MOOC Autonomous Mobile Robots Week 2

(https://www.cyberbotics.com/doc/guide/using-the-pioneer-3-at-and-pioneer-3-dx-robots)

Week 2 - Ultrasonic Sensing

An autonomous robot needs to acquire knowledge about its environment. This is done by taking measurements using sensors and then extracting information from those measurements.

There is a wide variety of sensors in mobile robots. In this week we present a very popular type of sensor, ultrasonic devices, which are used for measuring distances to solid obstacles.

We will explain how ultrasonic sensors can be used for simple navigation tasks, including wandering while avoiding obstacles, and wall following.

- Ultrasonic Sensors (Ultrasonic%20Sensors.ipynb)
- Detecting Obstacles (Detecting%20Obstacles.ipynb)
- Searching for Free Space (Searching%20Space.ipynb)
- Wandering (Wandering.ipynb)
- Wall Following (Wall%20Following.ipynb)

For more comprehensive information about sensors used on mobile robots, refer to H.R. Everett's book *Sensors for Mobile Robots: Theory and Application* (http://barteverett.com/book.html).

Try-a-Bot: an open source guide for robot programming

Developed by:

Universitat Jaume I — Robotic Intelligence Lab

(http://robinlab.uji.es)

Sponsored by:

IEEE Robotics & Automation Society (http://www.ieee-ras.org)

CYBERBOTICS professional mobile robot simulation (http://www.cyberbotics.com)

The Construct Just Simulate (http://www.theconstructsim.com)

Follow us:

(https://www.facebook.com/RobotProgrammingNetwork) (https://www.youtube.com/user/robotprogrammingnet)

(https://www.generationrobots.com/en/content/65-ultrasonic-sonar-sensors-for-robots)

Ultrasonic Sensors

Ultrasonic sensors work by measuring the return time of a high-frequency sound wave emitted by the sensor (over 20,000 Hz, which is therefore inaudible to humans). As the speed of sound is essentially known, the obstacle's distance can then be deduced.

The distance d of the object causing the reflection is:

$$d = \frac{c \cdot t}{2}$$

where c is the speed of the sound (343 m/s in air at standard pressure and 20ºC) and t is the time of flight.

The Pioneer 3-DX robot includes 8 forward-facing ultrasonic sensors, and 8 optional rear-facing sonar for distance measurements. In the simulations of this course, we are going to use only the forward-facing set of sensors.

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

With the following GUI widget you can move the robot around.

In [ ]:
```
import motion_widget
```

And you can plot a diagram with the position of the robot in the environment, and the measurements of the ultrasonic sensors.

The plot is not updated automatically as the robot moves. You need to click on the "Refresh" button.

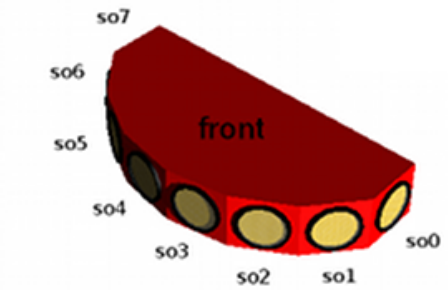
In [ ]:
```
%matplotlib inline
import sonar_plot
```

In [ ]:
```
# in case that the plot does not appear, you may run this other cell
reload(sonar_plot)
```

The values of the measurements are stored in an array:

In [ ]:
```
p3dx.distance
```

The sensors are numbered from 0 to 7 starting from the left side of the robot, in clockwise order (see the figure).



Each measurement can be read individually with the appropriate index in the array, e.g. the first measurement is

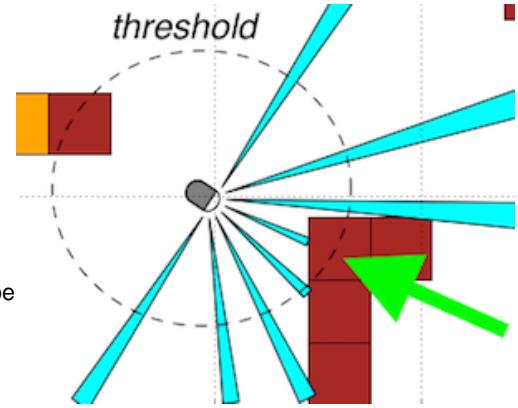
`p3dx.distance[0]`

The following code uses a `for` loop for displaying the values of the eight sensors with a precision of three digits. Even if the robot is not moving, the values are likely to be different from those measured above, because of the (simulated) noise in the readings.

In [ ]:
```
for i in range(8):
    print("Sensor %d is %.3f" % (i, p3dx.distance[i]))
```

Let's test the sensors in a first simple application: detecting obstacles (Detecting%20Obstacles.ipynb).

Detecting Obstacles



An obstacle can be detected by comparing the values returned by the ultrasoinc sensor with a predefined ***distance threshold***.

