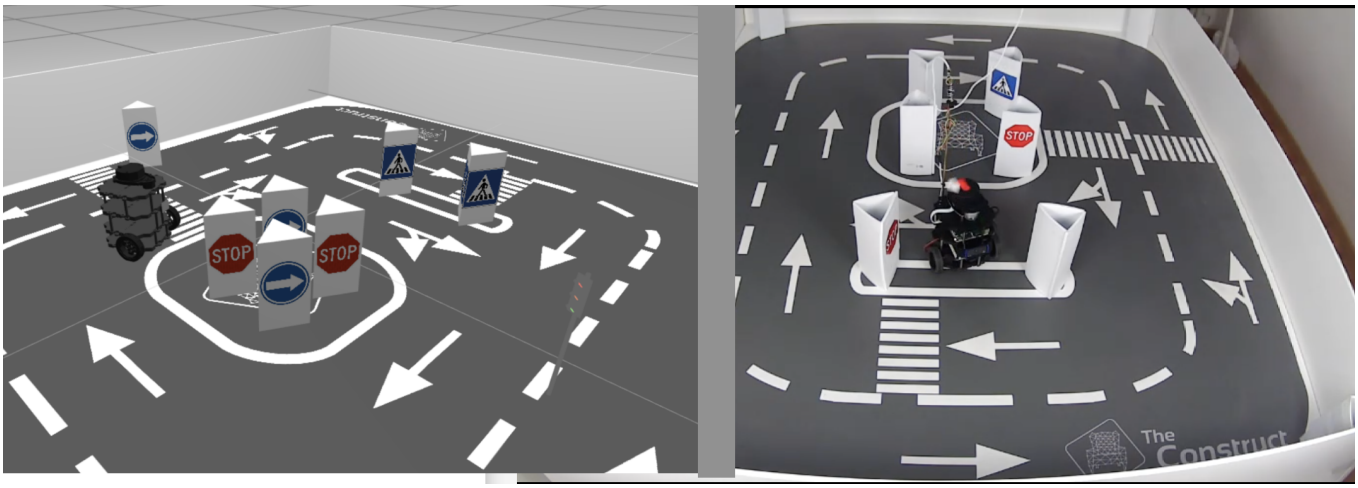
ROS2 Basics in 5 Days: Course Project

In this rosject, you will have to apply all that you have learned throughout the ROS2 Basics course to a real robot.

- You will practice with a simulation and a real robot Turtlebot3
- You will launch the robot simulation here
- The real robot is running in Barcelona, Spain. You will connect remotely to it from this rosject



Section 0 How to proceed

The rosject environment

If this is the first time you've used a rosject, then it might seem a little confusing at first. Don't worry, you'll get the hang of it.

Rosjects are like a real development environment for ROS robots:

- You will need to start yourself the robot simulation
- You will need to decide when to open a Linux terminal
- You will need to decide when to connect to the real robot
- You will need to decide when to open and work with the IDE

This is how the real ROS developers work every day.

How to launch a terminal.

To open a terminal, use the following icon of the bottom bar.

Terminals are where you will do most of the work: looking for topics, running commands and checking nodes are just a few things. It is the main interface with the linux system of the robot.

How to launch the IDE

To open the IDE, use the following icon of the bottom bar.

That is the code editor. Very similar to Visual Studio Code. You will write your programs here, and you can also see the folder organization of the rosject. It is worth taking a minute here to identify the most important worskpaces where you will be working and the ones you don't need to pay attention:

- `ros2_ws` : The main workspace. ALL of your work will be in there, inside the `src` directory
- `simulation_ws` : The workspace where all of the code relating to the simulation is. You do not need to work in there
- `notebook_ws` : The workspace where the notebook and associated images are stored. You do not need to work in there

How to open the instructions notebook

The instructions notebook is what you are reading right now. Contains the instructions of what you need to do to complete the project.

To open the notebok, use the following icon of the bottom bar.

How to launch a simulation

As in every profesional robot programming system, you will need to decide when to launch the simulation.

The simulation will not be there automatically for you as it happens in the course.

To launch a simulation:

1. You need to have the simulation code inside the `simulation_ws` (already done in this rosject)
2. You need to know before hand the Linux command necessary to launch that simulation (you will get it below)
3. You will need to open a terminal
4. Then type on the terminal the Linux command
5. Once the simulation has fully started, the simulation window will automatically appear
6. In case you close that window and want to re-open it again, use the following icon of the bottom bar:

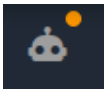
How to open the graphical tools

You already know about the graphical tools:

Opens `rviz` **if rviz has been launched from a terminal**. You will also see any other GUI applications in here, like `rqt` .

How to connect to the real robot

When you are ready to connect to the real robot and practice with it, you will need to do a booking of the robot and then use the following icon to connecto the real robot:

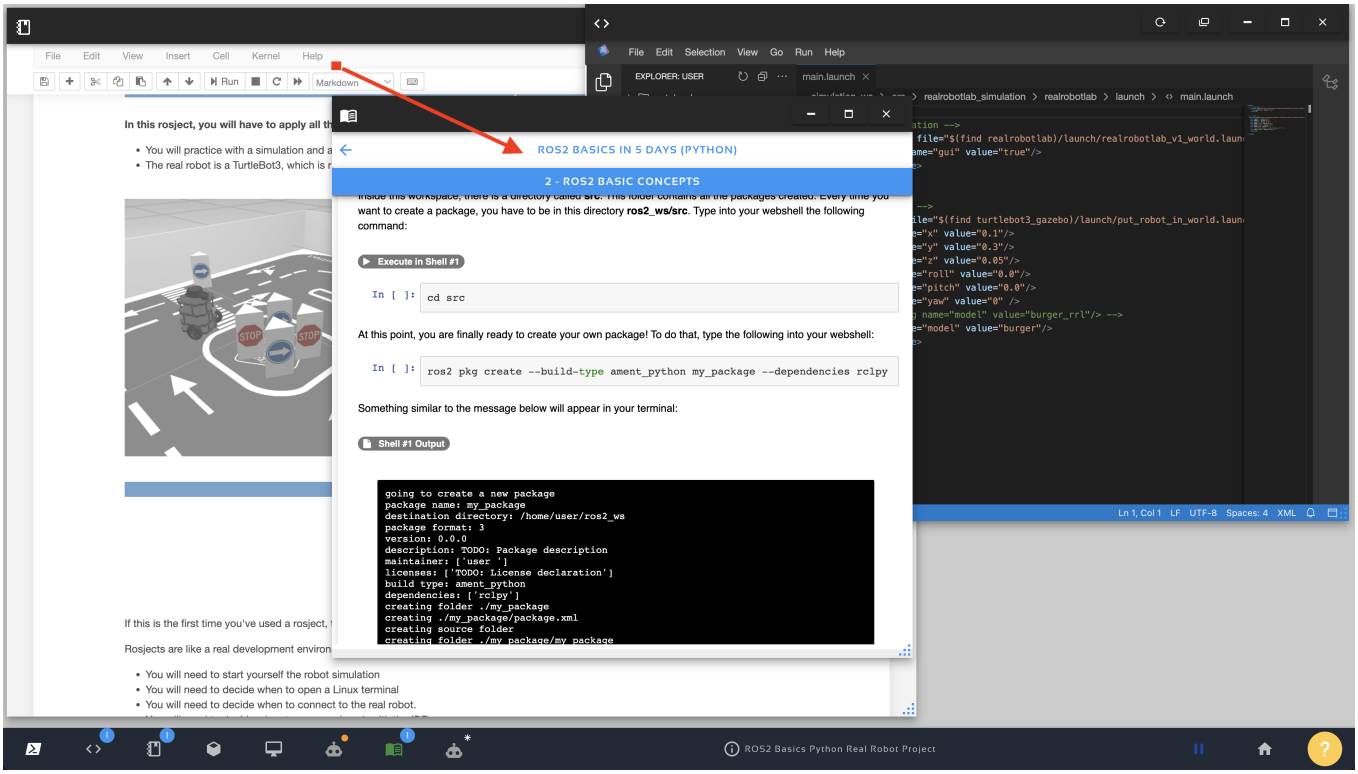


Check the Appendix below to learn **how to book a session with the real robot**.

How to check the course notebooks

In case you need to check something from the courses that you forgot, **YOU DON'T NEED TO SWITCH TO THE COURSE!!**. You can open any course content from within the rosject using the following icon.

Opens any notebook from all of our courses. Very useful if you want to quickly check something from the course without having to open it.



The project

This rosject is in three parts. Complete each section as directed in the **ROS2 Basics Python in 5 Days** course.

- **Section 1**: Practice for topic Publishers and Subscribers
- **Section 2**: Practice for Services
- **Section 3**: Practice for Actions

Before anything else, you will need to **launch the simulation** of the project. You will use the simulation to practice with the simulated environment.

0.1 Launching a simulation

Before working on your code, you need a simulation (or an actual robot) to be running. During the **ROS2 Basics** course, you saw the simulation running in the top right corner. The simulation was automatically launched.

In real development environments, you decide when to launch the simulation in order to start working.

Now, let's launch the simulation we will need for this project:

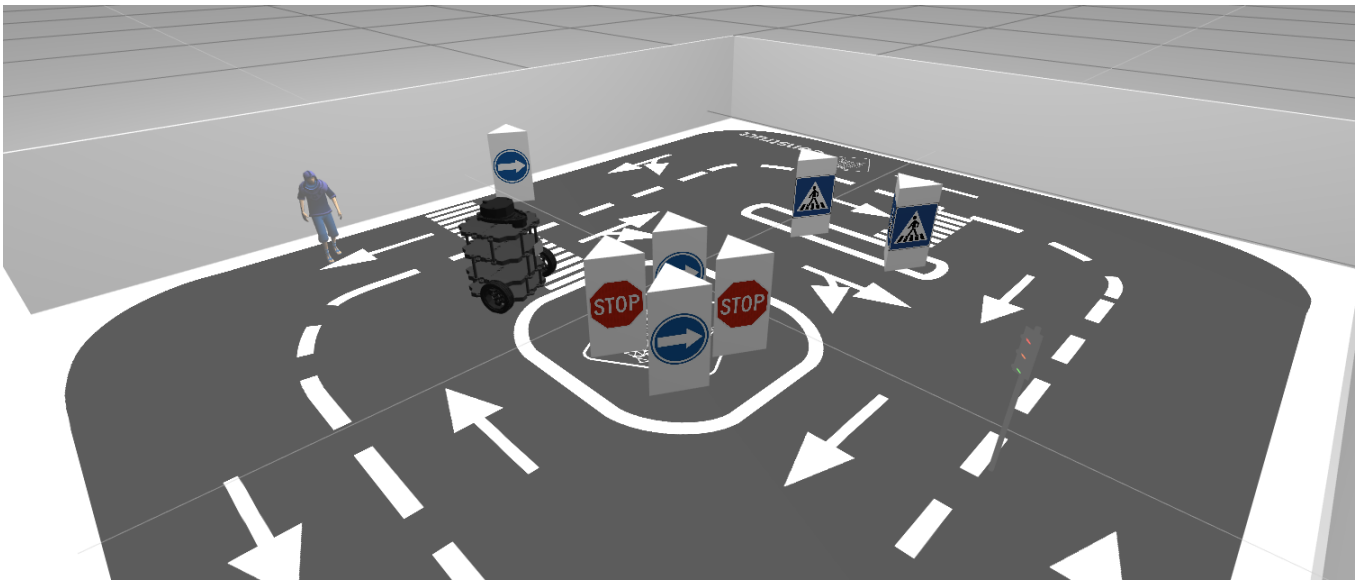
1. Open a terminal by pressing on the *terminal icon* at the bottom left croner of your screen.
2. On the opened terminal, type the following ROS command that launches the simulation

Entrée [ ]:

```
source /opt/ros/noetic/setup.bash
source ~/simulation_ws/devel/setup.bash
roslaunch realrobotlab main.launch
```

Wait approximately 30 seconds for the simulation to start. The simulation should automatically appear in a Gazebo window.

If it doesn't automatically appear, open yourself the Gazebo window, using the Gazebo icon:



IMPORTANT: the terminal that has the simulation running is occupied by it, so will not be able to use it unless you kill the simulation. So you will need to open other terminals for other purposes while keeping the simulation running

Launch the ros1_bridge

The simulation of Turtlebot3 runs on ROS1, but your programs will run on ROS2. Hence, you will need to launch also the `ros1_bridge` program to allow communication between them.

For the project we will use the `parameter_bridge` instead of the `dynamic_bridge` because the former allows to select only the topics we want to bridge. This allows important topics like `scan` to go through the bridge at a higher frequency.

ros1_bridge launch

1. Open another terminal by clicking the *terminal icon* at the bottom left of your screen and the PLUS icon

2. Then on that terminal, type the following commands to launch the bridge:

Entrée [ ]:

```
source ~/catkin_ws/devel/setup.bash
roslaunch load_params load_params.launch
source /opt/ros/foxy/setup.bash
ros2 run ros1_bridge parameter_bridge
```

You'll see the topic connections being established. From that moment, you can communicate with the topics offered by the simulation from ROS2.

Section 1 Topics

In this part of the project, you will practice using topics to control a robot. Your goal is to create a ROS program that makes the robot have a wall-following behavior.

1.1 Wall following behavior

Remember that you can check the ROS2 Basics course clicking on the notebooks icon in the bottom menu bar.

The wall following behavior is a behavior that makes the robot **follow along the wall on its right hand side**. This means that the robot must be moving forward at a 30cm distance from the wall, having the wall on its right hand side, the entire time.

