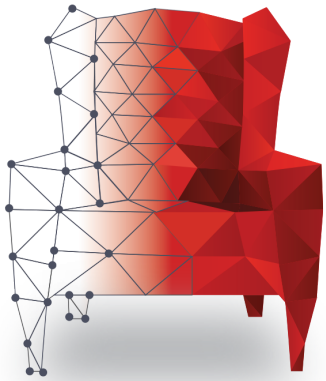


The Construct

Learn and develop for robots with ROS

PRESENTS

ROS Developers Live Class n82

How to create a course for The Robot Ignite Academy

In this class, you will learn how to create a new course for The Robot Ignite Academy

Interested in make a living teaching ROS and robotics subjects? In this Live Class we teach you how you can create courses for teaching specific subjects of ROS and robotics and how to submit them to Robot Ignite Academy to get 1.500€ per course.

From how to create it, it must contain minimally, how to assemble the different units, how to use the simulations, and much more. Collaborate with us in this incredible community, and help ROS Developers to continue their arduous learning path in a more enjoyable and unique way.

If you are interested in becoming a **Robotics Developer** you will need to know how to represent the robot structure in the proper way so you can program it with ROS.

(To know more about becoming a robotics developer, read this guide about [How To Become a Robotics Developer (http://www.theconstructsim.com/become-robotics-developer/)](http://www.theconstructsim.com/become-robotics-developer/))

This rosject has been created by **Christian Chavez** and **Ricardo Tellez** from **The Construct**. You can use this rosject freely as long as you keep this notice.

REQUIREMENTS :

- **Basics of Linux**. If you don't have that knowledge, check this FREE online course (https://www.robotigniteacademy.com/en/course/linux-robotics/details/)

- **Ros Basics**. If you don't have that knowledge, check this online course (https://www.robotigniteacademy.com/en/course/ros-in-5-days/details/)

- ... That's it! Let's go!

In this class, we'll learn:

- How to create a course for the Robot Ignite Academy

How to use this ROSject

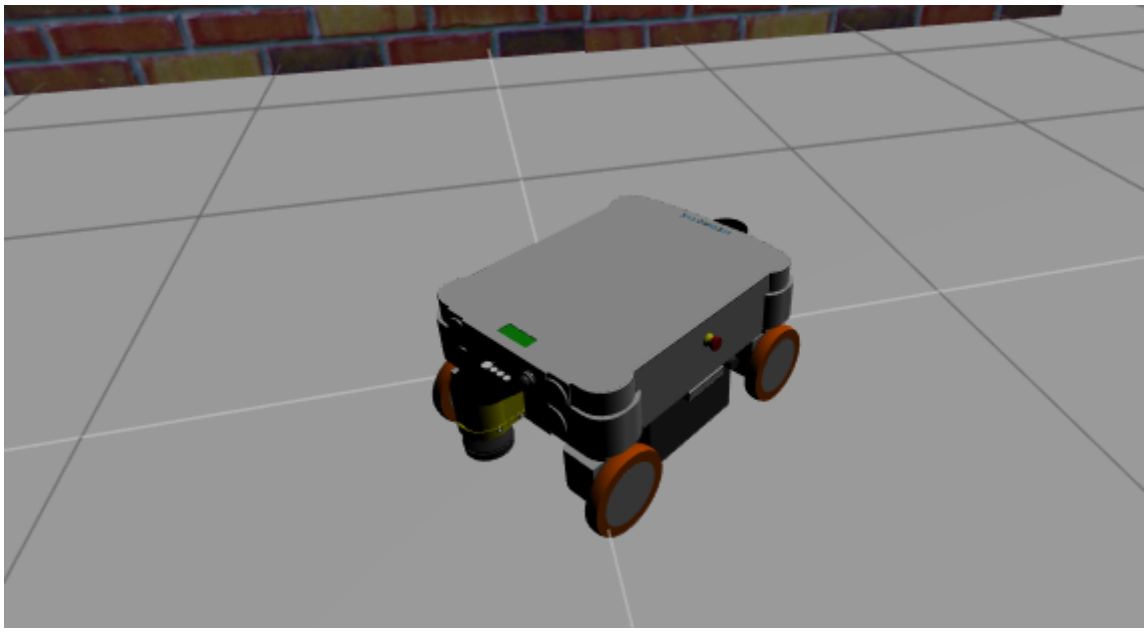
A **ROSject** (http://rosjects.com) is a **ROS project** packaged in such a way that all the material it contains (**ROS code, Gazebo simulations and Notebooks**) can be shared with any body **using only a web link**. That is what we did with all the attendants to the Live Class, we shared this ROSject with them (so they can have access to all the ROS material they contain).

Check this webinar to learn more about ROSjects and how to create your own ROSjects.

You will need to have a free account at the ROS Development Studio (http://rosds.online) (ROSDS). Get the account and then follow the indications below.

Robot for today's Live Class

Today you're going to use the MPO-500 by Neobotix



How to create a course for Robot Ignite Academy

0. What you need to create for the course?

In order to submit your course, you need to create 5 different notebooks teaching the material you want to teach. You may also include some code or datasets (those are optionals).

First notebook is just a demo of what the whole course is going to achieve The units 2, 3, 4 are the ones where you teach the subject divided into parts. Unit 5 is the project, where you make the student apply all the part you taught into a single full project.

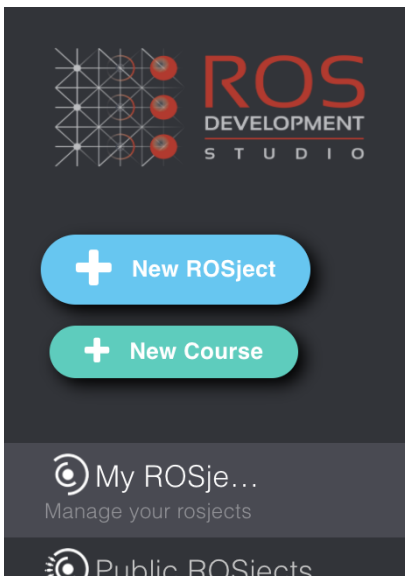
Before start creating the course, you need to decide a few things:

- Decide the subject you want to teach. Select one of the available subjects here
- Decide how you are going to divide your teaching in 3 notebooks
- Decide the simulation you are going to use

The following is a list of all the steps required to create the course

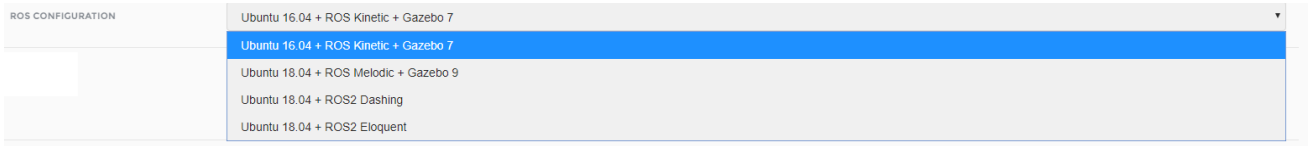
1. Create the course template

After login into ROSDS, press on the button *New Course*

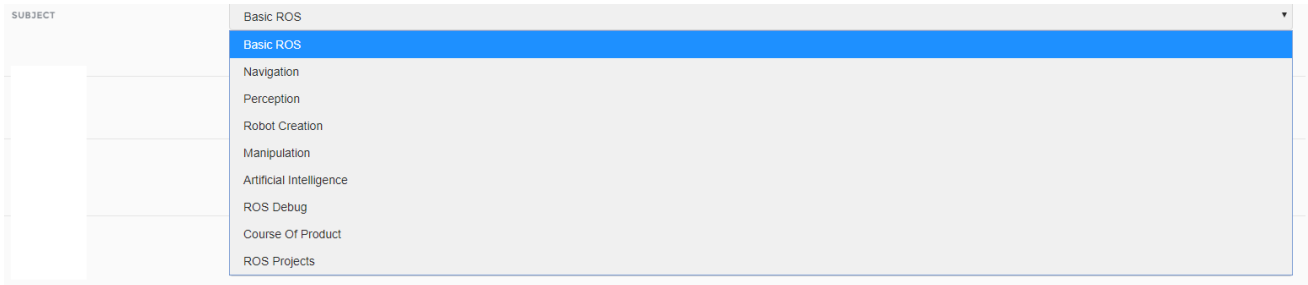


On the new page, fill all the details as follows:

- **ROS Configuration**: Select the distribution of ROS you want to create the course for. You can create courses for ROS Classic and ROS2. **At present only ROS Kinetic and ROS2 Eloquent are accepted**.

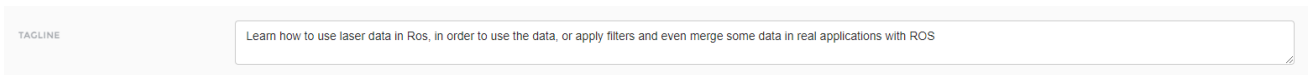


- **Subject**: select the main subject of your course.

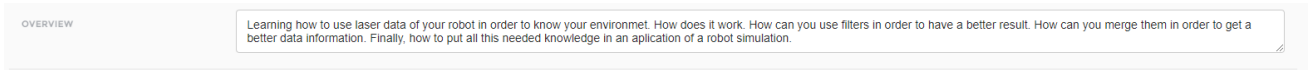


- **Title**: What is the title of your course?

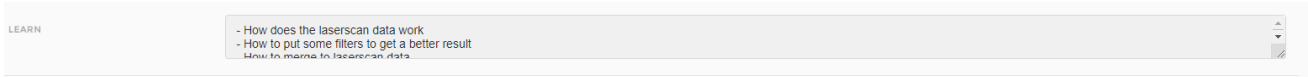
- **Tagline**: that is a short single sentence that describes your course.



- **Overview**: include a few sentences describing your course.



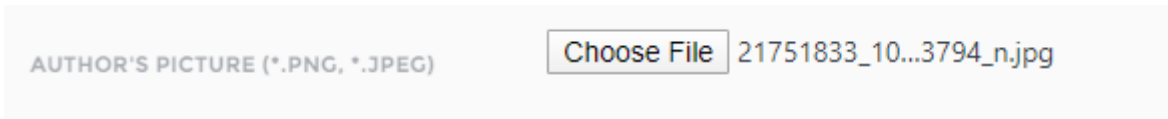
- **Learn**: Indicate a list of the main things that the student will learn with your course



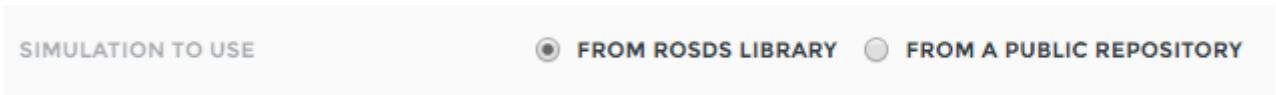
- **Author name**: your full name

- **Author description**: a brief description of you indicating why you are an expert in the subject you are teaching in this course

- **Author's picture**: upload a picture of yourself.



