Welcome to the UJI Autonomous Mobile Robots Online Course

In this course you are going to explore and learn with a simulator of the Pioneer P3-DX mobile robot (http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx).

During the first week, you will learn how a mobile robot moves, and how to program basic functions for sending commands to the motors, and calculating the distance travelled by the robot.

Week 1

- Hello World! (Hello%20World.ipynb)
- Moving the Robot (Moving%20the%20Robot.ipynb)
- Motion Functions (Motion%20Functions.ipynb)
- Encoders (Encoders.ipynb)
- Angles and Distances (Angles%20and%20Distances.ipynb)
- Exercises:
    1. Move the robot forward for a given distance (exercises/Distance.ipynb)
    2. Turn the robot right for a given angle (exercises/Angle.ipynb)
    3. Square Test (exercises/Square.ipynb)
    4. Robot Speed (exercises/Robot%20Speed.ipynb)

(https://www.python.org)

Hello World!

In this course, you are going to program a mobile robot, using a language called Python (https://www.python.org). If you have never used it, but you are familiar with another language (C, Java), you will learn the basics very quickly.

(http://jupyter.org/) The programs will run directly in the browser, in an environment called Jupyter Notebook (http://jupyter.org/). How? Quite simple: notebooks consist of cells, which can be either text or code.

Text cells (like this one) are written in *markdown*, a lightweight markup language that is converted to HTML. Such cells can embed images, links, and mathematical expressions. They are used for providing you with information, explanations and indications about the course.

The **code cells** are plain Python statements that can be executed with ***Shift+Enter***. Try it in the following cell:

In [ ]:
```
# this is a code cell
# click on it, then press Shift+Enter
print('Hello World!')
```

If the "Hello World!" message has been displayed as the output of the cell, congratulations! everything is working fine :-)

Code cells can be interactively edited and re-run. Try it in the previous cell: for example, change the message in the `print` statement. Just click on the cell, edit the text, and re-run it again by pressing `Shift+Enter`.

For the programming tasks in this course, you need to know the basics of structured programming:

- Variables
- Data structures (arrays, lists)
- Control flow (conditional statements, loops)
- Functions

For a quick introduction to these topics in Python, you may browse through The Python Tutorial (https://docs.python.org/2.7/tutorial/).

For an overview of Jupyter notebooks, please watch this nice tutorial by Corey Schafer (https://www.youtube.com/watch?v=HW29067qVWk&t=407s).

Let's proceed now to the next notebook: Moving the Robot (Moving%20the%20Robot.ipynb)