For values below that threshold, the detected obstacle is considered too close to the robot, and an action should be taken, for example stopping and/or turning, in order to avoid collision.

In the example figure, the value of sensor 3 is less than the threshold (represented by the dotted circle), as signaled by the green arrow.

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

Exercise:

Make a program for the robot to move forward until any of the front sensors (numbered 3 and 4) detects an obstacle below a given distance threshold, for example 1 meter. Use the following template code:

In [ ]:
```
...
while ...:
    p3dx.move(2.5,2.5)
    ...
p3dx.move(0,0)
```

You may confirm the proper functioning of your code with the following test:

In [ ]:
```
for i in range(3,5):
    print("Sensor %d is %.3f" % (i, p3dx.distance[i]))
```

And you can plot a diagram with the position of the robot in the environment, and the measurements of the ultrasonic sensors.

The plot is not updated automatically as the robot moves. You need to click on the "Refresh" button.

In [ ]:
```
%matplotlib inline
import sonar_plot
```

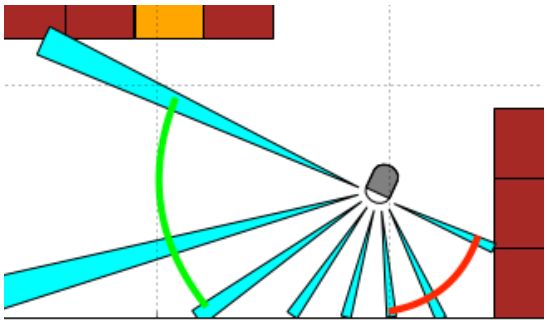
In [ ]:
```
# in case that the plot does not appear, you may run this cell
reload(sonar_plot)
```

Let's try the next step: searching for free space (Searching%20Space.ipynb).

Searching for Free Space

After an obstacle is detected, the robot must turn either left or right in search for free space, and move forward again.

Here is one possible solution:



- Find the minimum of the three left sensors (0, 1, 2)
- Find the minimum of the three right sensors (5, 6, 7)
- If the left minimum is bigger than the right minimum
  - Turn left
- Else
  - Turn right
- In any case, keep turning until both front sensors (3, 4) are bigger than the chosen minimum

In the sample figure, the robot would turn right, since the minimum of the right side sensors (green arc) is bigger than the one of the left side (red arc).

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

Minimum and maximum of an array

For finding the minimum and maximum of an array, you can use the Python built-in functions min and max (https://docs.python.org/2/library/functions.html#max).

In [ ]:
```
p3dx.distance
```

In [ ]:
```
min(p3dx.distance)
```

In [ ]:
```
max(p3dx.distance)
```

For finding the minimum in a specific sub-array, you can use the Python slice notation for lists (http://stackoverflow.com/questions/509211/explain-pythons-slice-notation):

In [ ]:
```
# left sensors
p3dx.distance[:3]
```

In [ ]:
```
# front sensors
p3dx.distance[3:5]
```

In [ ]:
```
# right sensors
p3dx.distance[5:]
```

Exercise

Implement the presented algorithm for turning towards free space.

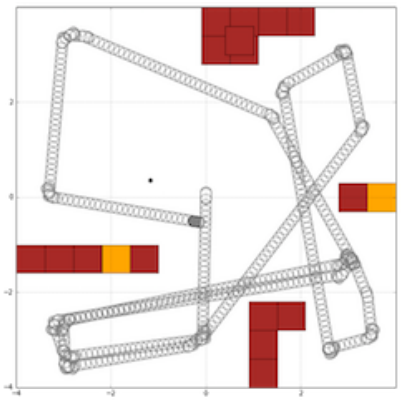
In [ ]:
```
...
if ...
    while ...
        p3dx.move(-1,1)
else:
    while ...
        p3dx.move(1,-1)
p3dx.move(0,0)
```

You can plot a diagram with the resulting position of the robot in the environment, and the measurements of the ultrasonic sensors.

In [ ]:
```
%matplotlib inline
import sonar_plot
```

Let's put together the last two exercises in a simple application: wandering (Wandering.ipynb).

Wandering



A simple wandering behavior can be achieved by the combination of the previously coded exercises:

```
repeat forever
  move forward until an obstacle is detected
  turn either left or right for free space
```

Instead of startinf from scratch, you will reuse the code in two Python functions, which can be called from inside the main loop.

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

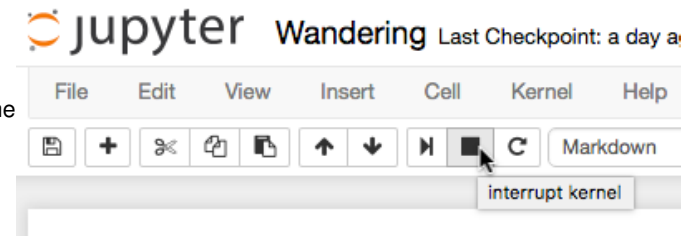
First, you need to copy and paste the code inside the following functions:

In [ ]:
```
def forward():
    ...
```

In [ ]:
```
def turn():
    ...
```

Finally, you should run the main loop in the following cell.