To achieve this behavior in the robot, you need to do:

1. **Create a ROS2 package named `wall_follower`**
    - This package is the one that will contain the full project
    - Inside the package include a ROS Python file named `wall_following.py`
2. **Subscribe to the laser topic of the robot**
    - Inside the `wall_following.py` , you need to subscribe to the laser topic and capture the rays
    - In the callback of the subscriber, select the ray on the right (the one that makes a 90º angle to the right with the front of the robot) and use it to know the robot's distance to the wall

      NOTE: The topics for different simulations most likely will not be named the same. So make sure that you are using the correct name. For example, you used `/kobuki/laser/scan` during the course. The laser topic in this simulation is `/scan`
3. **Publish to the velocity topic of the robot**
    - Also inside the `wall_following.py` , create a publisher to the `/cmd_vel` topic that controls the wheels
    - At every step of the control loop, you need to publish the proper velocity command on that topic, based on the value of the distances detected by the laser:
        - If the distance to the wall is **bigger than 0.3m**, you need to make the robot approach the wall a little, by adding some rotational speed to the robot
        - If the distance to the wall is **smaller than 0.2m**, you need to move the robot away from the wall, by adding rotational speed in the opposite direction
        - If the distance to the wall is **between 0.2m and 0.3m**, just keep the robot moving forward

      IMPORTANT

      When the robot is moving along a wall, it can reach the next wall just infront of it. At that point in time, you should take into account how to progressively transition the robot from following the current wall to the next one.

To detect the wall in the front, we recommend that you use the laser ray just in the front of the robot. If the distance measured by that ray is shorter than 0.5m, then make the robot turn fast to the left (moving forward at the same time).

The result of this whole behavior must be that the robot moves along the whole environment (see video below).

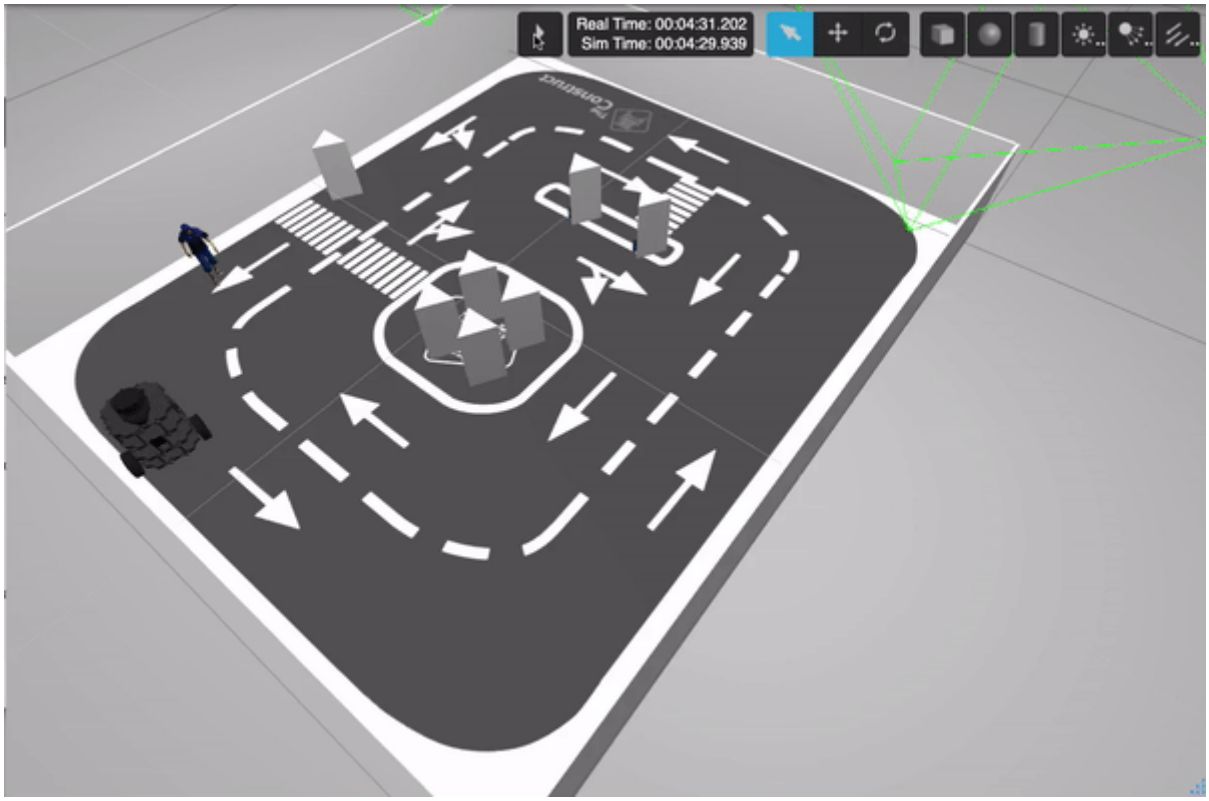
4. **Create a launch file named `start_wall_following.launch.py`**

1.2 Test your program in the simulation

Create the wall-following program and test it on the simulation.

If the program doesn't work on the simulation, it is 100% not going to work on the real robot.

Real-life robot development works this way: first, test your program in a simulation. When it works there, try it on a real robot. Never the other way around.



Follow the steps below to test your program in the simulation:

1. Make sure the simulation is launched, as explained above
2. In another terminal, launch the keyboard teleop with the following command:

Entrée [ ]:

```
source /opt/ros/noetic/setup.bash
source ~/simulation_ws/devel/setup.bash
rosrun turtlebot3_teleop turtlebot3_teleop_key
```

3. Now use the keys to move the robot to a convenient position to test your ROS program. This means, close to a wall, and with the wall on the robot right hand side

IMPORTANT-1: Remember that in order to be able to move the robot with the keys, **the terminal where you launched the teleop has to have the focus**.

IMPORTANT-2: close this *teleop* program once you have the proper position. Otherwise, it will interfere with your program.

4. Now launch your ROS program and see the results. If the robot doesn't behave correctly, debug your program and try to figure out the reason.

1.3 If program works in simulation, then go to the real robot

Once the robot is moving along the walls of the simulated environment, it is time for you to test it on the real robot.

