grid ✓
RobotModel ✓
LaserScan ✓
Status: Ok
Topic /kobuki/laser/scan
Unreliable ☐
Selectable ✓
Style Flat Squares
Size (m) 0.01
Alpha 1

e) If you want to, you can also move the robot, for instance, using the programs you launched on Example 1.1

- End of Example 1.4 -

- Expected Behavior for Example 1.4 -

NOTE: In order to visualize the wall through the laser beams, the robot must be facing the wall. Otherwise, obviously, it won't detect anything.



- End of Expected Behavior -

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 wall in front of the robot, which is being detected by the laser beams.

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 Developer**?

1.2 Main Goal

1. The objective of this course is to give you the basic tools and knowledge to be able to understand and create any basic ROS related project. You will be able to **move robots, read their sensor data, make the robots perform intelligent tasks, see visual representations of complex data such as pointclouds and debug errors in the programs.**
2. The course will allow you to **understand packages that others have created**. So you can take ROS code made by others and understand what is happening and how to modify it for your own purposes.
3. This course can serve as an introduction to be able to understand the ROS documentation of complex ROS packages for object recognition, text to speech, navigation and all the other areas where has ROS developed code.

1.3 Learning ROS: attack in two ways

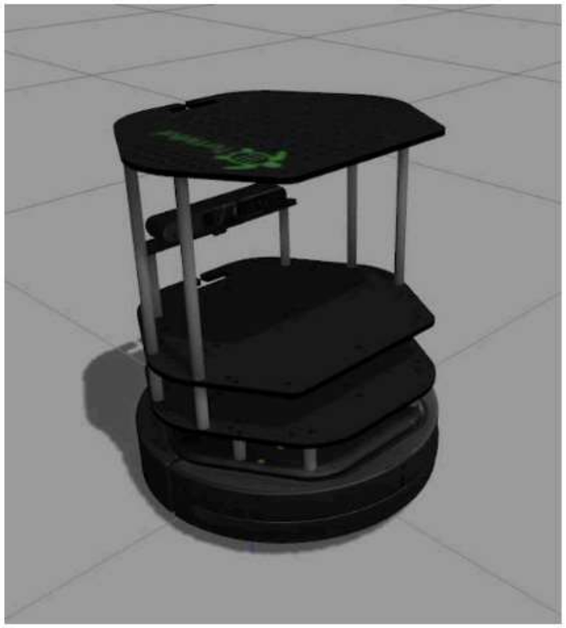
The course provides teaching lessons in **two different ways**:

1. **Learn ROS programming several robots**: This part consists of units each of which teaches you some topic of ROS. But you will learn creating and executing code while using different robots for it (the robots above).**Theory through hands on experience**.
2. **Apply what you learned to a Robot Project**: Here you will apply what you have learned in the previous units by attacking a **full project controlling a real TurtleBot3 robot**. The objective is to make the TurtleBot3 follow a wall using laser data.

1.3.1 Learning ROS by programming several robots

Along this course you are going to program several robots of several types. Hence, you will be applying the same concepts over and over again, with different configurations and styles of robots. Applying the same concepts on different robots will make the concepts stick into your head.

The following robots will be used along the course:

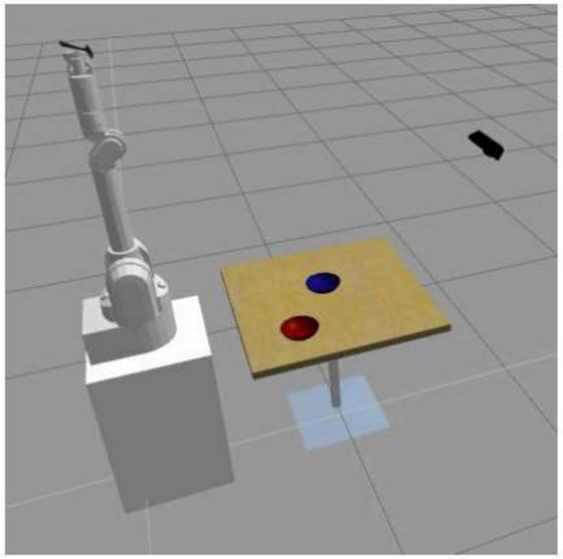


Parrot Drone

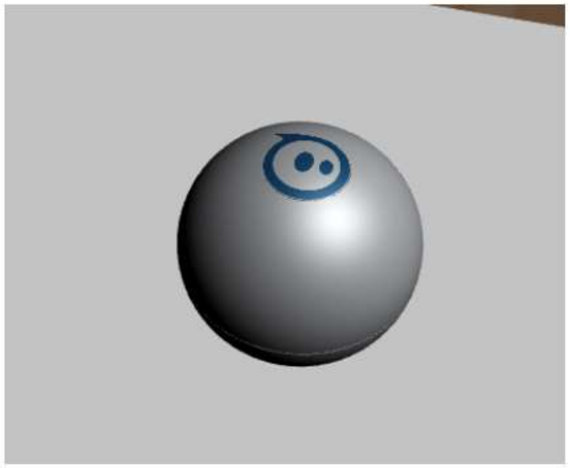
Kobuki

BB-8

Husky



Wam Arm

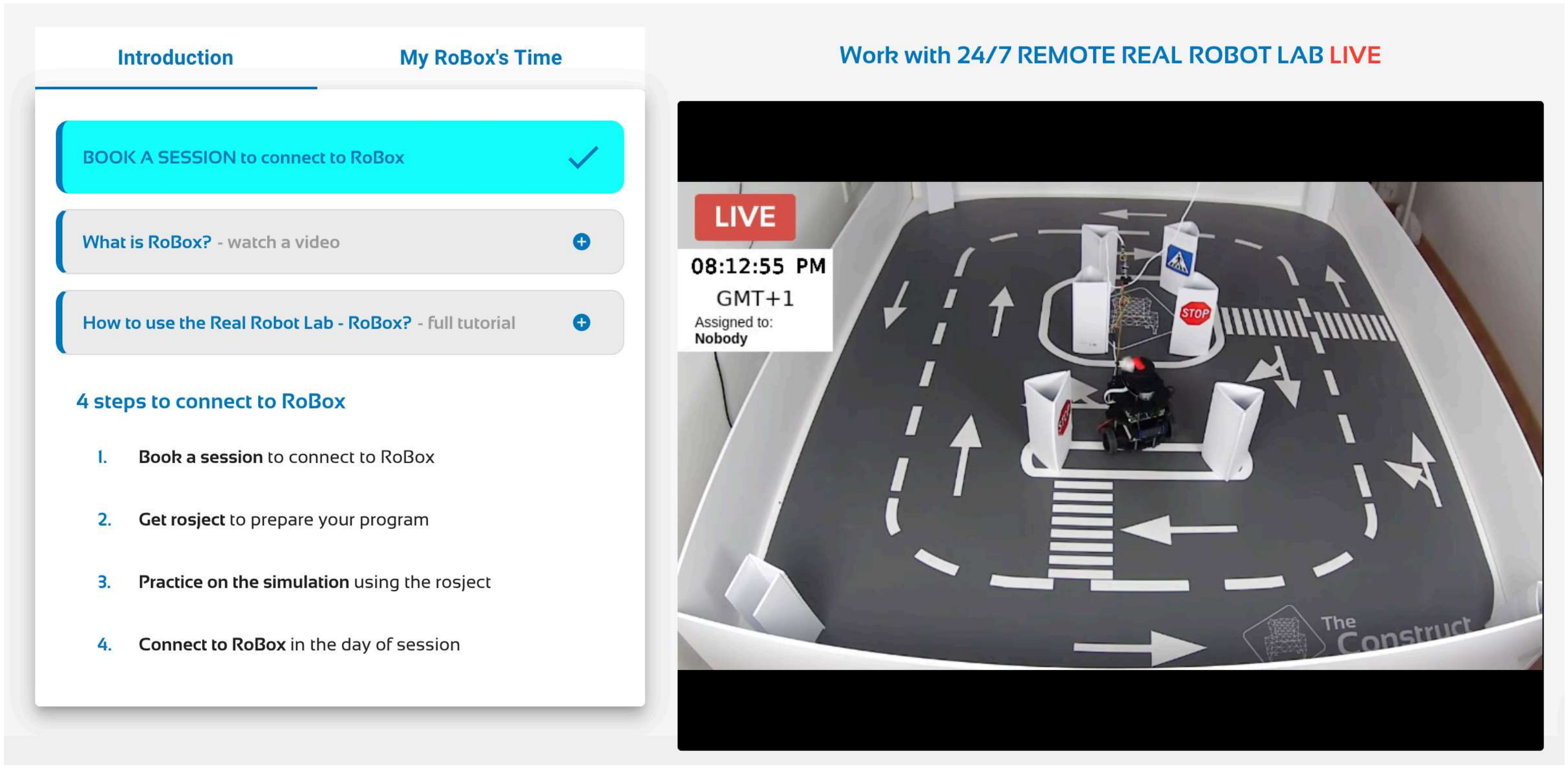


Sphero

1.3.2 Apply what you learnt to a Real Robot Project

As you master the different concepts of ROS, you will have to apply everything you learn during the course in a complete robot project. And I'm not talking about any kind of robot project, but a project based on a **real robot** which is located in our facilities in Barcelona.

For this purpose, we will be using the Real Robot Lab (https://app.theconstructsim.com/#/RealRobot) tool. This amazing tool will allow you to remotely control and run your ROS programs, from any place of the world, in a real robot.