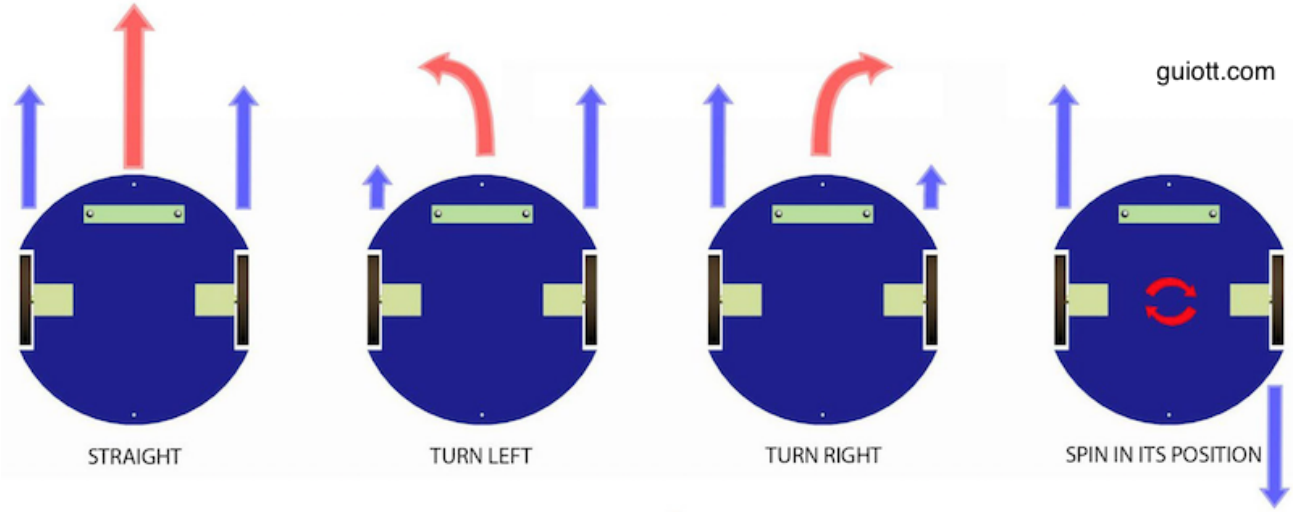
Moving the Robot

The Pioneer 3-DX robot is an all-purpose base, used for research and applications involving mapping, teleoperation, localization, monitoring and other behaviors.

It is a so-called ***differential-drive*** mobile platform (https://en.wikipedia.org/wiki/Differential_wheeled_robot), with a powered wheel on either side of the robot body, and a rear castor wheel for balance.

Each wheel is powered by its own motor. The motion of the robot is determined by the speed on the wheels:

- If both wheels are driven at the same direction and speed, the robot will move in a straight line.
- If one speed is higher than the other one, the robot will turn towards the direction of the lower speed.
- If both wheels are turned with equal speed in opposite directions, the robot will spin around the central point of its axis.



(http://www.guiott.com/CleaningRobot/C-Motion/Motion.htm)

Let's see a Pioneer robot moving!

In [ ]:
```
# this is code cell -> click on it, then press Shift+Enter
from IPython.display import YouTubeVideo
YouTubeVideo('vasBnRS3tQk')
```

Initialization

Throughout the course, some code is already written for you, and organized in modules called *packages*. The cell below is an initialization step that must be called at the beginning of each notebook. It can take a few seconds to run, so please be patient and wait until the running indicator `In[*]` becomes `In[2]`.

In [ ]:
```
# this is another code cell -> click on it, then press Shift+Enter
import packages.initialization
import pioneer3dx as p3dx
p3dx.init()
```

Motion

Let's move the robot on the simulator!

You are going to use a *widget*, a Graphical User Interface (GUI) with two sliders for moving the robot in two ways: translation and rotation.

In [ ]:
```
# and this is again a code cell -> you already know what to do, don't yo
u?
import motion_widget
```

The cell above outputs two sliders, which control the translation and rotation of the robot. Initially both values are zero; move the slider left or right to change their values and move the robot.

Once you are familiar with the motion of the robot, please proceed to the next notebook: Motion Functions (Motion%20Functions.ipynb).

Motion Functions

In this notebook you will learn how use Python functions for moving the robot in your programs.

First, the initialization step needs to be executed, because each notebook is a program that is running separately.

In [ ]:
```
# click on this cell and press Shift+Enter
import packages.initialization
import pioneer3dx as p3dx
p3dx.init()
```

Functions

There are three functions for controlling the motion of the robot:

- `p3dx.move(ls,rs)`
- `p3dx.stop()`
- `p3dx.sleep(t)`

The main motion function is:

```
p3dx.move(ls, rs)
```

where

```
ls : left wheel speed (rad/s)
rs : right wheel speed (rad/s)
```

The function sets the speeds of the wheels, and the robot moves until blocked by an obstacle, or a new speed is set, or it is stopped with the function:

```
p3dx.stop()
```

For controlling the amount of time that the robot moves, there is the function:

```
p3dx.sleep(t)
```

where `t` is the number of seconds. During that pause, the program sleeps but the robot keeps moving with the last speed set.

Example

Use the code below for moving the robot. Feel free to explore with different speed and time values.