SELECT THE SIMULATION



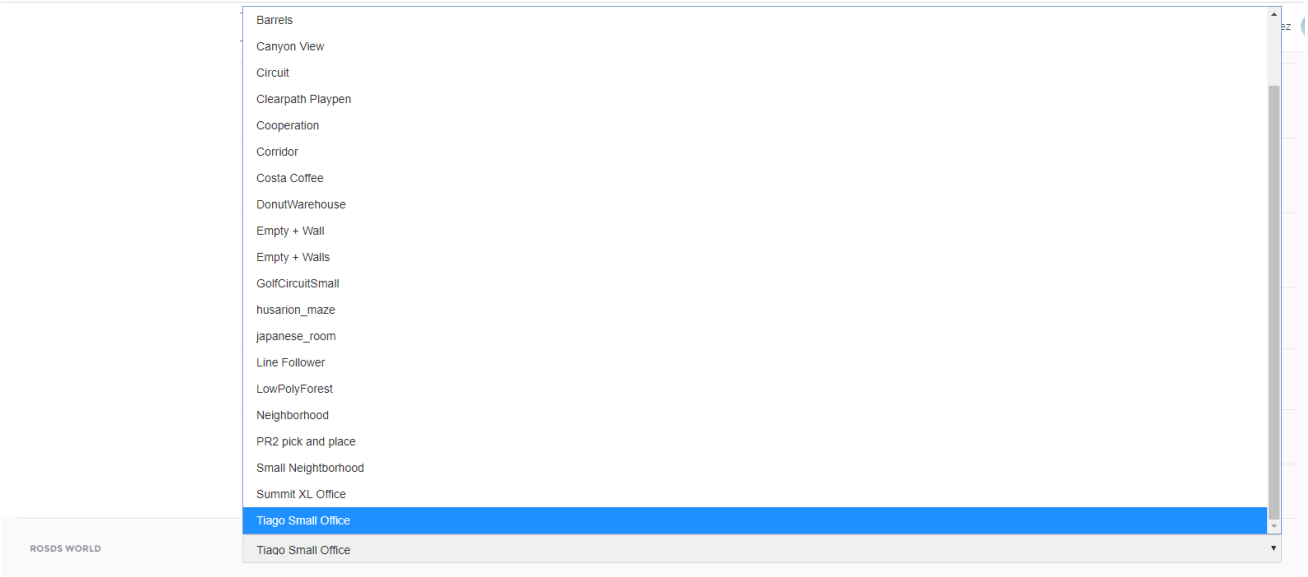
Next thing you have to indicate is the simulation the students will use to practice. You have two options:

- Use one of the simulations we already provide (ROSDS library)
- Use your own simulation

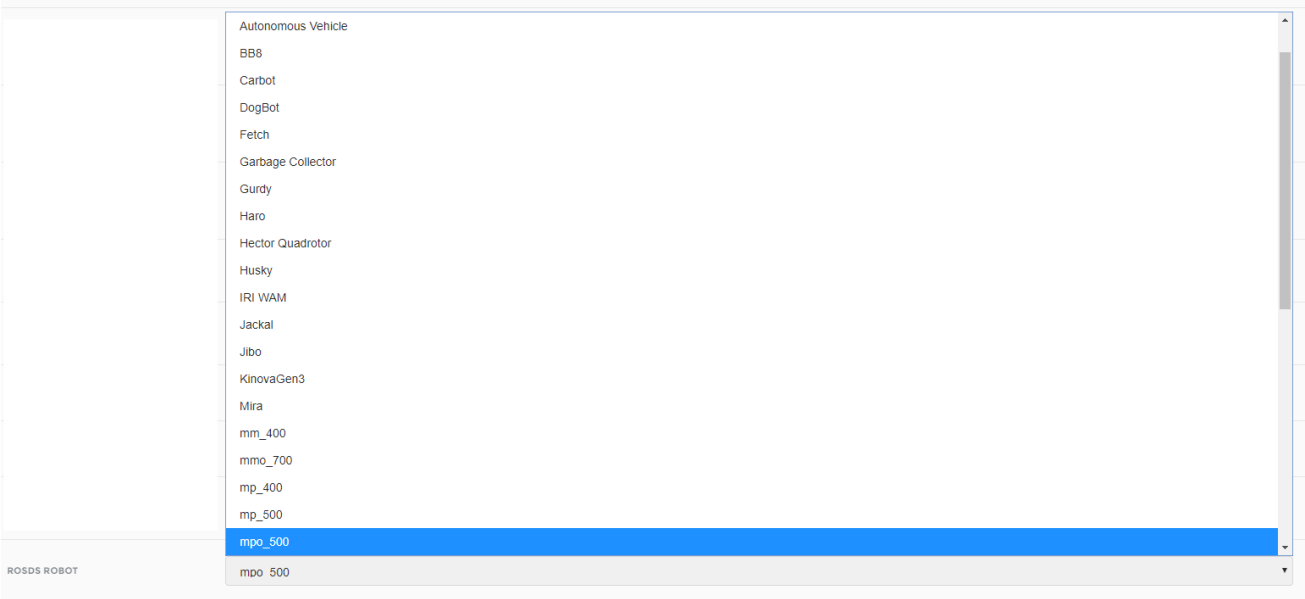
IMPORTANT: if this is your first time doing a course, we highly recommend you to use one of the ROSDS library.

If you select *ROSDS Library*

- Select the world in which the robot will show



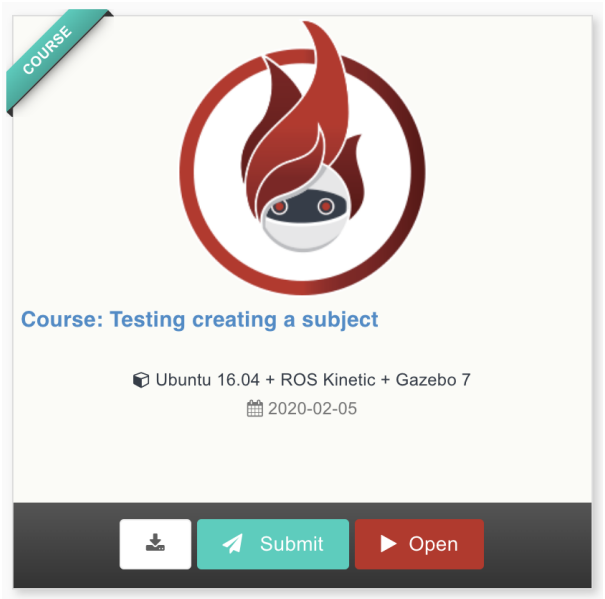
- Select the robot that will be used in that world for the practice



If you select *From a public repo*

- Indicate the repo address to clone with git
- From that repo, indicate the package that launches the simulation
- From that repo, indicate the launch file that launches the simulation

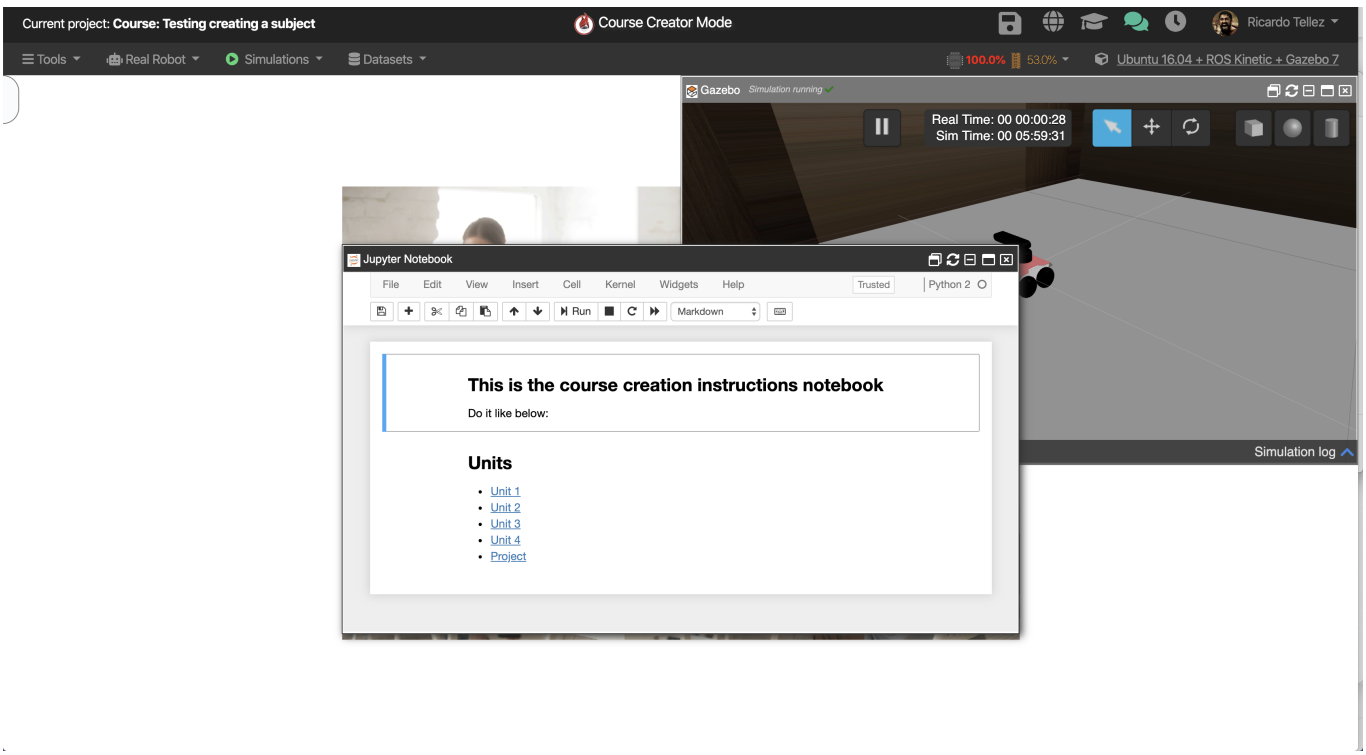
After pressing the button *Create* the template of the course will be created and you will get a screen similar to this:



2. Launch the template to start creating the content

Press the red button that says *Open* of the previous image. Then press again the same button on the screen that appears. Your template is being started.