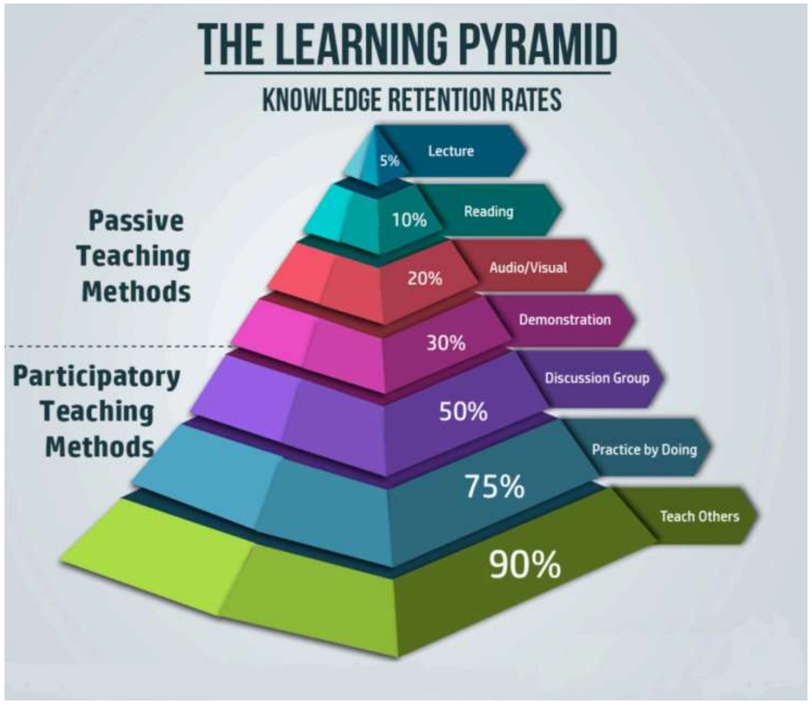

Introduction
My RoBox's Time
BOOK A SESSION to connect to RoBox
What is RoBox? - watch a video
How to use the Real Robot Lab - RoBox? - full tutorial
Work with 24/7 REMOTE REAL ROBOT LAB LIVE
LIVE
08:12:55 PM
GMT+1
Assigned to:
Nobody
STOP
The Construct

4 steps to connect to RoBox

1. **Book a session** to connect to RoBox
2. **Get rosject** to prepare your program
3. **Practice on the simulation** using the rosject
4. **Connect to RoBox** in the day of session

You will get introduced into the real robot project as you advance through the different units of the course.