The steps are:

1. Book a date and time in the *Real Robot Lab* to test your program (see Appendix below that teaches how to do that).
2. On the day and time selected, open this rosject
3. Connect to the real robot lab from within this rosject (see Appendix below)
4. Use the web-joystick that appears on the cameras window and move the real robot to the proper position to start before you launch your program
5. You don't need to launch the simulation, because you are using the real thing
6. **IMPORTANT**: you need to launch the `ros1_bridge` because the real Turtlebot3 also runs ROS1. Unless you launch it, your program will not be able to talk to the robot
7. Launch your program and see if it works properly. Chances are it will not because simulation is not an exact copy of the real environment. Now is your time to debug
8. Test on the real robot and debug your errors until you have it working as intended

Section 2 Services

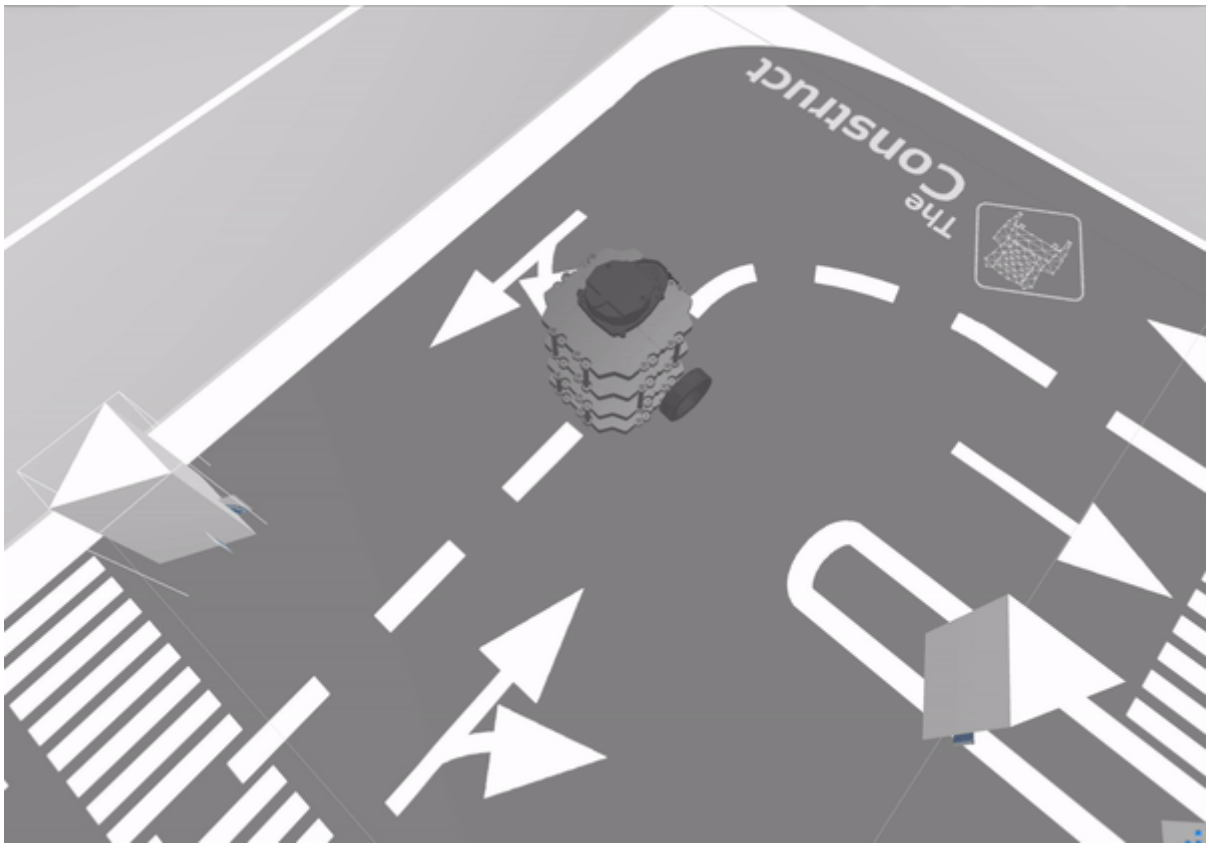
In the previous section, you used the keyboard to put the robot at a proper position so it can do the follow wall behavior.

In this section you will create a program that does that alignment automatically.

The goal in this section is to create a new program that when called,

- Makes robot search for the nearest wall
- Once found, the robot will approach to the proper distance (0.3 m)
- And then aligns itself with the wall on its right hand side, waiting in that position, ready to do the *wall following behavior*

Check this video, to understand the behavior that is required:



Robot behavior required

The robot of the video above does the following:

1. Because of the 360º laser, the robot can detect the closest wall (shortest laser ray)
2. Once identified the closest wall, the robot rotates to face that wall
3. Once the robot is facing that wall, it moves closer to it, until the robot is at 0.3 m of it
4. Then, the robot rotates again so it ends with the wall to its right hand side

We will call that behavior, the *find wall behavior*.

The *find wall behavior* must be **provided by a service server**. What does it mean?

- You have to create a new node with a service server called `find_wall`
- When somebody calls that server, the server *callback function* must execute the code of the *find wall behavior*, and make the robot find the closest wall and align with it

This server must be called before starting the *wall following behavior*.

To accomplish all that, you will have to do three things:

1. Create a service server node that, when called, will make the robot do the *find wall behavior*
2. Modify the program from Section 1 so that it includes now a service client that calls the service server **before** starting the *wall following behavior*
3. Create a new launch file to launch both nodes (*wall following node* and the node with the `find_wall` service server)

LET'S GOT FOR IT!

2.1 Create Service Server

- Inside the `wall_follower` package create a new ROS file named `wall_finder.py`
- Inside `wall_finder.py` create a service server named `find_wall`
  - The callback of the server must contain the code that performs the *find wall behavior*

    Some tips for that

    1. Identify which laser ray is the shortest one. Assume that this is the one pointing to a wall
    2. Rotate the robot until the front of it is facing the wall. This can be done by publishing an angular velocity until front ray is the smallest one
    3. Move the robot forward until the front ray is smaller than 30cm
    4. Now rotate the robot again until ray number 270 of the laser ranges is pointing to the wall
    5. At this point, consider that the robot is ready to start following the wall
    6. Return the service message with `True`

- The server must use a new `FindWall` interface that you must also create, and that has the following structure:

Entrée [ ]:

```
FindWall.srv

---
bool wallfound
```

Check sections 4.6 and 4.7 to remember how to do this part.

- Create a launch file named `start_wall_finder.launch.py` that launches the `wall_finder.py`

Test Service Server

How can you test and debug the service server independently of the code of Part 1? By doing the following steps:

1. Launch the service server node using `start_wall_finder.launch.py`
2. Put the simulated robot in a proper location for your test. You can do that using the `keyboard_teleop` explained in the previous section
3. Call the service from a terminal by issuing the command:

Entrée [ ]:

```
ros2 service call /find_wall std_srvs/Empty '{}'
```

This should start the robot's movement looking for the wall. If it doesn't, it means your program doesn't work properly. You need to debug!

2.2 Add server call to Section 1 program

Once the service is working, you need to call it from the `wall_following.py` program that you created in section 1.

The idea is that before the *wall following behavior* starts following the wall, the robot has to be already aligned to the wall. Hence, the `wall_following.py` must call the `/find_wall` service before starting doing the *wall following behavior*.