After a few seconds, you will have the template opened. You should get a screen like this:



On the image you can see your working space for creating a course. You can see two main things:

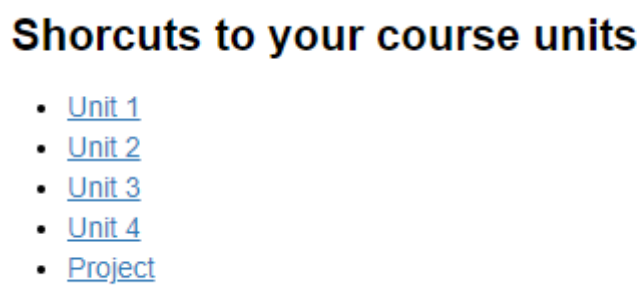
1. **The simulation** on the back: the robot simulation has been launched by default so you can start testing from minute one.
2. **The main Jupyter notebook**: similar to this notebook that you are reading

What you have to create for the course

- **The notebook units**. They are stored in the **notebook_ws**. Always create 5 units as indicated in the template. **This content is mandatory**.
  - **Unit 1**: introduction to the course with practical demo of what the student is going to achieve. Why it is important for robotics
  - **Unit 2, 3, 5**: here you include the content of the subject you want to teach, divided into 3 units.
  - **Project unit**: this is the final unit which proposes to the student a full project that uses all the concepts explained in the previous 3 units. This project requests the student to apply what he has learnt during the unit into a full robotics project.
- **Some code required**. If the code is C++, ROS, Python code, it must be stored in the **catkin_ws**. If the code is web code, then store it in **webpage_ws**. This is optional code you want to provide to the student, so he doesn't have to prepare everything. This is code needed to be in the system in order the student be able to learn the course subject. **This is optional**.
- **Datasets**. In case you require datasets for the course. **This is optional**.
- **Functional tests**. Stored in the **catkin_ws**. This is the code that test that all the exercises and commands you ask the user to develop will actually work in the online academy. **This is mandatory**.

3. Start creating the units

You need to create 5 units for each course. To access each notebook that you are going to fill, go to the top of the notebook that will open when you start the course and use the links provided. Press on the link to open the given notebook, and write/update thee notebooks.



IMPORTANT:

If you want to add code, include it in the *catkin_ws*, but, how will you do it? well let's explain it.

Inside the src folder in the catkin_ws you have to create a new folder with and **call it with the same title of the course**. According to our example it should be called **How_to_use_laser_data_with_ROS**.

Now in this folder you can add the packages of each unit, and the package with the tests that will check that everything is working well.

Example Unit 1 Introduction

You will create a notebook similar to this one.

How to use Laserscan data with ROS

Unit 1. Introduction

Why shoul you take this course?

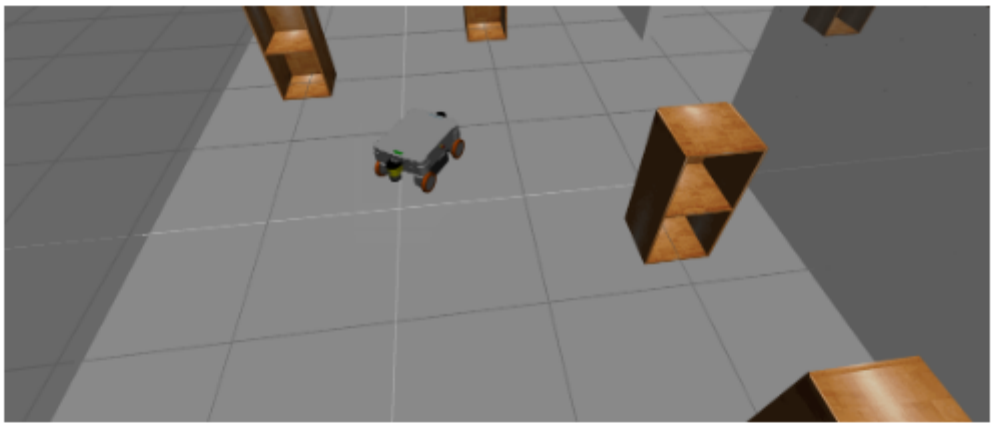
In robotics there are different types of data obtained from sensors that allow us to know the environment in which our robot operates, one of the most important is laserscan. **Laserscan is used to know the environment around the robot**, that is, to know if there is an obstacle and its distance from the robot, therefore, this **allows to know the environment and to navigate in it**. Navigation and evading obstacles is a fundamental part of mobile robotics, and using a laser sensor is basically essential, it is for this reason that it is very important to know how to use the data provided by this kind of laser.

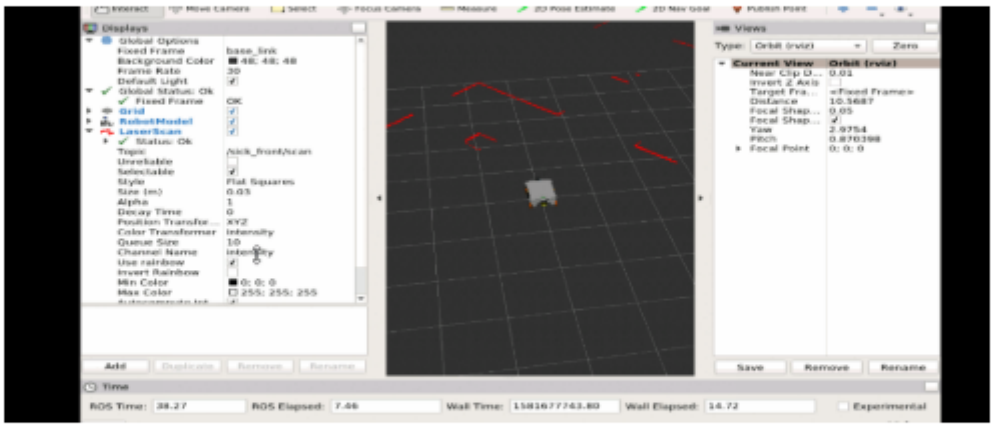
And using a laser is an easy way to do this?

Well, most of the time is quite easy, ROS has several packages that allow us to work quickly and easily with this type of data, however, there are occasions where it will be necessary to apply certain filters or modifications to obtain a better reading of the environment , which optimize the operation of our robot.

Hands-on right now!

We better start than with a practical example of the use of a laser. For this example, we will use the mpo_500 robot in one of our simulated environments, and observe the data emitted by our laser. Follow the following instructions.





Execute in Shell #1

In [ ]:
```
roslaunch unit1 introduction.launch
```

With this command you are executing a simple ROS program which allows you to see how laser is working and how can the robot use this data to navigate and avoid obstacles. Quite interesting, right? Keep reading for further instructions!

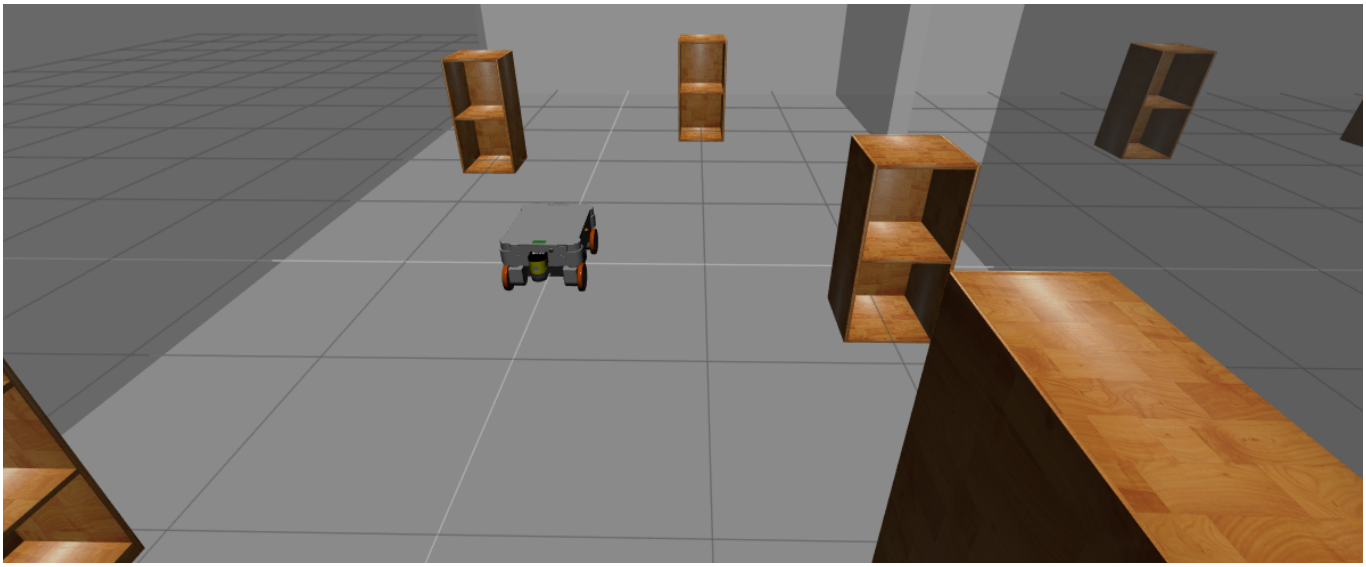
End of demo

What you will learn

But, is not only the notebook. you have to prepare the unit as a project, in order to have everything ready to the student, let's see what we did to get this unit finished.

Simulation

You have already the simulation already running, the one you choose before, so don't worry about that.