In [ ]:
```
# Move forward
p3dx.move(2.5,2.5)
p3dx.sleep(1)
p3dx.stop()
```

In [ ]:
```
# Move backward
p3dx.move(-2.5,-2.5)
p3dx.sleep(1)
p3dx.stop()
```

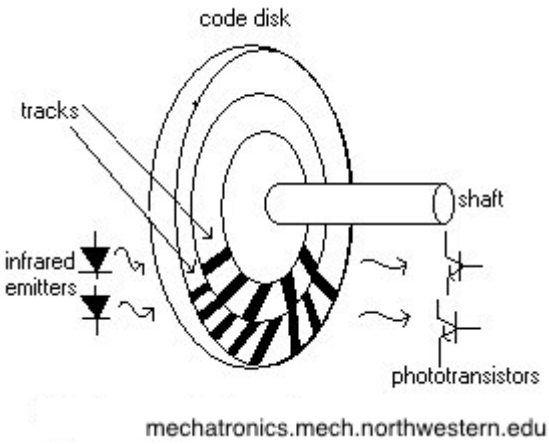
In [ ]:
```
# Turn left
p3dx.move(-2.5,2.5)
p3dx.sleep(1)
p3dx.stop()
```

In [ ]:
```
# Turn Right
p3dx.move(2.5,-2.5)
p3dx.sleep(1)
p3dx.stop()
```

You can also copy and paste the functions several times with different values for a composition of motions:

In [ ]:
```
# Your own wonderful motion sequence:
# Replace the dots below with a sequence of motion commands
...
```

For a better control of the motion, we are going to introduce the first sensors in our mobile robot: the encoders (Encoders.ipynb).



(http://mechatronics.mech.northwestern.edu/design_ref/sensors/encoders.html)

Encoders

For computing the distance traveled by a mobile robot, we are going to use sensors that mesure the motion of the wheels, i.e. the angle turned by each wheel.

Optical encoders are very popular in mobile robotics for measuring the position within a motor drive of at the shaft of a wheel. They consist of a source of light, a pattern disc, and a light detector. As the disc turns, light pulses are converted to a binary signal, which is counted and as a result an angle is produced.

First, as usual, let's initialize the robot.

In [ ]:
```
import packages.initialization
import pioneer3dx as p3dx
p3dx.init()
```

Now, a GUI widget will allow you to control the speed of **each wheel** independently, and display the value (in radians) of the angle turned by each wheel.

In [ ]:
```
import encoder_widget
```

The values of the encoders are stored in these variables:

In [ ]:
```
p3dx.leftEncoder
```

In [ ]:
```
p3dx.rightEncoder
```

You can use them in your programs; for example, the following code displays their values more prettily.

In [ ]:
```
print(" Left Encoder: %7.3f radians" % p3dx.leftEncoder)
print("Right Encoder: %7.3f radians" % p3dx.rightEncoder)
```

Radians and Degrees

Not familiar with radians (https://en.wikipedia.org/wiki/Radian)? Angles can be converted from radians to degrees with some simple math:

$$degrees = radians\frac{180}{\pi}$$

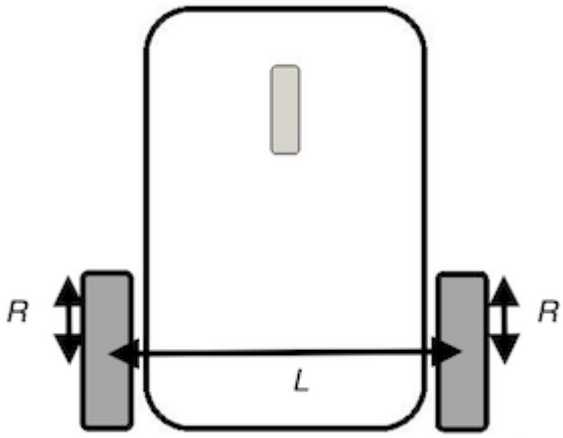
Use the code below, and replace the dots in the first line with the value in radians that you want to convert, then run the cell.

In [ ]:
```
radians = ...
import math
degrees = radians * 180 / math.pi
print("%.3f radians are equivalent to %.3f degrees" % (radians,
degrees))
```

However, radians will be widely used as angular magnitude, since they result in simpler mathematical expressions.