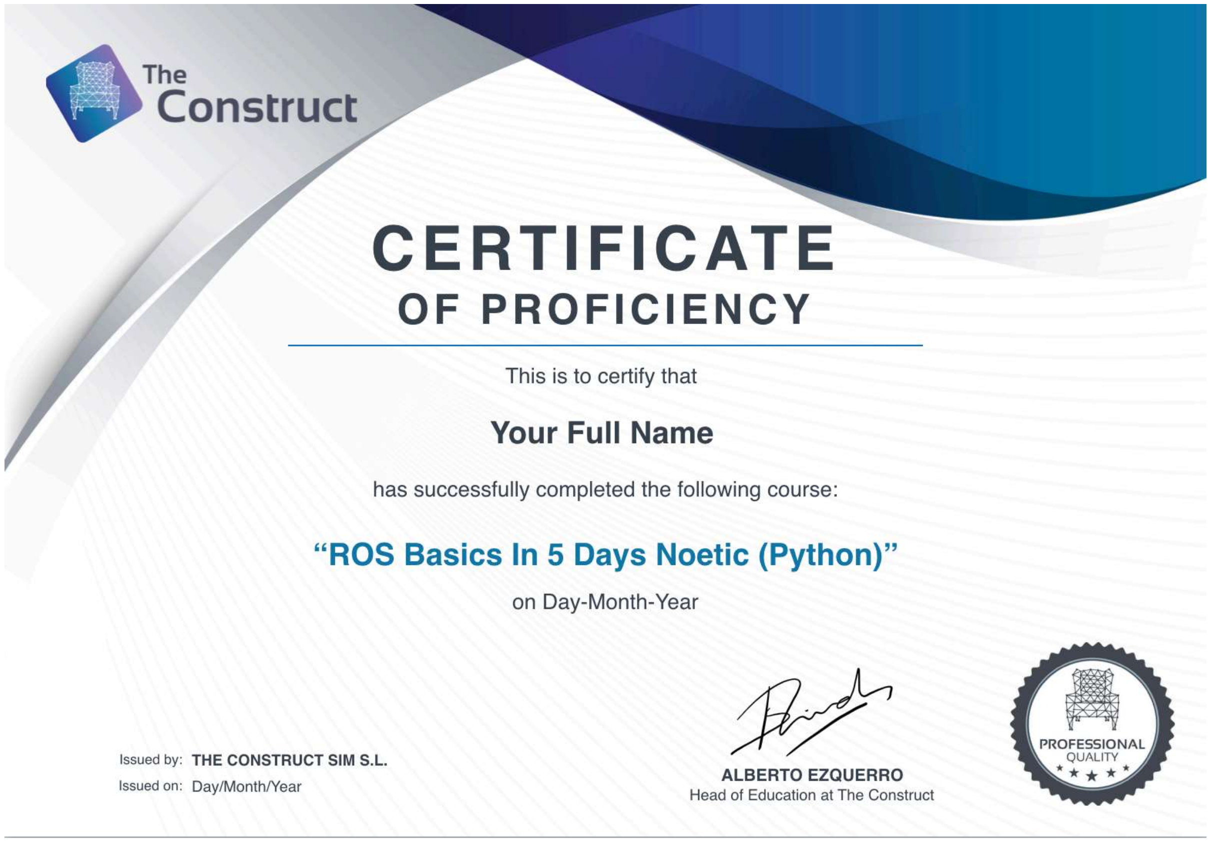
But that's not all! After you complete the real robot project, you will have the chance to do a **live presentation** of your project. And why do we want you to show and explain your project in a live session? Because at The Construct, we strongly believe that this is the most efficient and challenging way of learning and proving your knowledge on a subjet.


THE LEARNING PYRAMID
KNOWLEDGE RETENTION RATES
Passive Teaching Methods
Participatory Teaching Methods
5% Lecture
10% Reading
20% Audio/Visual
30% Demonstration
50% Discussion Group
75% Practice by Doing
90% Teach Others

1.4 Get a certificate

Upon completion of the course, you will be get the chance to earn a certificate proving your knowledge on ROS Basics. In order to earn the certificate, you will have to successfully complete the following tasks:

- Pass all the course Quizzes.
- Successfully complete the live presentation of your project.



The Construct

CERTIFICATE
OF PROFICIENCY

This is to certify that

Your Full Name

has successfully completed the following course:

"ROS Basics In 5 Days Noetic (Python)"