The code

In this case you as a demo that the student just will run to see what he or she could learn, you have to prepare everything,

starting with the package, the scripts, launch files or configuration of rviz

In this case we create a package an called it *unit1*, **don't forget that this should be in the *catkin_ws/src/How_to_use_laser_data_with_ROS***

In [ ]:

```
cd catkin_ws/src/How_to_use_laser_data_with_ROS
```

In [ ]:

```
catkin_create_pkg unit1 rospy
```

inside we create a folder and call it *src* and inside it we create a file and call it *avoid.py*. This is the file

In [ ]:

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan

class mpo_500():
     def __init__(self):
        self.mpo_500_vel_publisher = rospy.Publisher('/cmd_vel', Twist, queue_si
ze=1)
        self.mpo_500_sub = rospy.Subscriber("/sick_front/scan", LaserScan, self.
scan_callback)
        self.cmd = Twist()
        self.a = 0.0
        self.b = 0.0
        self.c= 0.0
        self.ctrl_c = False

        self.rate = rospy.Rate(10) # 10hz
        rospy.on_shutdown(self.shutdownhook)

     def scan_callback(self, msg):
         self.a = msg.ranges[90]
         self.b = msg.ranges[len(msg.ranges)/2]
         self.c = msg.ranges[len(msg.ranges)-90]


     def how_move(self):
         while not self.ctrl_c:
            if self.b >5:
                self.b=5
            if self.a >5:
                self.a=5
            if self.c >5:
                self.c=5



            if self.b >self.a and self.b> self.c :
                self.cmd.linear.x = 0.45
                self.cmd.angular.z = 0.0
                rospy.loginfo("Moving mpo_500 forward!")

            if self.a >self.b and self.a> self.c :
                self.cmd.angular.z = 0.15
                self.cmd.linear.x = 0.09
                rospy.loginfo("Moving mpo_500! left")
            if self.c >self.a and self.c> self.b :
                self.cmd.angular.z = -0.15
                self.cmd.linear.x = 0.09
                rospy.loginfo("Moving mpo_500! right")



            self.mpo_500_vel_publisher.publish(self.cmd)
             #print "a = "+ str(self.a)+"  b = "+str(self.b)+"  c= "+str(self.c)

     def shutdownhook(self):
        # works better than the rospy.is_shutdown()
        self.ctrl_c = True
```

```
if __name__ == '__main__':
    rospy.init_node('mpo_500_test', anonymous=True)
    mpo_500_object = mpo_500()
    try:
        mpo_500_object.how_move()



    except rospy.ROSInterruptException:
        pass
```

Run this script into the shell and then start rviz and create a configuration in order to see all the topics needed and show the student how our laser will work, we recommend you to save the configuration in a folder inside the package of unit 1 with the name *rviz*.Then we create a launch file with an rviz configuration inside a new folder called *launch*. This file will be called *introduction*

In [ ]:

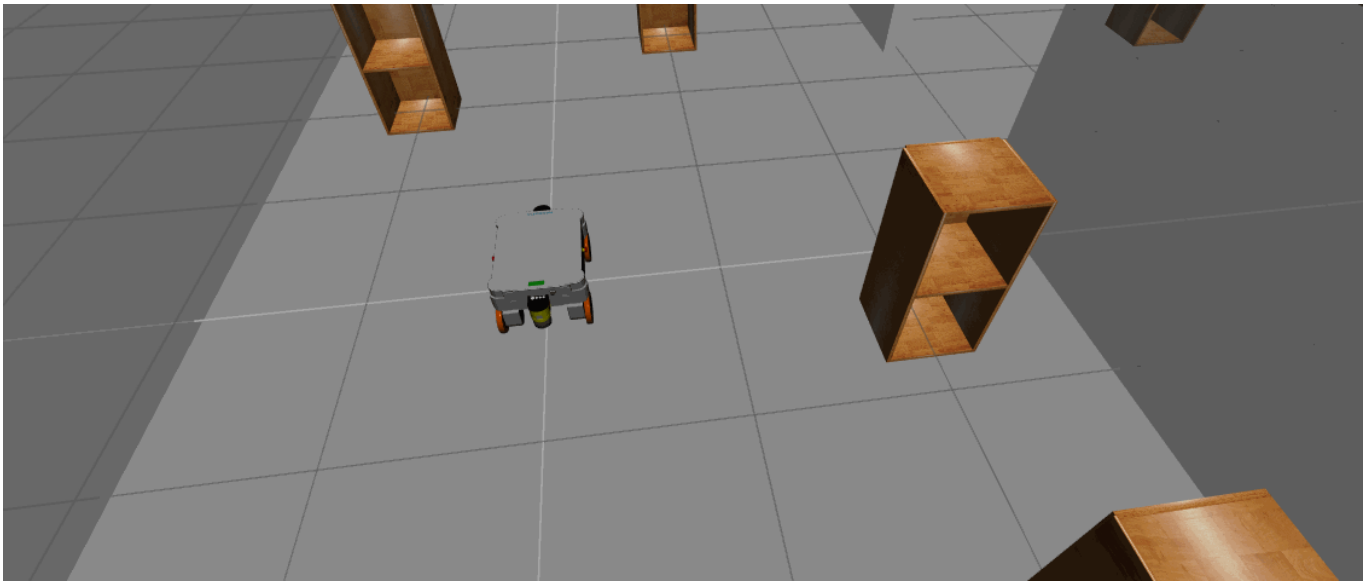
```
<launch>
    <!-- My Package launch file -->
    <node pkg="unit1" type="avoid.py" name="mpo_test"  output="screen">
    </node>
    <node type="rviz" name="rviz" pkg="rviz" args="-d $(find unit1)/rviz/laser.r
viz" />

</launch>
```

We know that you know how to do these steps, however, mentioning them is not too much.

Preparing the notebook

As you develop this unit we advise you to take pictures or gif, this will help the notebook to be better understood and that the student will pay more attention, here are some examples.



As an introductory part do not forget to comment that it is all that the student will learn in the course, and in turn will help you better organize the distribution of the following units

Example content unit (2,3,4)

This is an example of a part of a notebook of the content of unit 2

How to use Laserscan data with ROS

Unit 2. What is Laserscan and how to use it

Estimated time to completion: 1'5 hours

Simulated robot: mpo_500

What will you learn with this unit?

- What kind of msg use the laser's topic
- What contain this msg
- How can we see what is publishing the topic
- How can we use this data in a program

What kind of msg use the laser's topic

As we know each topic published by our robot contains a type of message, and in the case of the laser sensor topic it is no different. Being a sensor, the type of message is sensor_msgs / LaserScan which is a type of message that is composed as follows:

- std_msgs/Header header
  - uint32 seq
  - time stamp
  - string frame_id
- float32 angle_min
- float32 angle_max
- float32 angle_increment
- float32 time_increment
- float32 scan_time
- float32 range_min
- float32 range_max
- float32[] ranges
- float32[] intensities

And where do we get all this information, follow us and find out.

First of all let's see wich topics does our robot have. in shell #1 execute the following command

In [ ]:
```
rostopic list
```

```
user:~$ rostopic list
/clock
/cmd_vel
/cob_scan_unifier/scan_unified
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/joint_states
/odom
/rosout
/rosout_agg
/sick_back/scan
/sick_front/scan
/tf
/tf_static
```

We can see that we have some scan data let's see the info of **/sick_front/scan** to see it you can use the following command in shell#1.

In [ ]:
```
rostopic info /sick_front/scan
```

you will see something similar to this

```
user:~$ rostopic info /sick_front/scan
Type: sensor_msgs/LaserScan

Publishers:
 * /gazebo (http://roscomputer:39209/)

Subscribers: None
```

Here you can see publishers and subscribers of the topic, but now the important thing that you see is the **Type** where says wich type of msg is using. in this case **sensor_msgs/LaserScan**

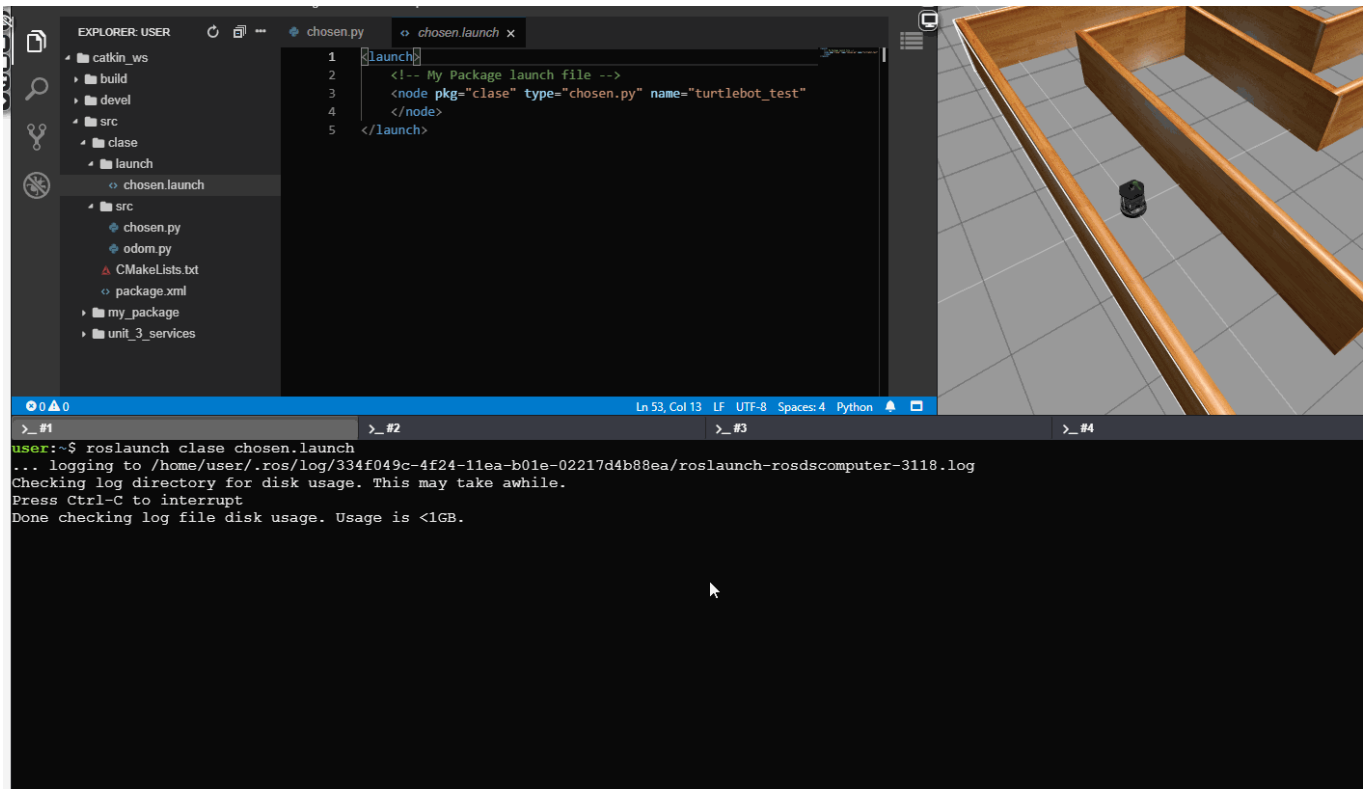
but how can we see the properties of this msg, run the following command into shell#1

As the first unit, is not enough to just write the notebook, sometimes you will have to prepare scripts, or packages or just prove in order to have some images to add in your notebook. let's see

Don't abuse the theory

The theory is very important when introducing new concepts, however, in robot ignite academy we consider that the practice is fundamental and the best way in which you can learn, that's why we give it so much importance, learn by doing. Do not hesitate to put concepts on the topics you are addressing, however, you can not leave something loose, have the student assimilate the knowledge by putting them into practice as soon as possible

as in the previous chapter, exploit the tools of the academy, you have to get used to the use of the shell, but it will be better if you can observe graphic tools or the same simulation, a complete practice can help to settle all the knowledge imparted much better



Remember that the academy has a distribution that allows you to see many things optimally at the same time, take advantage of this distribution so that the student can observe multiple things at the same time, in this example, an output that shows the terminal and how it acts in the simulation .

Also due to this distribution remember that the student has listed the shell that you can use, do not forget to specify in which shell to run a command, so as not to confuse the student in the process

Images of the unit

We also know that a picture is worth a thousand words, so do not forget to include in your notebook various images that show the process through which the student will go through to complete the unit. This will allow you to guide the student, and prevent them from getting lost along the way, and give you the assurance that everything is tested and is on track.

For example here are some pictures that we use in unit 2

```
user:~$ rostopic list
/clock
/cmd_vel
/cob_scan_unifier/scan_unified
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/joint_states
/odom
/rosout
/rosout_agg
/sick_back/scan
/sick_front/scan
/tf
/tf_static
```

```
user:~$ rostopic info /sick_front/scan
Type: sensor_msgs/LaserScan

Publishers:
 * /gazebo (http://rosdscomputer:39209/)

Subscribers: None
```

```
user:~$ rosmsg show sensor_msgs/LaserScan
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

One of the most difficult parts is to put yourself in the student's place, as you know the subject you are teaching, sometimes many steps may seem obvious to us, however, for someone who is just beginning it will be important to have each procedure step by step if possible.