The execution can be stopped at any time by pressing the *interrupt kernel* button.



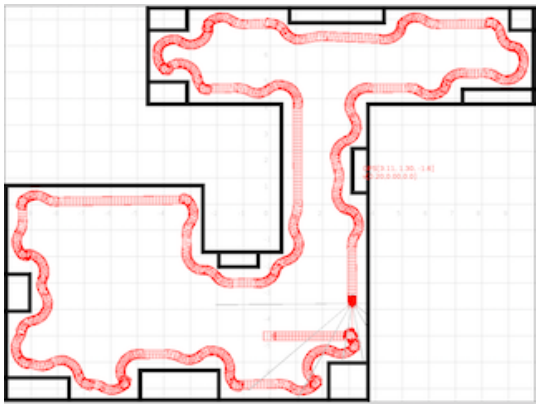
In [ ]:
```
try:
    while True:
        forward()
        turn()
except KeyboardInterrupt:
    p3dx.move(0,0)
```

The resulting trajectory can be plotted.

In [ ]:
```
%matplotlib inline
import trajectory_plot
```

Next application: wall following (Wall%20Following.ipynb).

(http://java-player.sourceforge.net/examples-3.php)



Wall Following

This is another popular behavior for mobile robots indoors. The robot keeps a constant distance to a lateral wall (either left or right) while moving forward and turning at corners.

In this way, the robot can explore rooms or corridors safely.

For this application, we are going to implement in Python a Java algorithm by Radu Bogdan Rusu (https://www.linkedin.com/in/radubogdanrusu) from his Javaclient Player/Stage Project (http://java-player.sourceforge.net/examples-3.php).

The full source code can also be found here (img/WallFollowerExample.java).

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

Define the wall threshold.

In [ ]:
```
MIN_WALL_THRESHOLD = 0.3
MAX_WALL_THRESHOLD = 0.4
```

Define the default translational and rotational speeds.

In [ ]:
```
DEF_X_SPEED = 0.2
DEF_YAW_SPEED = 0.15
```

We need to control the speed of the robot with the function that computes the angular velocities of the wheels based on the linear and rotational speed of the robot. This function was defined in an exercise during the first week of the course. You can copy and paste the code here:

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

We need to implement now the `getSonars` function (lines 142-156 of the Java source).

This function returns the minimum value of the left sensors (0, 1, 2) and the minimum value of the front sensors (3, 4).

As in previous exercises, you can use the built-in `min` function and array slicing, for greatly simplifying the code.

In [ ]:
```
def getSonars():
    leftSide = ...
    frontSide = ...
    return leftSide, frontSide
```

Now it's time to implement the `getWall` function (lines 109-140 of the Java source).

This function works in two steps:

1. The robot goes forward until a wall is detected
2. The robot turns right until it gets a smaller value in sonar 0

WARNING: due to the fact that sensor noise is simulated in Webots, the condition

```
sonarValues[0] <= previousLeftSide
```

will not work properly, and it has been replaced by a comparison with the wall threshold.

In [ ]:
```
def getWall():
    leftSide, frontSide = getSonars()
    while ...:
        move(...)
        leftSide, frontSide = getSonars()

    while p3dx.distance[0] > MAX_WALL_THRESHOLD: # don't use previousLeftSide
        if ...:
            yawSpeed = ...
        else:
            yawSpeed = ...
        move(...)
        leftSide, frontSide = getSonars()
    move(0,0)
```

Finally, we implement the `main` function (lines 52-107 of the Java source - obviously the Player/Stage code is not necessary).

In [ ]:
```
try:
    # Go ahead and find a wall and align to it on the robot's left side
    getWall()
    while True:
        # get all SONAR values and perform the necessary adjustments
        leftSide, frontSide = getSonars()
        # by default, just move in front
        xSpeed = ...
        yawSpeed = ...
        # if we're getting too close to the wall with the front side...
        if ...:
            # back up a little bit if we're bumping in front
            ...
        else:
            # if we're getting too close to the wall with the left side...
            if ...:
                # move slower at corners
                ...
            else:
                # if we're getting too far away from the wall with the left sid
                if ...:
                    # move slower at corners
                    ...
        # Move the robot
        move(...)
except KeyboardInterrupt:
    move(0,0)
```

The resulting trajectory of the robot can be plotted.

In [ ]:
```
%matplotlib inline
import trajectory_plot
```

Try-a-Bot: an open source guide for robot programming

Developed by:

(http://robinlab.uji.es)

Sponsored by:

(http://www.ieee-ras.org)

(http://www.cyberbotics.com)

(http://www.theconstructsim.com)

Follow us:

(https://www.facebook.com
/RobotProgrammingNetwork)

(https://www.youtube.com
/user/robotprogrammingnet)