For the proper use of encoders, it's time for a bit of geometry in the next notebook: Angles and Distances (Angles%20and%20Distances.ipynb)

Angles and Distances

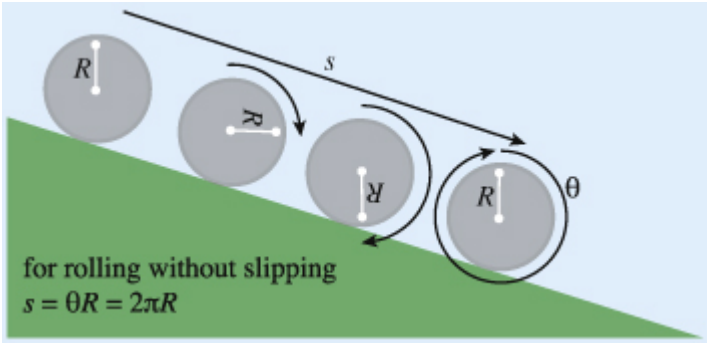


In a differential-drive mobile robot, there are two geometrical parameters that determine the motion of the robot:

- the radius of the wheels
- the length of the wheel axis, i.e. the distance between the centers of the wheels

With some simple operations, we can calculate the motion of the robots for two particular cases: moving straight and turning in place.

Distance traveled in a straight motion



(http://iopscience.iop.org/article/10.1088/0031-9120/47/2/189) When the robot moves forward, both wheels are turning at the same speed. For a complete turn of the wheels, the traveled distance s is equal to the length of the circumference:

$$s = 2\pi R$$

In the general case, if the wheels turn an angle θ (in radians), the traveled distance is:

$$s = \theta R$$

Angle turned in an in-place rotation

When the robot spins around its own axis, both wheels are turning at the same speed, but in opposite directions.

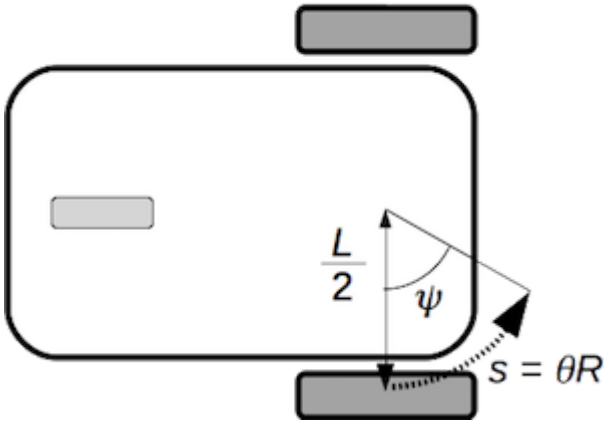
In this case, the distance traveled by the wheel must be equal to the length of the arc turned by the robot around its axis, whose radius is $L/2$:

$$s = \theta R = \psi \frac{L}{2}$$

The turned angle can thus be determined as:

$$\psi = \frac{2\theta R}{L}$$

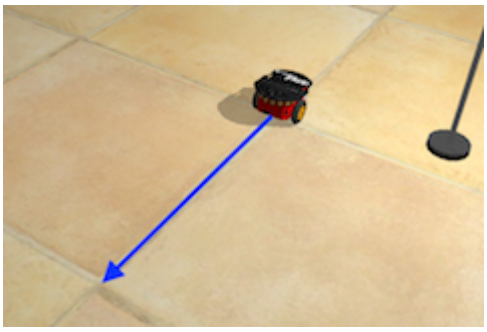
Got it? Let's proceed then with the first exercises of this course!



Exercises

1. Move the robot forward for a given distance (exercises/Distance.ipynb)
2. Turn the robot right for a given angle (exercises/Angle.ipynb)
3. Square Test (exercises/Square.ipynb)
4. Robot Speed (exercises/Robot%20Speed.ipynb)

Exercise: Move the robot forward for a distance.



You are going to program the robot for moving forward from the initial position at the start of the simulation, in the center of the room.

The robot should stop after traveling a distance equal to the size of the tile, which is 2 m.

According to the specifications, the *diameter* of the wheels of the Pioneer 3DX robot is 195.3 mm.

1. Starting position

For a better visual understanding of the task, it is recommended that the robot starts at the center of the room.

You can easily relocate the robot there by simply restarting the simulation, by clicking on the second icon of the button bar, as depicted in the figure.

2. Initialization

After restarting the simulation, the robot needs to be initialized.

In [ ]:
```
import packages.initialization
import pioneer3dx as p3dx
p3dx.init()
```

3. Motion code

In the following program template, you must fill the gaps with the appropriate code.