Example Unit 5 project

This is an example of the notebook of the project of the course

How to use Laserscan data with ROS

Unit 5. Project

Estimated time to completion: 10 hours

What will you learn in this unit?

- Practice everything you learn through the course
- Put together everything that is needed in order to complete the task of the project.

Help the mpo_500 to avoid all the obstacles in the office.

Help the mpo_500 to avoid all obstacles while moving around the office, the robot must reach both ends of the office dodging furniture and tables, in the case of the tables take into account that you should avoid colliding with the legs due to the robot height

Basically, in this project, you will have to:

Apply all the theory that is given in the course Decide on a strategy to solve the problem Implement this strategy in the simulation environment Do as many tests as required in the simulation environment until it works To achieve success in this project, please follow the 6 steps below that will provide clear instructions, and even solutions.

Also, remember to:

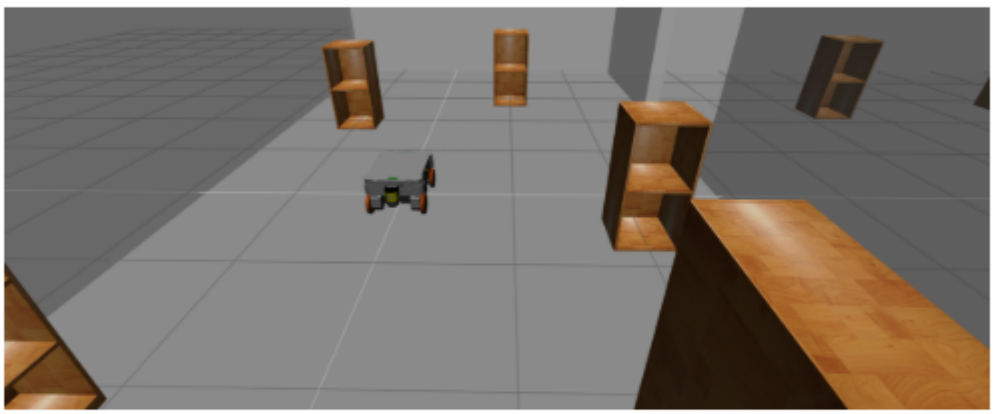
- Create your packages and code in the simulation environment as you have been doing throughout the course.
- Use the consoles to gather information about the status of the simulated robot.
- Use the IDE to create your programs and execute them through the consoles, observing the results on the simulation screen. - You can use other consoles to watch calls to topics, services, or action servers.
- Everything that you create in this unit will be automatically saved in your space. You can come back to this unit at any time and continue with your work from where you left off.
- Every time you need to reset the position of the robot, just press the restart button in the simulation window.
- Use the debugging tools to try to find what is not working and why (for instance, the rviz tool is very useful for this purpose).

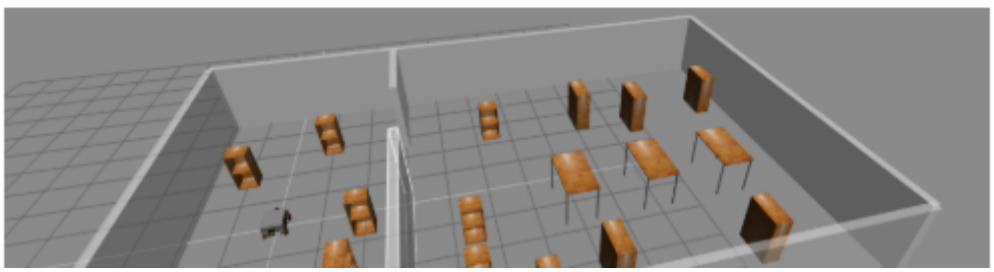
What does mpo_500 Provide to Program it? So the main question is, what can you do with mpo_500 from a ROS programming point of view? Which sensors and actuators does mpo_500 provide that will allow you to do the test?

Good question! mpo_500 provides the following sensors and actuators:

Sensors -Laser sensor: mpo_500 has two laser sensor, and topics for each one and a topic that have the merge of both laser, you can find them because they finish in /**scan**. -Odometry: The odometry of the robot can be accessed through the /odom topic. **Actuators** -Speed: You can send speed commands to move the robot through the /cmd_vel topic.

Now that you know the relevant topics of the robot, it is your job to figure out the types of messages, and how to use them, in order to make the robot do what you want it to do.



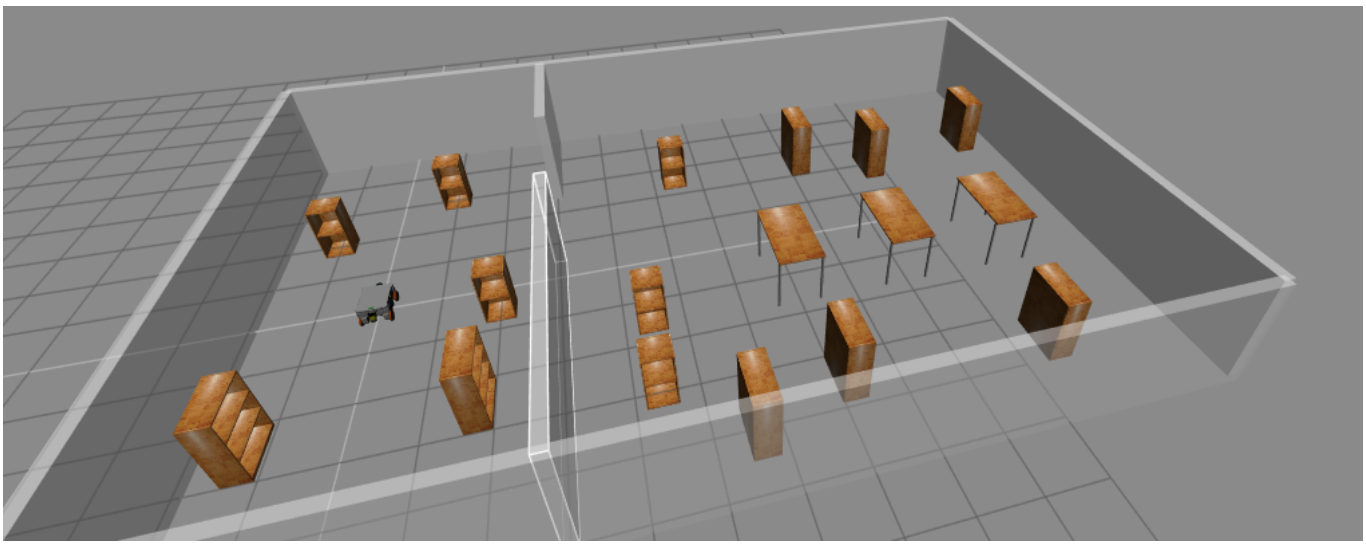


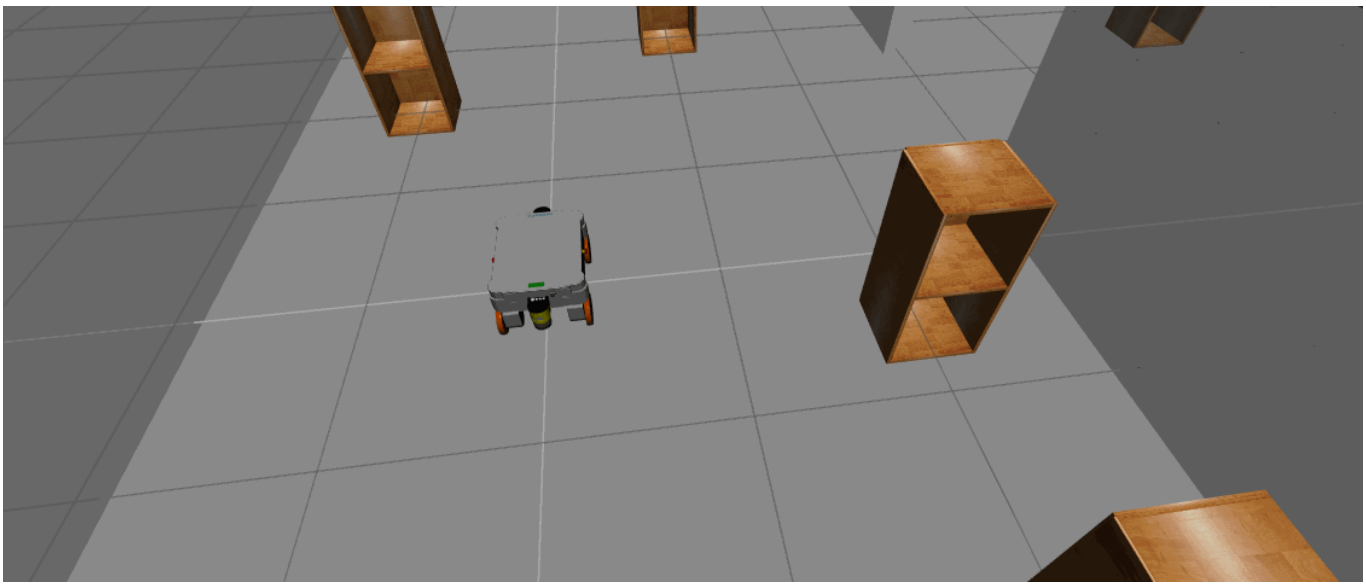
Projects that involve a challenge

The projects to be carried out must contain the application of everything learned in the course in such a way that it represents a challenge present in real situations within the robotics, the student must feel that the project he is carrying out is interesting, useful, and that he demonstrates How much you have learned during the course.