So you need to do the following:

1. Add to the `wall_following.py` node a synchronous service client that connects to the `/find_wall` service
2. Use that client to call the service before the control loop of that node, so the robot gets prepared autonomously before following the wall

2.3 Create new launch file

Now you need to create a launch file that launched both the `wall_following.py` and the `wall_finder.py` , because both of them have to be running in order to work together.

1. Create a new launch file named `main.launch.py` that launches both nodes: first the service server node and then the wall following node
2. Put the robot in the simulation at a proper location for testing
3. Then call the `main.launch.py` to start the full program

This should start the robot movement (first the robot looks for the wall, then it starts to follow it).

IMPORTANT: Test everything on the simulation. When it works in the simulation, book a date in the *Real Robot Lab* and make it work there, too. Same as in Section 1.

Section 3 Actions

Create an action server that records odometry

The goal of this section is to **create an action server that records the robot's odometry while the robot is moving along the wall**, so we can later use it to see the path followed by the robot.

To achieve that, you will need to create another node that contains an action server in charge of doing the odometry recording when the action server is activated.

The steps required to achieve that are:

1. Create an action server that, when called, starts recording odometry
2. Add a call to the action server from the wall following node
3. Include the launch of the action server in `main.launch.py`

3.1 Create action server

You need to do the following things:

- The server must start recording the (x, y, theta) odometry of the robot as a Point32, one measure every second
- As feedback, the action server must provide the total amount of meters that the robot has moved so far
- When the robot has done a complete lap, it has to finish and return the list of odometries recorded

To achieve that, make the following steps:

1. Create a new ROS node in a new Python file named `odom_recorder.py`
2. `odom_recorder.py` must have a subscription to the `/odom` topic. The callback of the subscriber must:
    - Get the last odometry value received and extract the `x`, `y`
    - Save those values on a `geometry_msgs/Point` variable of the class named `self.last_odom`
3. Include in `odom_recorder.py` an action server named `/record_odom`
4. Inside the callback of the action server, you need to do the following:
    - Get the first odometry value received and store in a variable named `self.first_odom`
    - Then every second, get the last odometry value of `self.last_odom` and store in an array of `Point` values named `self.odom_record`
5. Inside the Feedback callback, you need to do the following:
    - Have a variable with current total distance named `self.total_distance`

- Have another variable with last (x,y) position, named `self.last_x` and `self.last_y`
- At every step of the feedback, compute the distance between (`self.last_x` , `self.last_y`) and (`self.last_odom.x` , `self.last_odom.y`)
- Then add that distance to `self.total_distance`
- Return as feedback the current value of `self.total_distance`

6. At every step, check that the distance between (`self.last_odom.x` , `self.last_odom.y`) and (`self.first_odom.x` , `self.first_odom.y`) is less than 5 cm
7. If less than 5 cm, then it means that the robot has reached the initial position again, so the action server must end and return as result the total list of (x,y) points that were stored in the `self.odom_record` array
8. This server uses an `OdomRecord.action` message that you must also create:

Entrée [ ]:

```
---
geometry_msgs/msg/Point[] list_of_odoms
---
float32 current_total
```

3.2 Add action service client to Section 1 program

Once the action server is working, you need to call it from the `wall_following.py` program that you created in section 1.

The idea is that before the *wall following behavior* starts following the wall, the `wall_following.py` must call the `/record_odom` action to indicate that the recording of the odometry must start.

So you need to do the following:

1. Add to the `wall_following.py` node an action client that connects to the `/record_odom` action server
2. Use that client to call the action before the control loop of that node, so it starts recording the odometry before following the wall

3.3 Add action server to main launch file

Now you need to modify the `main.launch.py` to include the launch of the `odom_recorder.py` .

1. Modify the `main.launch.py` launch file to launch the three nodes: first the service server node, then the action server node, and then the wall following node
2. Put the robot in the simulation at a proper location for testing
3. Then call the `main.launch.py` to start the full program
4. After the robot starts moving you should see the feedback appearing

IMPORTANT: Test everything on the simulation. When it works in the simulation, book a date in the *Real Robot Lab* and make it work there, too. Same as in Section 1 and 2.

Section 4 Book a date to present your project on YouTube (optional)

An important part of your learning is to teach others what you have learned.

To get the most out of your learning, you should present your project.

If you're ready to take this brave step, contact The Construct at info@theconstructsim.com (mailto:info@theconstructsim.com) and request a presentation. We can agree on a day and time.

In the presentation, **you will be asked to do the following**:

1. Prepare your presentation. Explain your project in 20 minutes: How did you solve your project, and how does it work?
2. On the agreed day, have a camera and mic ready
3. The Construct will contact you on the day and prepare the whole broadcast. We handle the technical details. You only need to be ready to present

The event will broadcast on YouTube, so anybody around the world can attend and watch your presentation.

It's important to remark that, although this step is optional, it is mandatory in order to get the certificate of the course.

Without successfully completing the presentation, you won't be able to earn the course certificate.

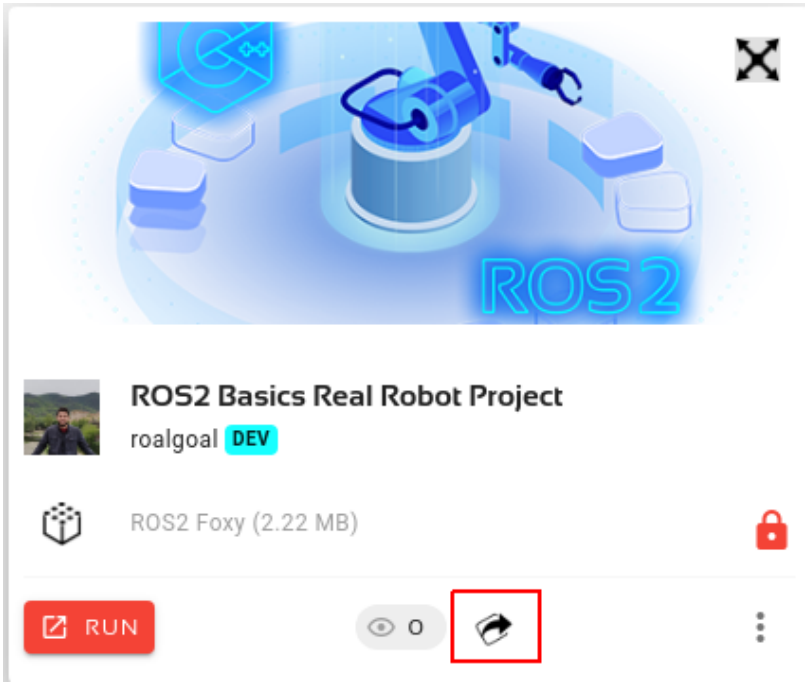
Section 5 Share your project on LinkedIn or social networks

The rosject that you created is a demonstration of your skill and value as a ROS Developer.