The idea is to compute the turned angle of one of the encoders (no matter which one) as the difference between the current and initial values, then compute the traveled distance. The robot will keep moving as long as this distance is lower than the target distance, then it will stop.

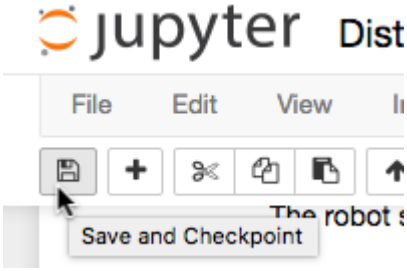
In [ ]:
```
target = 2          # target distance
radius = ...        # wheel radius
initialEncoder = ...
distance = 0
while distance < target:
    p3dx.move(2.5,2.5)
    angle = ...
    distance = ...
p3dx.stop()
```

Did the robot stop at the right place?

If not, you need to modify your code, reload the simulation, and run the code again.

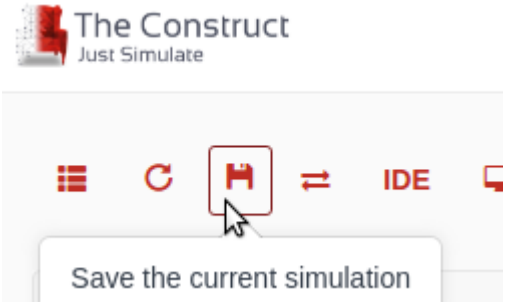
If yes, please proceed to the next exercise, but before, let's do something important: **save your work!**

Save your work!!!



For saving your progress, you need to save your progress **twice**:

- **First**: you must save the notebook with the button labeled as **Save and Checkpoint** (or its equivalent option in the File menu).

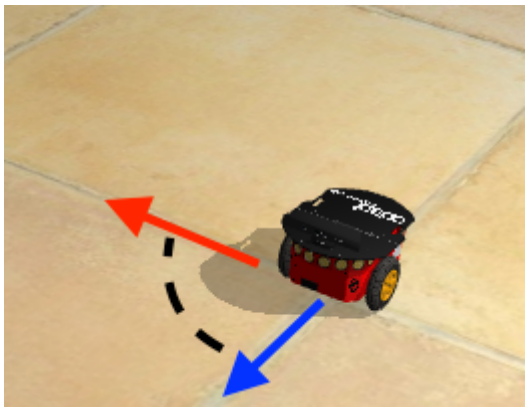


- **Second**: you must save the simulation environment with the button labeled as **Save the current simulation** in the simulation window. Otherwise, your changes will not be stored in the cloud.

In addition, for extra safety, you can download a copy of your notebook to your local computer with the menu **File / Download as / iPython Notebook (ipynb)**.

If you have already saved your work, it's time to proceed to the next exercise: Turn the robot right for a given angle (Angle.ipynb)

Exercise: Turn the robot for an angle.



You are going to make a program for turning the robot from the initial position at the start of the simulation, in the center of the room.

The robot should stop after turning 90 degrees.

According to the specifications, the *diameter* of the wheels of the Pioneer 3DX robot is 195.3 mm and the distance between the wheels is 330 mm.

1. Starting position

For a better visual understanding of the task, it is recommended that the robot starts at the center of the room.

You can easily relocate the robot there by simply restarting the simulation, by clicking on the second icon of the button bar, as depicted in the figure.

2. Initialization

After restarting the simulation, the robot needs to be initialized.

In [ ]:
```
import packages.initialization
import pioneer3dx as p3dx
p3dx.init()
```

3. Motion code

In the following program template, you must fill the gaps with the appropriate code.

The idea is to compute the turned angle of one of the encoders (no matter which one) as the difference between the current and initial values, then compute the angle turned by the robot. The robot will turn as long as this second angle is lower than the target angle, then it will stop.