It is important to specify the limits and scope of the project, and many times to draw a path so that the student does not deviate from the objective, however, it should not be given so much help so that the student feels that self-solvency when managing to face this challenge.

Never forget that the images are worth a thousand words, and when explaining the environment where the project will be developed or the objective that the robot should meet, explaining through images or videos can help to better understand the situation and avoid confusion on the part of the student.





4. Create the functionality tests

Every course needs to contain functionality tests that automatically check that the commands you request the student to do, will work.

How to create a test

Inside **catkin_ws/src/[name_of_the_course]** create a new package and call it **tests_[name_of_the_course]**

In our case, it will be, go to ***catkin_ws/src/How_to_use_laser_data_with_ROS*** and create a new package with ***tests_How_to_use_laser_data_with_ROS*** as a name.

So, in the Shell run the following command.

In [ ]:

```
$ cd ~/catkin_ws/src/How_to_use_laser_data_with_ROS
```

In [ ]:

```
$ catkin_create_pkg tests_How_to_use_laser_data_with_ROS rospy
```

Now you have the test package you have to create some scripts, inside a folder with ***scripts*** as a name, that will test the units that you have already created. But there is a script that all test should have so create a new script and call it *testing_library.sh*, now inside it copy the following code.

In [ ]:

```
#! /bin/bash

BLUETEXT="\033[1;34m"
REDTEXT="\033[1;31m"
WHITETEXT="\033[0;37m"
GREENTEXT="\033[1;32m"
RESET_COLOR="\033[0m"

LIB_RESULT=()

LIB_PASSED_TESTS=0
LIB_FAILED_TESTS=0
LIB_TOTAL_TESTS=0

function printTestStarted {
    echo -e "\n\n ${BLUETEXT}Testing Line ${BASH_LINENO[0]} : ${FUNCNAME[1]}${RE
SET_COLOR} \n\n"
}

function testPassed {
    LIB_RESULT+=("${GREENTEXT} Line ${BASH_LINENO[0]} : Passed - ${FUNCNAME[1]}
${RESET_COLOR}")
    ((LIB_PASSED_TESTS++))
    ((LIB_TOTAL_TESTS++))
}

function testFailed {
    LIB_RESULT+=("${REDTEXT} Line ${BASH_LINENO[0]} : Failed - ${FUNCNAME[1]}${R
ESET_COLOR}")
    ((LIB_FAILED_TESTS++))
    ((LIB_TOTAL_TESTS++))
}

function showTestResults {
    echo -e "\n${BASH_SOURCE}:\n"
    for msg in "${LIB_RESULT[@]}"; do
        echo -e $msg;
    done
    echo -e "\n RESULTS: $LIB_PASSED_TESTS tests passed.  $LIB_FAILED_TESTS test
s failed.  $LIB_TOTAL_TESTS tests total.\n"


}
```

The rest of the scripts that will test units have an structure that will help ypu make the test easier.

Here is an example of a unit test. This test will look for a specific topic, a topic that is essential to run the unit examples, in this particular case is to look for the **"/sick_front/scan"** that will be necessary to use the laser that is in front of the mpo_500 and the topic **"/cmd_vel"** that will be necessary to make the robot move.

In [ ]:

```
#!/usr/bin/env bash

function setup_testing_library {
    echo -e "$BLUETEXT\nSetting Library Test env"

    DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" >/dev/null && pwd )"
    echo $DIR
    source $DIR/testing_library.sh


    echo -e "$BLUETEXT\nSetting Library Test DONE"

    echo -e "$WHITETEXT\n"
}



function ros_env_setup {
    echo "Setting up ROS env"

    . /home/user/.bashrc
    source /opt/ros/kinetic/setup.bash
    source /home/user/catkin_ws/devel/setup.bash

    echo "Setting up ROS env DONE"
}


function launch_unit1_laser_test {


  sleep 5

  local status=0

  expected_topics=(
      "/sick_front/scan"
      "/cmd_vel"


  )

  echo "Checking Topics..."${expected_topics[@]}
  for topic in "${expected_topics[@]}"; do
      topics_found=$(rostopic list | grep "$topic" | wc -l)
      if [ "$topics_found" -lt 1 ]; then
          echo -e "$REDTEXT\nTopic $topic not found!"
          echo -e "\n$WHITETEXT"
          status=1
      fi
  done


  sleep 2

  if [ $status -eq 0 ]; then

      echo -e "$BLUETEXT\nTest Passed"
      echo -e "\n$WHITETEXT"
```

```
      testPassed
  else
      echo -e "$BLUETEXT\nTest Failed"
      echo -e "\n$WHITETEXT"
      testFailed
  fi

}

function launch_unit1_demo {


    roslaunch unit1 introduction.launch & demo_unit1_pid=$!
    sleep 15
    local status=0
    expected_nodes=(
        "/mpo_test"
    )
    for node in "${expected_nodes[@]}"; do
        nodes_found=$(rosnode list | grep "$node" | wc -l)
        if [ $nodes_found -lt 1 ]; then
            kill $demo_unit1_pid
            echo -e "$REDTEXT\n/Checking.., node NOT FOUND"
            echo -e "\n$WHITETEXT"
            echo "Giving time to close everything..."
            testFailed
            status=1
            sleep 15
        fi
    done
    if [ $status -eq 0 ]; then
        kill $demo_unit1_pid
        echo -e "$GREENTEXT\n/Checking..., node FOUND"
        echo -e "\n$WHITETEXT"
        echo "Giving time to close everything..."
        testPassed
        sleep 15
    fi

}


function main {
    set +x
    setup_testing_library
    ros_env_setup
    launch_unit1_laser_test
    launch_unit1_demo
    showTestResults

}

main

echo -e "$GREENTEXT\n Unit0 Test Finished OK"
```

but what will have every test in common, let's see step by step.

In [ ]:

```
#!/usr/bin/env bash

function setup_testing_library {
    echo -e "$BLUETEXT\nSetting Library Test env"

    DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" >/dev/null && pwd )"
    echo $DIR
    source $DIR/testing_library.sh


    echo -e "$BLUETEXT\nSetting Library Test DONE"

    echo -e "$WHITETEXT\n"
}
```

the first part, setting up the testing library, This is a fundamental part for the operation of the test, here is where we add the library before that show if the test passed or failed. let's see the next part.

In [ ]:

```
function ros_env_setup {
    echo "Setting up ROS env"

    . /home/user/.bashrc
    source /opt/ros/kinetic/setup.bash
    source /home/user/catkin_ws/devel/setup.bash

    echo "Setting up ROS env DONE"
}
```

In this part it will setting up the ROS environment, when this test will be running, ROS will not be set up, so it's too important to add this part to every test that you will create.

in the next part of the test we have the following...

In [ ]:

```
function launch_unit1_laser_test {


  sleep 5

  local status=0

  expected_topics=(
      "/sick_front/scan"
      "/cmd_vel"

  )

  echo "Checking Topics..."${expected_topics[@]}
  for topic in "${expected_topics[@]}"; do
      topics_found=$(rostopic list | grep "$topic" | wc -l)
      if [ "$topics_found" -lt 1 ]; then
          echo -e "$REDTEXT\nTopic $topic not found!"
          echo -e "\n$WHITETEXT"
          status=1
      fi
  done


  sleep 2

  if [ $status -eq 0 ]; then

      echo -e "$BLUETEXT\nTest Passed"
      echo -e "\n$WHITETEXT"
      testPassed
  else
      echo -e "$BLUETEXT\nTest Failed"
      echo -e "\n$WHITETEXT"
      testFailed
  fi

}
```

This is the function that will try to find the topics that you wrote in the *expected_topics*, is an specific test for this unit so could be similar or maybe you will have other tests to check if your unit is working correctly, like launch some file, or see if the odometry changes, etc.

But this test only shows that the simulation is working and the robot has the topics needed to run the launch file, but do we have a problem to launch it?, the next part will show us.

In [ ]:

```
function launch_unit1_demo {


    roslaunch unit1 introduction.launch & demo_unit1_pid=$!
    sleep 15
    local status=0
    expected_nodes=(
        "/mpo_test"
    )
    for node in "${expected_nodes[@]}"; do
        nodes_found=$(rosnode list | grep "$node" | wc -l)
        if [ $nodes_found -lt 1 ]; then
            kill $demo_unit1_pid
            echo -e "$REDTEXT\n/Checking.., node NOT FOUND"
            echo -e "\n$WHITETEXT"
            echo "Giving time to close everything..."
            testFailed
            status=1
            sleep 15
        fi
    done
    if [ $status -eq 0 ]; then
        kill $demo_unit1_pid
        echo -e "$GREENTEXT\n/Checking..., node FOUND"
        echo -e "\n$WHITETEXT"
        echo "Giving time to close everything..."
        testPassed
        sleep 15
    fi

}
```

It works similar to the previous part, it will look for a node, that show if the launch file is running well. In this particular case we gonna look for the node called */mpo_test* and if we don't find it, it means that there is a problem with the launch file or some script that is inside it.

And as you see we have the structure with functions, because is a good way to modify or debug if something is going wrong. So let's see the main function that will run everything in this test.

In [ ]:

```
function main {
    set +x
    setup_testing_library
    ros_env_setup
    launch_unit1_laser_test
    launch_unit1_demo
    showTestResults

}
```

here we have the main function, set +x is to hide the commands that are running in the shell, the rest are functions that we're calling in order to run every part of the test. And the last part of the test will be the following.

In [ ]:

```
main

echo -e "$GREENTEXT\n Unit0 Test Finished OK"
```

that is only running the main function and making an echo that said that the test finished :D

How to run a test

Well, let's create a file inside the ***catkin_ws/src/How_to_use_laser_data_with_ROS/tests_how_to_use_laser_data_with_ROS/scripts*** and call it **unit1_test_script.sh**, copy the following code in this file.