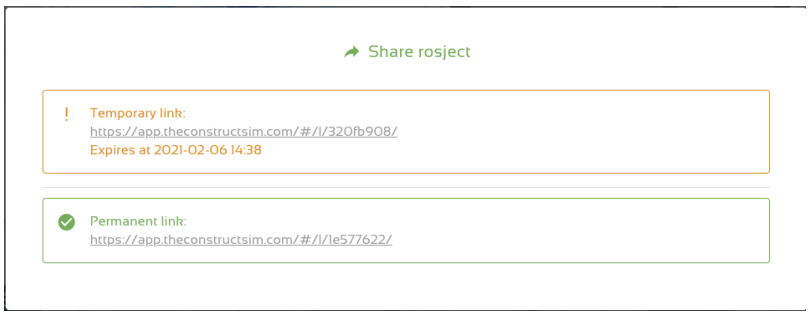
You can share your rosject on any website to show the ROS projects you have worked on. Examples of your capability can encourage potential employers to contact you.

To share your rosject, follow the steps below:

- Go to your list of rosjects
- Click on your rosjects Share button to get the share link



- Publish the link anywhere. Remember to use the *Permanent link*.



Appendix How to connect to the real robot

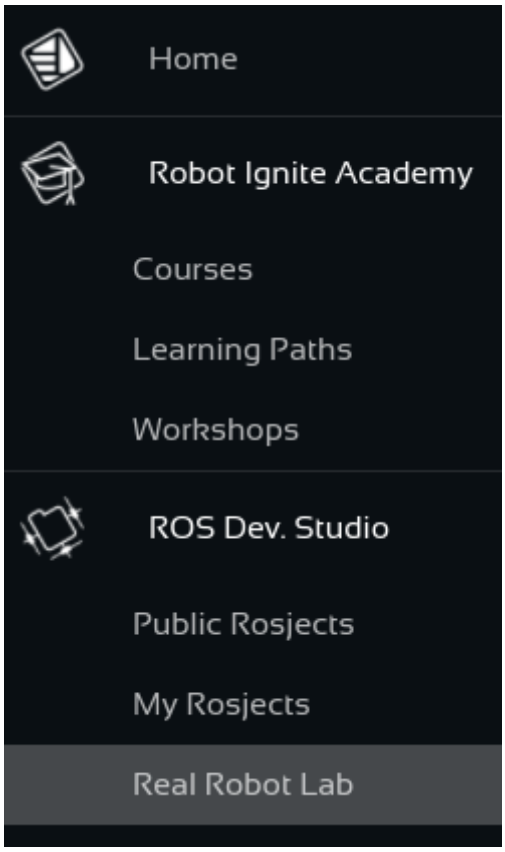
Once you know the basics of the operation with a simulated version, it's time to practice what you've learned with a TurtleBot3 robot.

Book up to two 25-minute sessions per week

To make a booking, follow these simple steps:

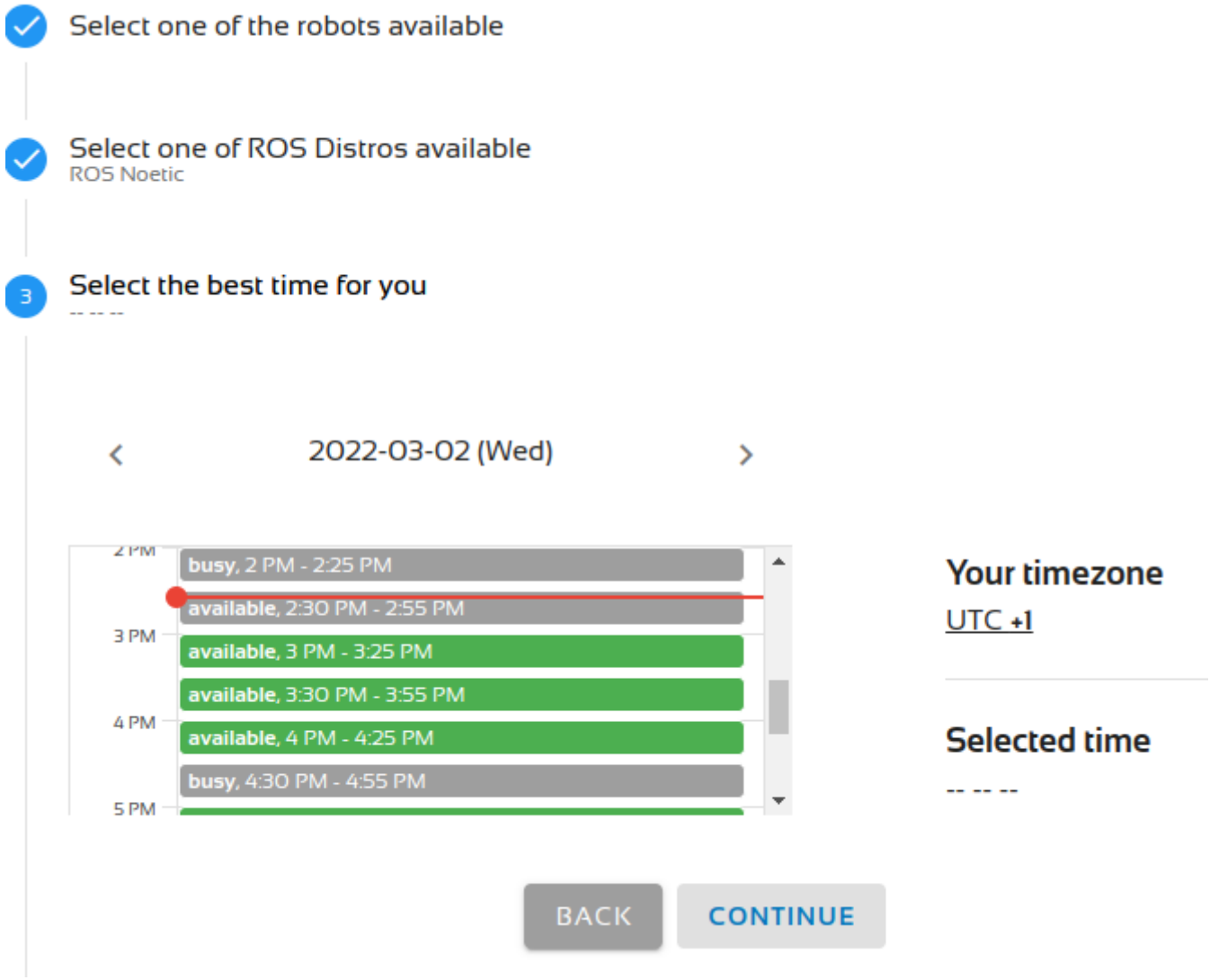
Step 1 Book a Session

From The Construct's main dashboard, book a session by clicking on the icon seen below:



A booking page will appear, where you can make a new booking.

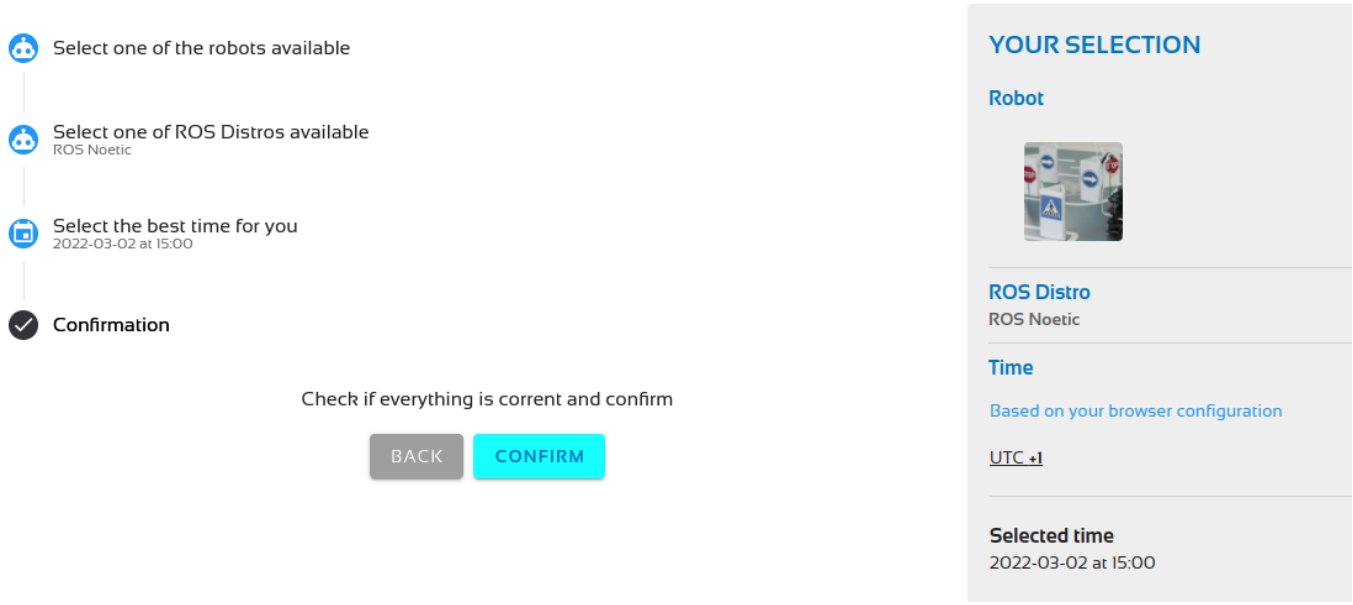
You can either click on `Book a robot` on the bottom and then select the Turtlebot, or just click on `Book now` where the image of the turtlebot is and select the date and time for your reservation:



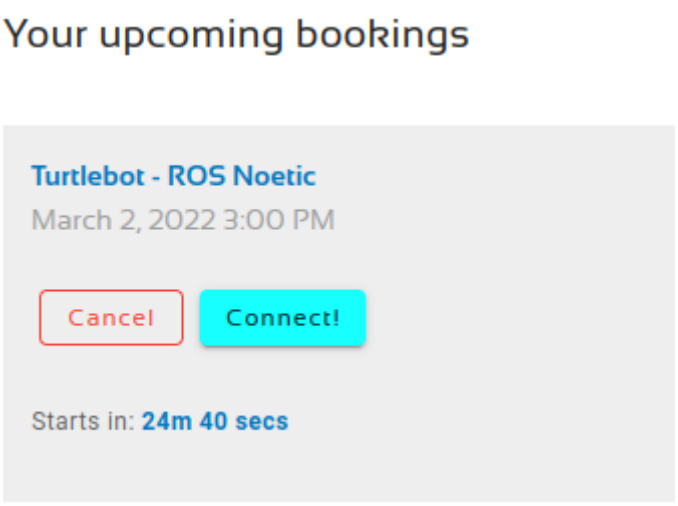
Only the **available dates and times** are shown in green. They come in **25-minute blocks**.

There is also a limitation on the number of bookings per week a user can make.

Your number of available bookings depends on your license and subscription.

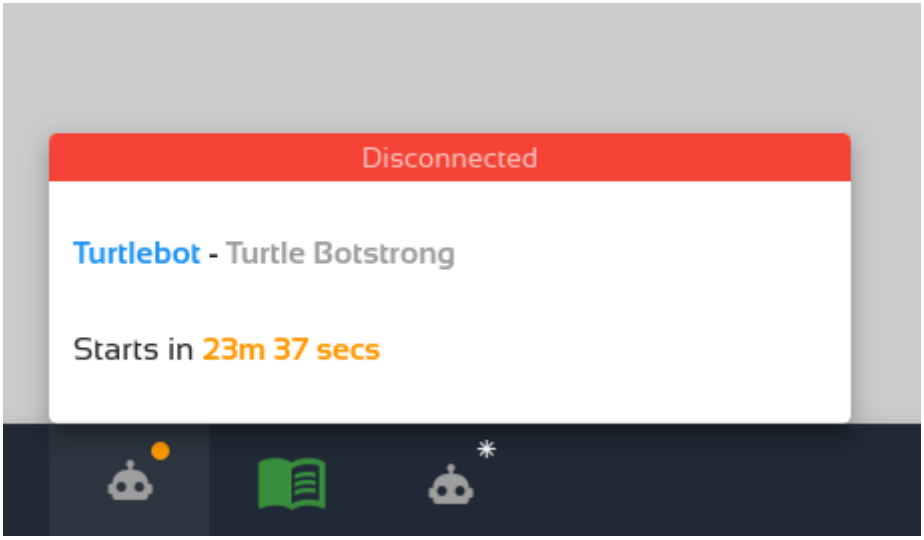


- At any point in time you can check for bookings you have:



Step 2 Open project rosject

Before the time of your booking, open your rosject. You will see the robot icon at the bottom bar has an orange dot. It means that your time to connect to the roobt will start soon. If you click on it you will see a countdown to your session start.



The orange dot will turn red when your booked session has started.