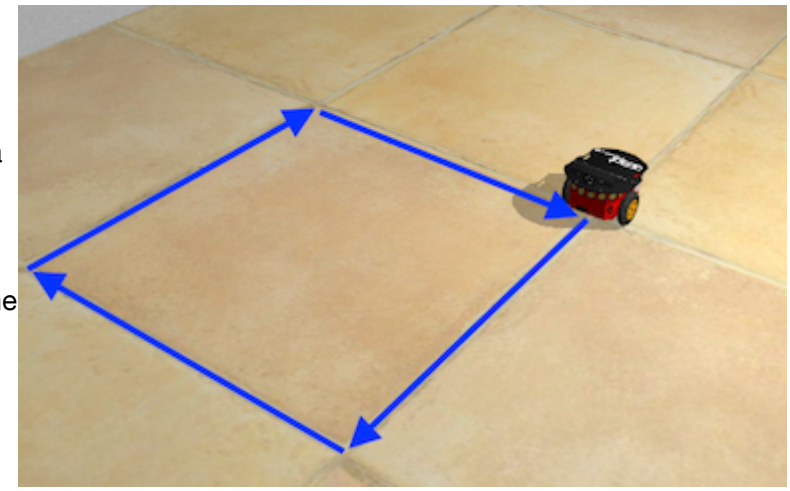
In [ ]:
```
target = ...      # target angle in radians
r = ...           # wheel radius
L = ...           # axis length
initialEncoder = ...
robotAngle = 0
while robotAngle < target:
    p3dx.move(1.0,-1.0)
    wheelAngle = ...
    robotAngle = ...
p3dx.stop()
```

Gentle reminder: save your work before proceeding to the next exercise!

Instructions for saving (Distance.ipynb#save).

Next exercise: Square Test (Square.ipynb)

Exercise: Square Test.



You are going to make a program for describing a square trajectory with the robot.

Instead of starting to code from scratch, you are going to reuse the code that you developed for the distance and turning exercises.

1. Starting position

For a better visual understanding of the task, it is recommended that the robot starts at the center of the room.

You can easily relocate the robot there by simply restarting the simulation, by clicking on the second icon of the button bar, as depicted in the figure.

2. Initialization

After restarting the simulation, the robot needs to be initialized.

In [ ]:
```
import packages.initialization
import pioneer3dx as p3dx
p3dx.init()
```

3. Program

The code is structured in three parts:

1. The first part is a function for moving forward: you must copy and paste the code of the distance exercise (Distance.ipynb) inside the body of the function template, in the following cell.
2. The second part is a similar function for turning, where you can copy and paste the code of the angle exercise (Angle.ipynb).
3. Finally, the third part is the main code, consisting of a loop that calls the previous functions four times. The code also displays the pose of the robot (position and orientation) before and after the motion.

In [ ]:
```
def forward():
    # copy and paste your code here
    ...
```

In [ ]:
```
def turn():
    # copy and paste your code here
    ...
```

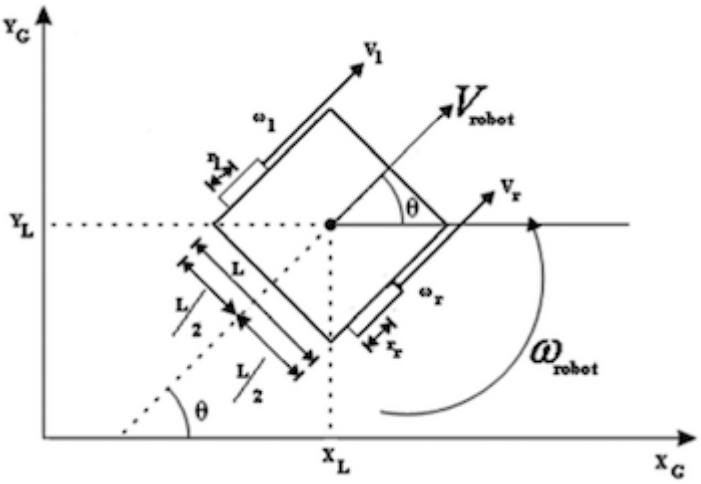
In [ ]:
```
print('Pose of the robot at the start')
p3dx.pose()
for _ in range(4):
    forward()
    turn()
print('Pose of the robot at the end')
p3dx.pose()
```

The trajectory can also be displayed:

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt   # WARNING: the first time, this import
 can take up to 30 seconds
x, y = p3dx.trajectory()          # because of font cache building, plea
se be patient and wait
plt.plot(x,y);
```

Next exercise: Robot Speed (Robot%20Speed.ipynb)

(http://www.intechopen.com/books/advances-in-robot-navigation/conceptual-bases-of-robot-navigation-modeling-control-and-applications)



Exercise: Robot Speed.

At the lowest level, we control the **angular** velocity of the right and left wheels (ω_r, ω_l).

The relationship between the angular and **linear** velocity of the wheels is:

$$V_r = \omega_r r$$
$$V_l = \omega_l r$$

where r is the radius of the wheel.

The linear and angular velocities **of the robot** can then be expressed by:

$$V_{robot} = \frac{V_r + V_l}{2}$$
$$\omega_{robot} = \frac{V_r - V_l}{L}$$

where L is the distance between the wheels.

However, for programming the robot to move at a desired speed, we need to solve the **inverse problem**: given the linear and angular speed of the robot, determine the angular speed of each wheel.

Computing the speed of the wheels

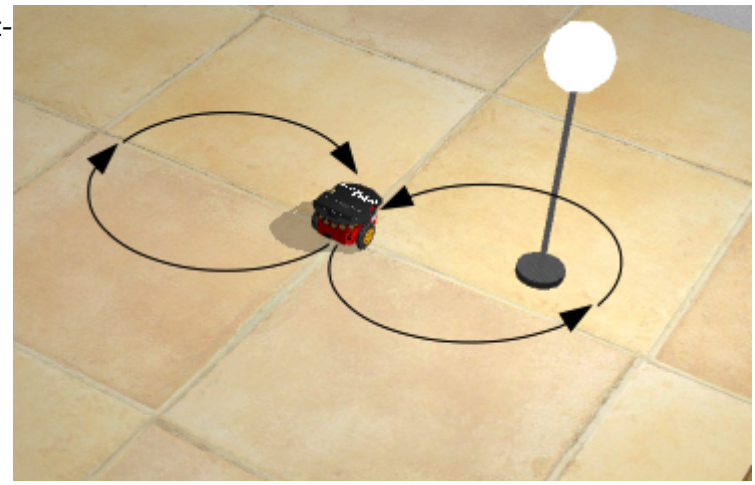
Given the previous equations, we can solve them for the angular velocities of the wheels (ω_r, ω_l):

$$\omega_r = \frac{2V_{robot} + L\omega_{robot}}{2r}$$
$$\omega_l = \frac{2V_{robot} - L\omega_{robot}}{2r}$$

Finally, you must implement the solution in a Python function that receives the robot velocities as arguments, computes the angular velocities of the wheels, and calls the motion function of the robot:

In [ ]:
```
def move(V_robot, w_robot):
    ...
    p3dx.move(w_r, w_l)
```

Eight-shaped trajectory test



We are going to test the motion function with a eight-shaped trajectory. First, the robot will turn left at a constant linear and angular speed. After completing a circle, the robot will turn right at the same linear speed, with the oppposite angular speed. It will complete a second circle and return to the initial point, approximately.

For a given circle radius R, you can define different linear and angular velocities, as long as this relationship is held:

$$V = \omega R$$

For example, you can choose:

$$R = 1m$$
$$V = 0.35m/s$$

For each circle, you can stop the robot after a timeout T:

$$T = \frac{2\pi R}{V}$$

Starting position

For a better visual understanding of the task, it is recommended that the robot starts at the center of the room.

You can easily relocate the robot there by simply restarting the simulation, by clicking on the second icon of the button bar, as depicted in the figure.

Initialization

After restarting the simulation, the robot needs to be initialized.

In [ ]:
```
import packages.initialization
import pioneer3dx as p3dx
p3dx.init()
```

In [ ]:
```
# First circle
R = 1
V = 0.35
w = ...
T = ...
move(V,w)
p3dx.sleep(T)

# Second circle
...
# Stop the robot
move(0,0)
```

The trajectory can also be displayed:

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
x, y = p3dx.trajectory()
plt.plot(x,y);
```

Congratulations!

This was the last exercise of this week.

Please remember to save your work, since we well reuse some code of this week for the exercises in the following modules.