In [ ]:

```
#!/usr/bin/env bash

function setup_testing_library {
    echo -e "$BLUETEXT\nSetting Library Test env"

    DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" >/dev/null && pwd )"
    echo $DIR
    source $DIR/testing_library.sh


    echo -e "$BLUETEXT\nSetting Library Test DONE"

    echo -e "$WHITETEXT\n"
}



function ros_env_setup {
    echo "Setting up ROS env"

    . /home/user/.bashrc
    source /opt/ros/kinetic/setup.bash
    source /home/user/catkin_ws/devel/setup.bash

    echo "Setting up ROS env DONE"
}


function launch_unit1_laser_test {


  sleep 5

  local status=0

  expected_topics=(
      "/sick_front/scan"
      "/cmd_vel"


  )

  echo "Checking Topics..."${expected_topics[@]}
  for topic in "${expected_topics[@]}"; do
      topics_found=$(rostopic list | grep "$topic" | wc -l)
      if [ "$topics_found" -lt 1 ]; then
          echo -e "$REDTEXT\nTopic $topic not found!"
          echo -e "\n$WHITETEXT"
          status=1
      fi
  done


  sleep 2

  if [ $status -eq 0 ]; then

      echo -e "$BLUETEXT\nTest Passed"
      echo -e "\n$WHITETEXT"
```

```
      testPassed
  else
      echo -e "$BLUETEXT\nTest Failed"
      echo -e "\n$WHITETEXT"
      testFailed
  fi

}

function launch_unit1_demo {


    roslaunch unit1 introduction.launch & demo_unit1_pid=$!
    sleep 15
    local status=0
    expected_nodes=(
        "/mpo_test"
    )
    for node in "${expected_nodes[@]}"; do
        nodes_found=$(rosnode list | grep "$node" | wc -l)
        if [ $nodes_found -lt 1 ]; then
            kill $demo_unit1_pid
            echo -e "$REDTEXT\n/Checking.., node NOT FOUND"
            echo -e "\n$WHITETEXT"
            echo "Giving time to close everything..."
            testFailed
            status=1
            sleep 15
        fi
    done
    if [ $status -eq 0 ]; then
        kill $demo_unit1_pid
        echo -e "$GREENTEXT\n/Checking..., node FOUND"
        echo -e "\n$WHITETEXT"
        echo "Giving time to close everything..."
        testPassed
        sleep 15
    fi

}


function main {
    set +x
    setup_testing_library
    ros_env_setup
    launch_unit1_laser_test
    launch_unit1_demo
    showTestResults

}

main

echo -e "$GREENTEXT\n Unit0 Test Finished OK"
```

This is the test that you saw before as an example.

Now run this test in the shell with the following commands. **Remember that the simulation has to be running while the test is running**.

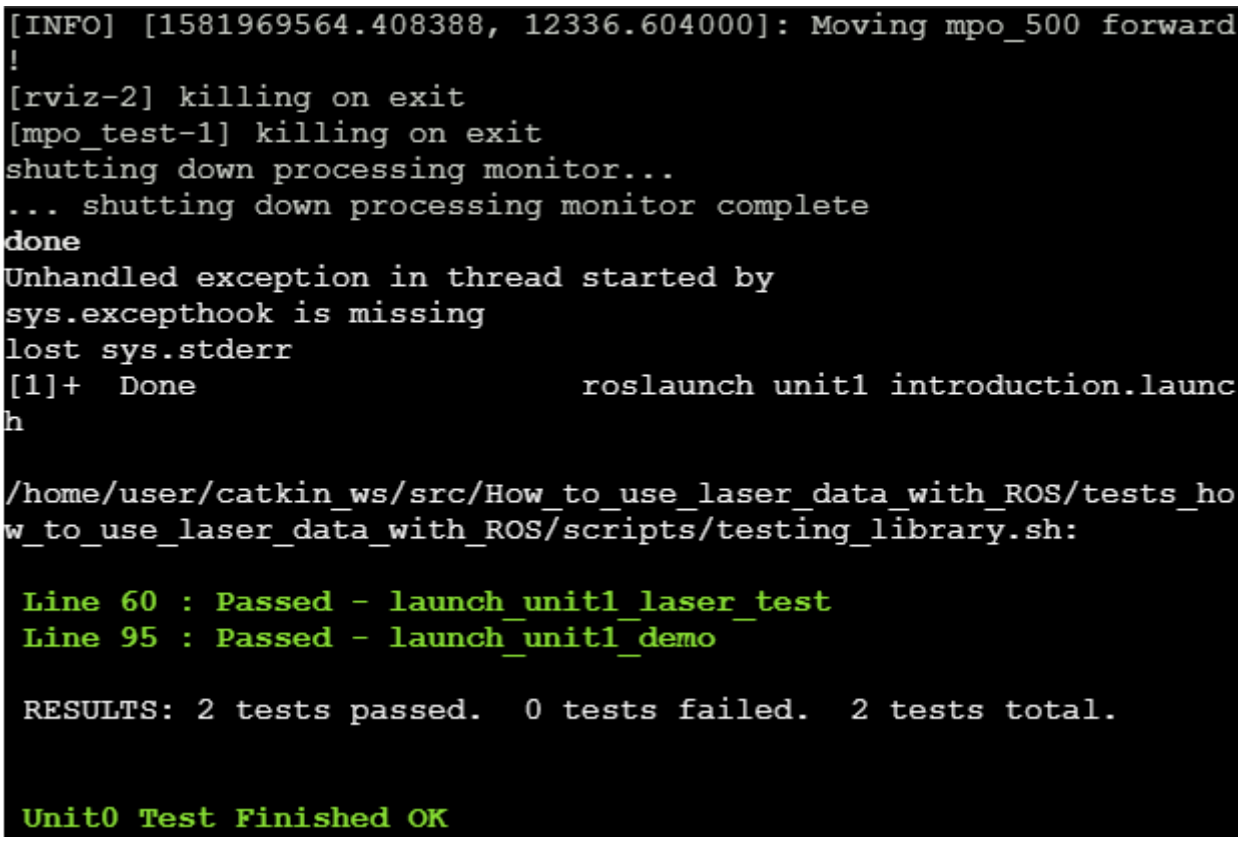
In [ ]:

```
$ cd catkin_ws/src/How_to_use_laser_data_with_ROS/tests_How_to_use_laser_data_with_ROS/scripts/
```

In [ ]:

```
$ . unit1_test_script.sh
```

And you will have something similar to the next image.



Where you can see that there are the topics that we want to find. But, does this test really work? let's modify and try to find a topic that doesn't exist.

in this part of the code.

In [ ]:

```
expected_topics=(
     "/sick_front/scan"
     "/cmd_vel"

 )
```

add a fake topic, for example:

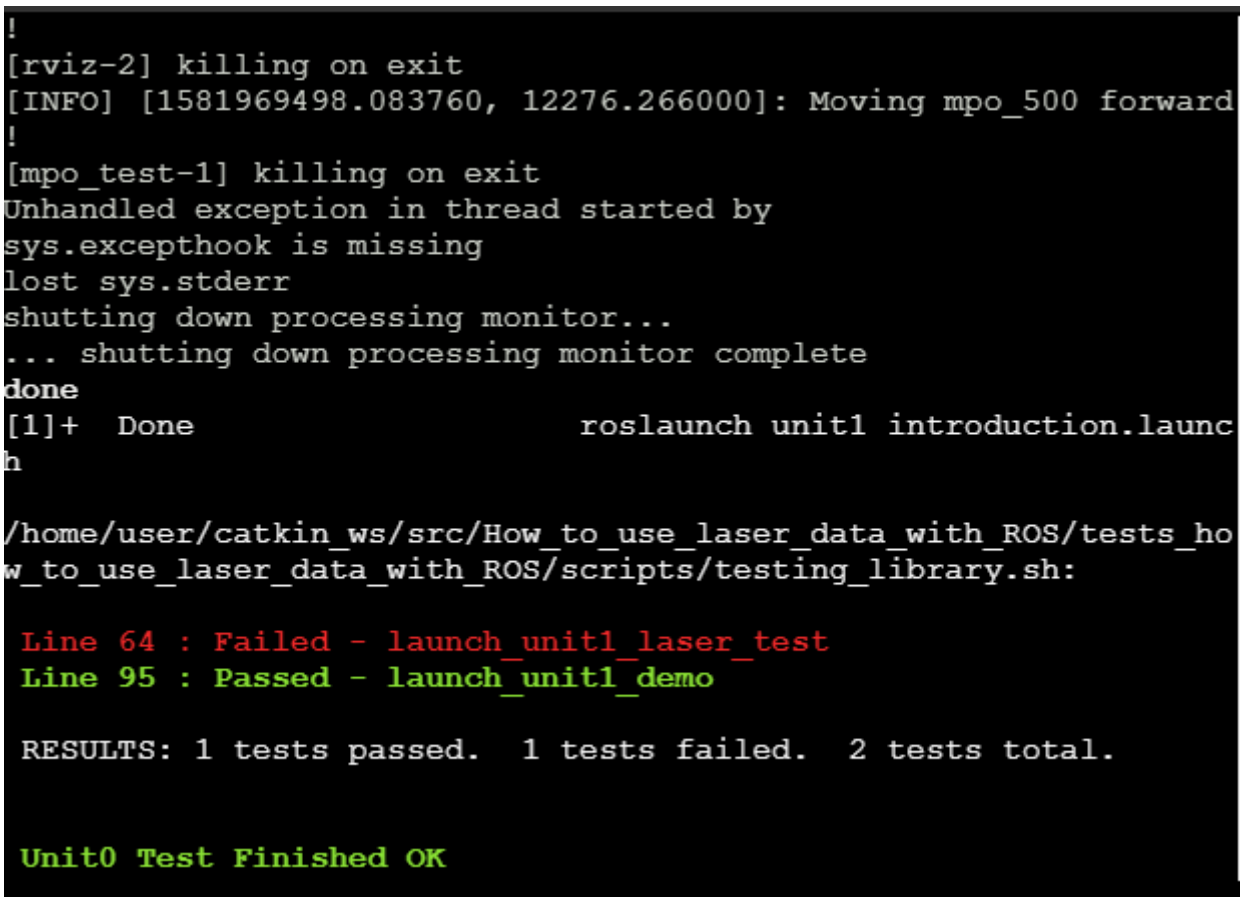
In [ ]:

```
 expected_topics=(
      "/sick_front/scan"
      "/cmd_vel"
      "/the_true_love"

  )
```

and then run the test again a see what happen.

you will have something similar to the next picture.



As you can see you cannot find the true love, well as a topic you know ;) , and this is how we see that this test works correctly. But the other part will work? let's modify our launch of unit1 to see if the test detect it.

go to ***catkin_ws/src/How_to_use_laser_data_with_ROS/unit1/launch/introduction.launch*** and change it like the following code

In [ ]:

```
<launch>
    <!-- My Package launch file -->
    <node pkg="unit1" type="avoid2.py" name="mpo_test"  output="screen">
    </node>
    <node type="rviz" name="rviz" pkg="rviz" args="-d $(find unit1)/rviz/laser.r
viz" />

</launch>
```

now execute the test again, you will see something similar to this

```
scores, and dashes.
ERROR: cannot launch node of type [unit1/avoid2.py]: can't locat
e node [avoid2.py] in package [unit1]
process[rviz-2]: started with pid [1249]

/Check1, nodes NOT FOUND


Giving time to close everything...
[rviz-2] killing on exit
shutting down processing monitor...
... shutting down processing monitor complete
done
[1]+  Done                    roslaunch unit1 introduction.launc
h

/home/user/catkin_ws/src/How_to_use_laser_data_with_ROS/tests_ho
w_to_use_laser_data_with_ROS/scripts/testing_library.sh:

 Line 64 : Failed - launch_unit1_laser_test
 Line 85 : Failed - launch_unit1_demo

 RESULTS: 0 tests passed.  2 tests failed.  2 tests total.
```

and if you correct the topics and let this last fail, you will have something similar to the next image.

```
process[rviz-2]: started with pid [2493]

/Checking.., node NOT FOUND


Giving time to close everything...
[rviz-2] killing on exit
shutting down processing monitor...
... shutting down processing monitor complete
done
[1]+  Done                    roslaunch unit1 introduction.launc
h

/home/user/catkin_ws/src/How_to_use_laser_data_with_ROS/tests_ho
w_to_use_laser_data_with_ROS/scripts/testing_library.sh:

 Line 60 : Passed - launch_unit1_laser_test
 Line 85 : Failed - launch_unit1_demo

 RESULTS: 1 tests passed.  1 tests failed.  2 tests total.
```

Well now you know how to create test to check if your units is working well.