Step 3 Turn ON robot connection

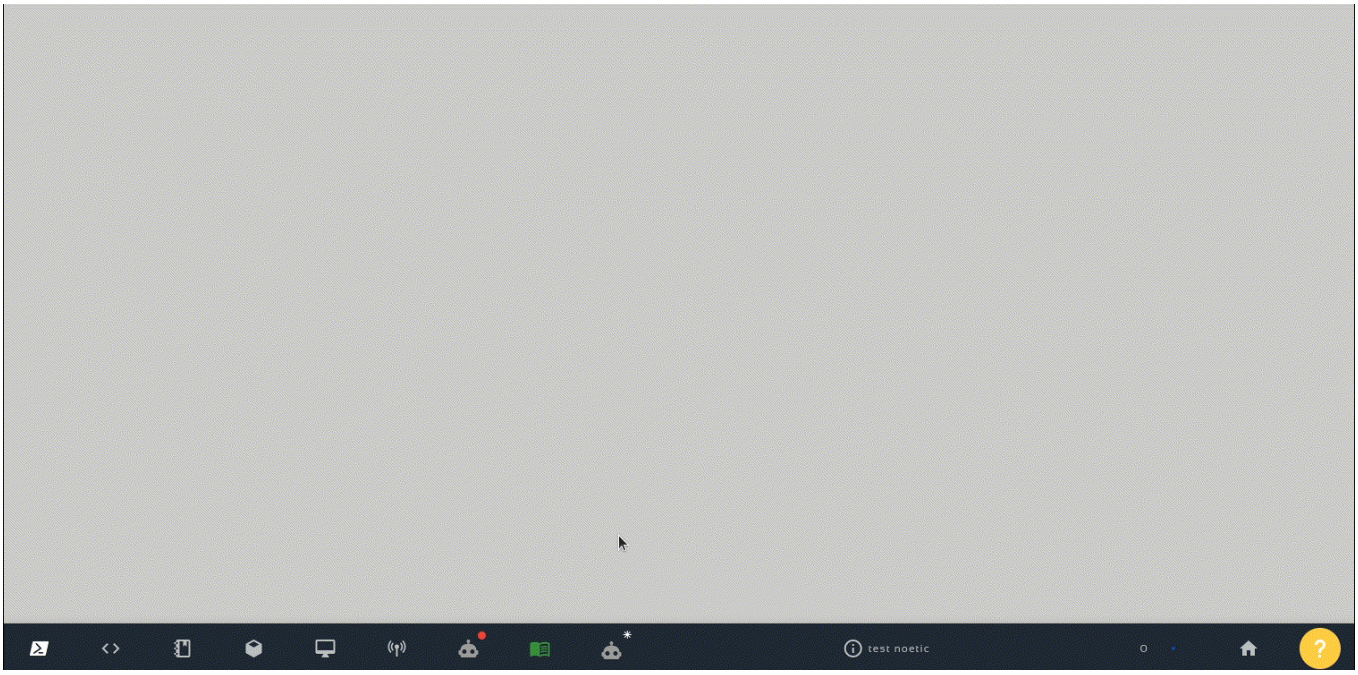
Once you are inside the rosject and your real robot session has started, click on the robot icon at the bottom. A popup with a toggle for **turning on the connection** will appear.

WARNING: Nothing will appear unless you have a booking. So if you didn't make a booking in the dashboard, now is the time to do it.

Now you just have to click on the toggle to connect the desktop environment to the remote lab. This will start the connection process, which lasts for a few seconds.

Once the connection process is finished, the **red dot will change to green** if succesfully connected. Now **any terminal you open will show the topics and services of the real robot**.

Move the joystick around to confirm you are connected to the robot:



You can perform the following command, generating a list of topics similar to the list below. The first time you issue a ROS command, it could take an extra 30 seconds to appear:

Execute in WebShell #1

Entrée [ ]:

```
rostopic list
```

Entrée [ ]:

```
/battery_state
/cmd_vel
/cmd_vel_rc100
/diagnostics
/firmware_version
/imu
/joint_states
/magnetic_field
/motor_power
/odom
/reset
/rosout
/rosout_agg
/rpms
/scan
/sensor_state
/sound
/tf
/tf_static
/version_info
```

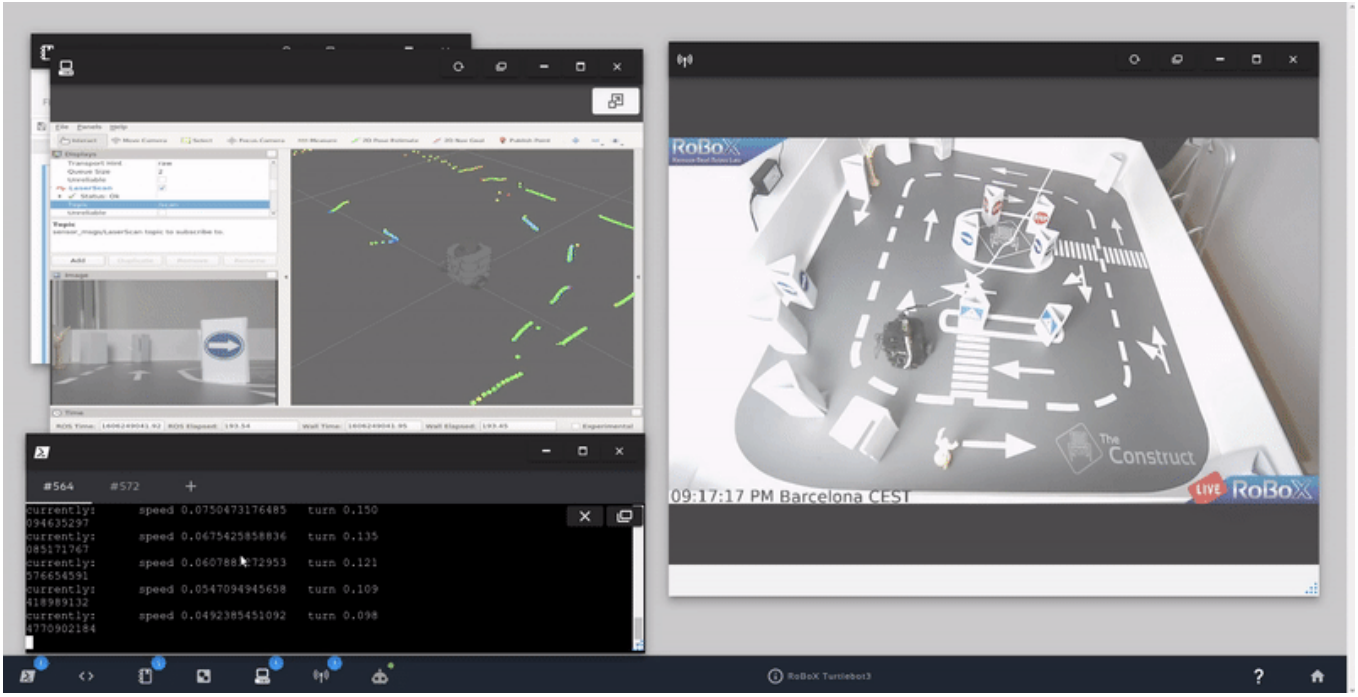
You are now connected to the robot! Try moving the robot around to see the lasers and the camera in RVIZ.

Move the robot around: Be careful **NOT to exceed a linear speed of 0.19 and an angular speed of 0.49**. Otherwise, the node will be terminated for security reasons.

Entrée [ ]:

```
rviz
```

Now, add an image and scan elements in the **base_link** frame. You should receive a result similar to the following:



Ready for your programs!

If you have reached this point, it means that you have a proper connection with the remote real robot lab.
Now you can launch the programs of your project and see the results on a real robot.