on Day-Month-Year

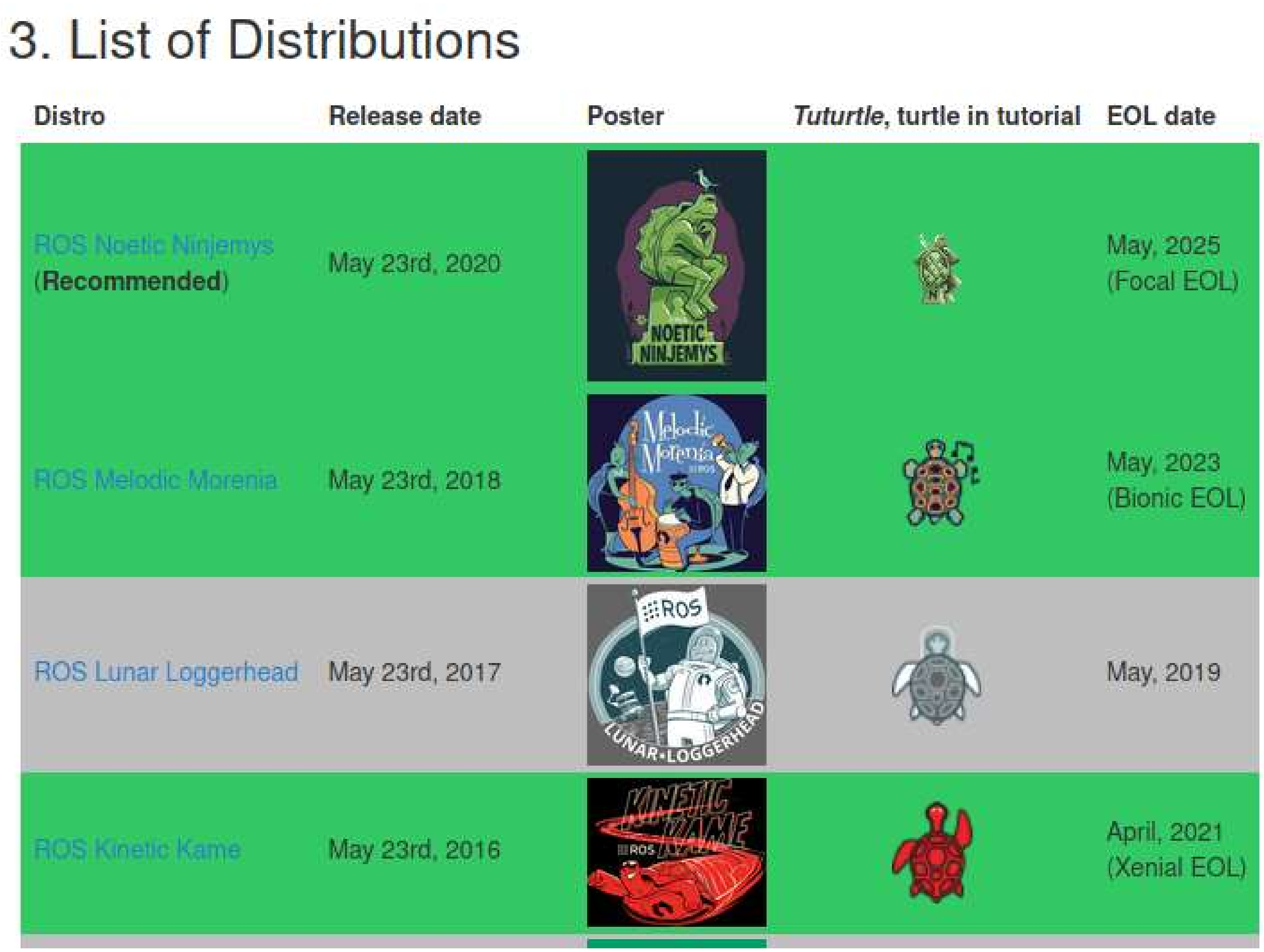
Issued by: THE CONSTRUCT SIM S.L.
Issued on: Day/Month/Year

[illegible]
ALBERTO EZQUERRO
Head of Education at The Construct

PROFESSIONAL QUALITY

1.5 ROS Distribution

Finally, it is important to mention that this course uses the **ROS Noetic** distribution. And why have we decided to use this distribution? Well, have a look at the below data, exctracted from the ROS Wiki site (http://wiki.ros.org/Distributions (http://wiki.ros.org/Distributions)):



3. List of Distributions

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020	NOETIC NINJEMYS		May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018	Melodic Morenia		May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017	ROS LUNAR·LOGGERHEAD		May, 2019
ROS Kinetic Kame	May 23rd, 2016	KINETIC KAME ROS		April, 2021 (Xenial EOL)

As you can see, **ROS Noetic** is the latest *supported* and *recommended* distribution released. Therefore, it is also the recommended distribution to use.

1.6 Special Thanks

- This course wouldn't have been possible without the knowledge and work of the ROS Community (http://www.ros.org/), OSRF (https://www.osrfoundation.org/), and Gazebo Team (http://gazebosim.org/).

ROS

GAZEBO