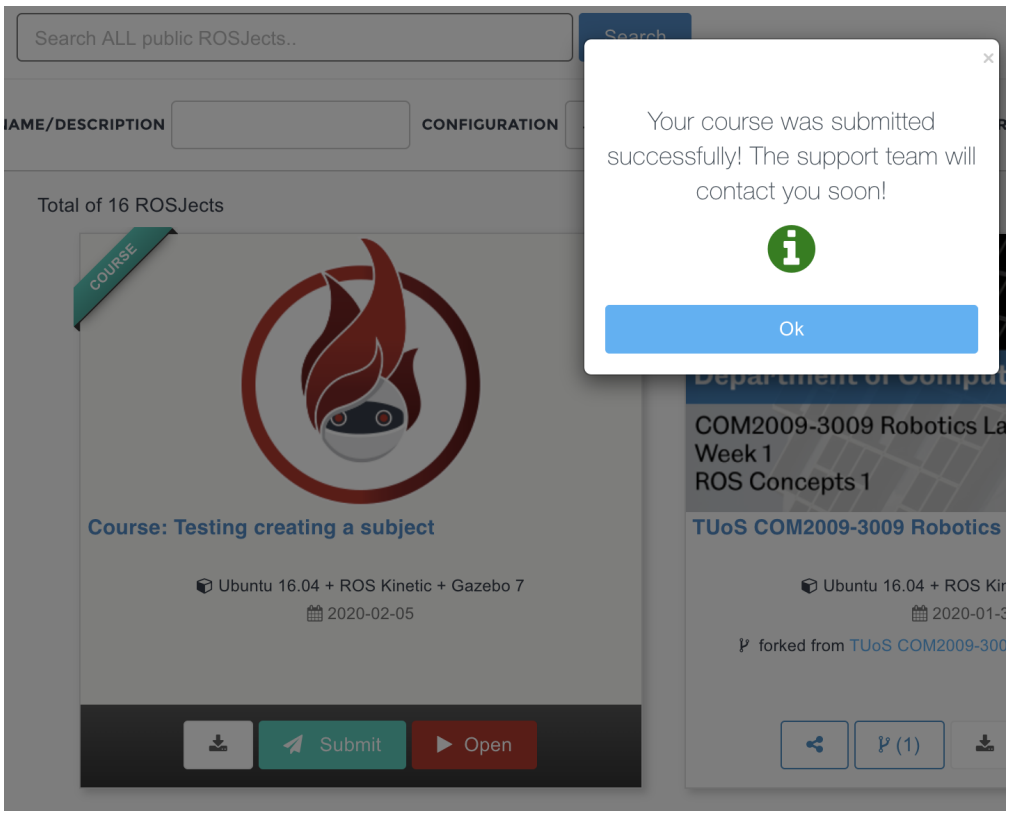
5. Saving the course

You don't need to save anything. Everything is saved automatically

6. Submit for revision

Once you are happy with the content, you have to submit it to Robot Ignite Academy for revision.

- Go to your list of rosjects by clicking on the *world* icon on the top menu.
- On the new tab, go to your course rosject and press the green button *Submit*
- Wait for our revision



7. Once accepted

- You will have to transfer the rights of use to us, so we can use the course in the Robot Ignite Academy
- You will receive 1.500€ per course. You will need to give us your bank account and personal details, as well as
- You will appear as the author of the course in the Academy course info
- We will take charge of maitaining the course
- We may translate it to other languages

Additional info

- You cannot create a course about any subject. Check the list of subjects we are requesting.
- Programming can be done in Python or C++
- Language of the courses must be English. Do not worry too much about English errors because we will proof read your course before publication
- Bear in mind that other people can be creating the same course as you, but we can only accept one of them. In case two courses about the same subjects are summitted, we will select the one with higher quality according to The Construct criteria.
- Courses can be about ROS or about robotics subjects, but all of them must contain practices and exercises with ROS robots to understand the concepts
- You cannot use the course material in other courses of yours or trainings or any other website, book or else. You are transferring the rights to us, and for that you are getting paid.

8. List of courses required for the academy

About ROS

- ROS Control advanced: How to create Harware Interfaces
- How to do simulations with Gazebo
- Gazebo Plugins How To
- How to debug a ROS program
- How to use a ROS buildfarm

About ROS2

- ROS2 Navigation
- ROS2 MoveIt
- How to convert ROS 1 programs to ROS2

About Robotics

- How to calibrate a robot
- Robot Navigation basic algorithms (SLAM, Kalman filters and so)
- Robot Kinematics explained
- Visual servoing

Mission completed!!

If you liked this video, please support us!

Really... we need your support!!!!

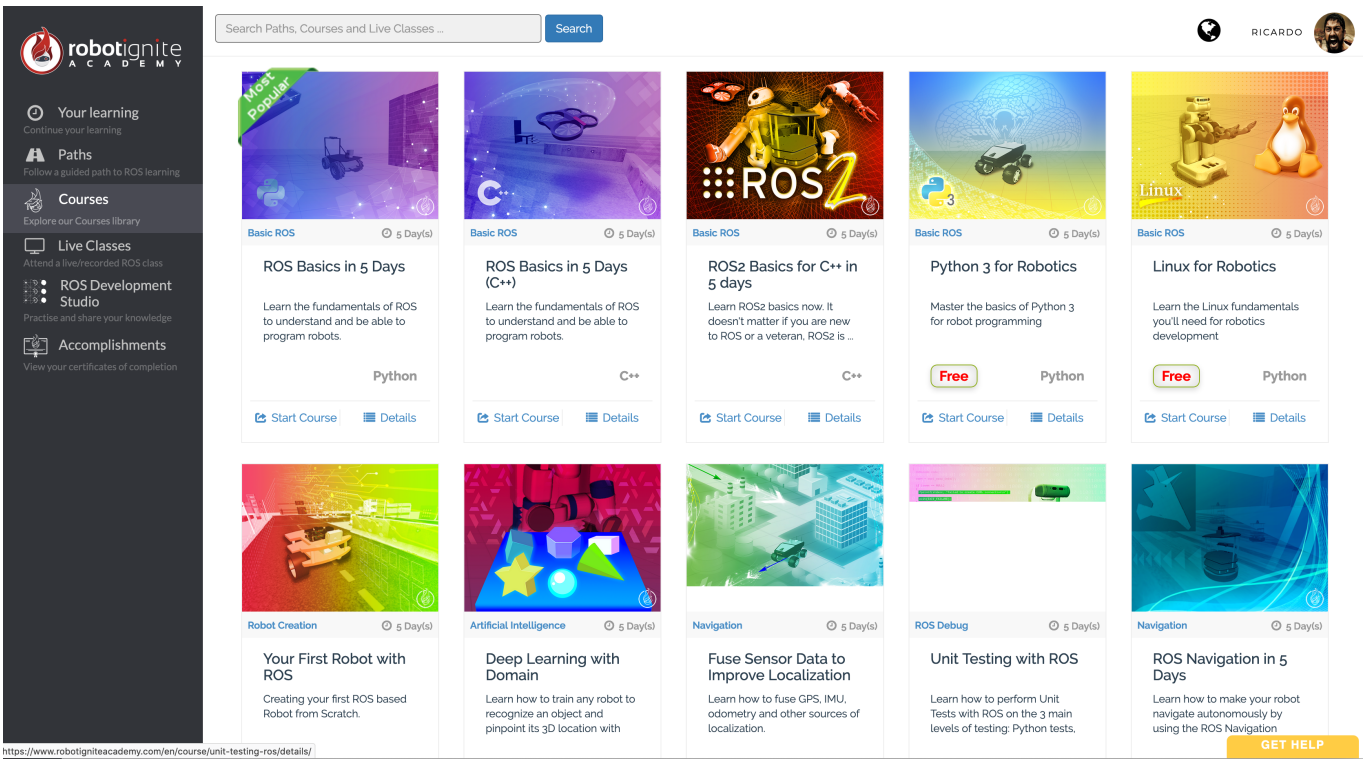
How can you support us?

1. Subscribe to our ROS online academy and become a Master of ROS Development

Go to our online academy. There is no faster way and funnier to learn ROS because we use the same method we did here.

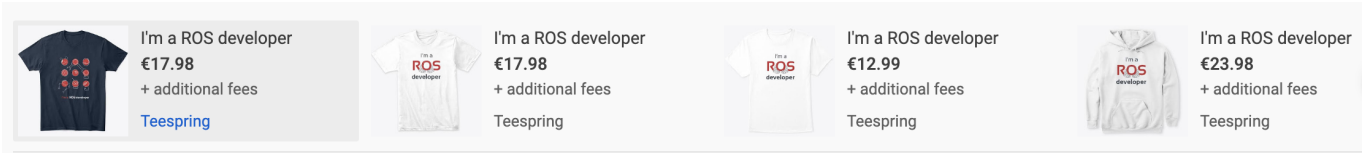
We call the 30/70 method

- **30% of the time learning theory**
- **70% of the time practicing with simulated robots**



Check it out at [http://robotignite.academy (http://robotignite.academy)](http://robotignite.academy)

2. Buy one ROS Developers T-shirt!



You can buy them at our Teespring area (https://teespring.com/stores/ros-developers (https://teespring.com/stores/ros-developers))

3. Give us a like in Youtube and subscribe to the channel

- **Go to our Youtube Channel (https://www.youtube.com/channel/UCt6Lag-vv25fTX3e11mVY1Q (https://www.youtube.com/channel/UCt6Lag-vv25fTX3e11mVY1Q)) and subscribe (it is free!!!)**
- **Give us a like to this video**

KEEP PUSHING YOUR ROS LEARNING WITH PATIENCE AND GOOD HUMOUR!

Build the future, become a ROS DEVELOPER

In [ ]: