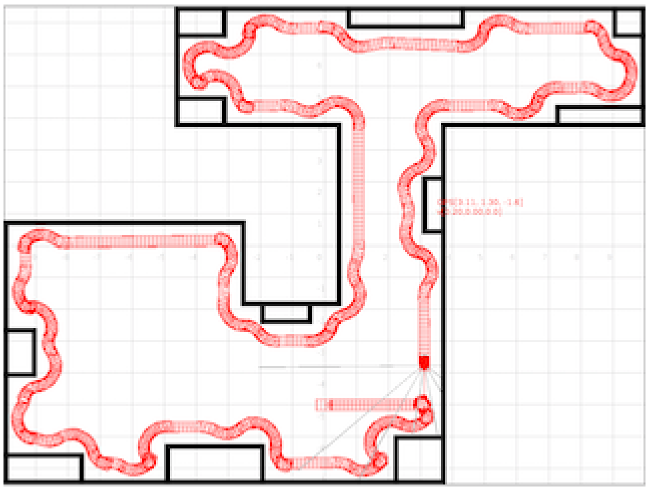


(http://java-player.sourceforge.net/examples-3.php)

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 an algorithm developed by Radu Bogdan Rusu (https://www.linkedin.com/in/radubogdanrusu) in his Javaclient Player/Stage Project (http://java-player.sourceforge.net/examples-3.php).

In [1]:

```
import packages.initialization
import pioneer3dx as p3dx
#import operator
p3dx.init()
```

Define two wall thresholds: the following behavior will try to keep the robot at a distance to the wall between those values.

In [2]:

```
min_wall_threshold = 0.3
max_wall_threshold = 0.4
```

Define the default translational and rotational speeds.

In [3]:

```
v_speed = 0.20 # default forward velocity
w_speed = 0.15 # default turning velocity

wheel_diameter = 0.1953          # In meters
wheel_radius = wheel_diameter/2  # In meters
axis_length = 0.33               # In meters
```

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 introduced in the first week of the course.

In [4]:

```
def move(v_robot, w_robot):
    w_r = (2 * v_robot + axis_length * w_robot) / (2*wheel_radius) # In rad/sec
    w_l = (2 * v_robot - axis_length * w_robot) / (2*wheel_radius) # In rad/sec
    p3dx.move(w_l, w_r)
```

Reading the ultrasonic sensors

In this implementation, the robot will follow the wall at its left side.

We need to implement a function named `getSonars`, which 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.

In [5]:

```
def getSonars():
    left_side_min_dist  = min(p3dx.distance[0:3])
    front_side_min_dist = min(p3dx.distance[3:5])
    #lmin_index, left_side_min_dist = min(enumerate(p3dx.distance[0:3]), key=ope
rator.itemgetter(1))
    #fmin_index, front_side_min_dist = min(enumerate(p3dx.distance[3:5]), key=op
erator.itemgetter(1))
    #print("Dmin_l(%d): %.3f m, Dmin_r(%d): %.3f" % (lmin_index, left_side_min_d
ist, fmin_index+3, front_side_min_dist))
    return left_side_min_dist, front_side_min_dist
```

Before going on with more code, we suggest to make a simple test for the function:

In [6]:

```
print(p3dx.distance)
getSonars()
```

```
[2.6241095066070557, 5.073628902435303, 2.3963849544525146, 3.908734
083175659, 3.9264471530914307, 4.469640254974365, 1.399770975112915,
3.8916399478912354]
```

Out[6]:

```
(2.3963849544525146, 3.908734083175659)
```

No syntax errors? Does it produce the good results? If not, you may edit the function and try again. When eveything works, please proceed with the next code.

Getting close to the wall

Initially, the robot can be in the middle of the room. Before the wall following behavior is triggered, we need to make sure that the robot is properly located. We will implement a function named `getWall`, which will work in two steps:

```
1. The robot goes forward until a wall is detected in front of it
2. The robot turns right until it reads a value in sonar 0 smaller than a
 given threshold
    1. if the front distance is smaller than the left distance
       turn the robot at x3 speed
    2. otherwise turn at default speed
```

In [7]:

```
def getWall():
    # The robot goes forward until a wall is detected in front of it.
    left_side_min_dist, front_side_min_dist = getSonars()
    while front_side_min_dist > max_wall_threshold:
        move(v_speed, 0)
        left_side_min_dist, front_side_min_dist = getSonars()

     #print("---")

    # The robot turns right until it reads a value in sonar 0 smaller than a giv
en threshold.
    while p3dx.distance[0] > max_wall_threshold:

        #print("ds0: %.3f" % p3dx.distance[0])
        left_side_min_dist, front_side_min_dist = getSonars()
        # if the front distance is smaller than the left distance turn the robot
 at x3 speed
        if left_side_min_dist > front_side_min_dist:
            w_robot = - 3 * w_speed
            #print("-3*w_speed")
        # otherwise turn at default speed
        else:
            w_robot = -w_speed
            #print("-1*w_speed")
        move(0, w_robot)
    move(0,0)
```

Again, let's test the function:

In [8]:

```
try:
    getWall()
except KeyboardInterrupt:
    move(0,0)
```

Main loop

Finally, we implement the main loop of Rusu's algorithm:

```
1. Go ahead and find a wall and align to it on the robot's left side
2. Repeat forever
   1. by default, just move forward
   2. if we're getting too close to the wall with the front side
      go backward and turn right quickly (x4)
   3. if we're getting too close to the wall with the left side
      move forward slowly (x0.5) and turn right
   4. if we're getting too far away from the wall with the left side
      move forward slowly (x0.5) and turn left
```

In [9]:

```
try:
    # Go ahead and find a wall and align to it on the robot's left side.
    getWall()
    while True:
        left_side_min_dist, front_side_min_dist = getSonars()
        # By default, just move forward
        v_robot = v_speed
        w_robot = 0
        # If we're getting too close to the wall with the front side...
        if front_side_min_dist < max_wall_threshold:
            # Go backward and turn right quickly (x4)
            v_robot = 0.1*v_speed #go forward!!!!
            w_robot = -4*w_speed
        # If we're getting too close to the wall with the left side...
        elif left_side_min_dist < min_wall_threshold:
            # Move slowly forward (x0.5) and turn right
            v_robot  = 0.5*v_speed
            w_robot = -w_speed
        # If we're getting too far away from the wall with the left side...
        elif left_side_min_dist > max_wall_threshold:
            # Move slowly forward (x0.5) and turn left
            v_robot  = 0.5*v_speed
            w_robot = w_speed
        # Move the robot
        move(v_robot, w_robot)
except KeyboardInterrupt:
    move(0,0)
```

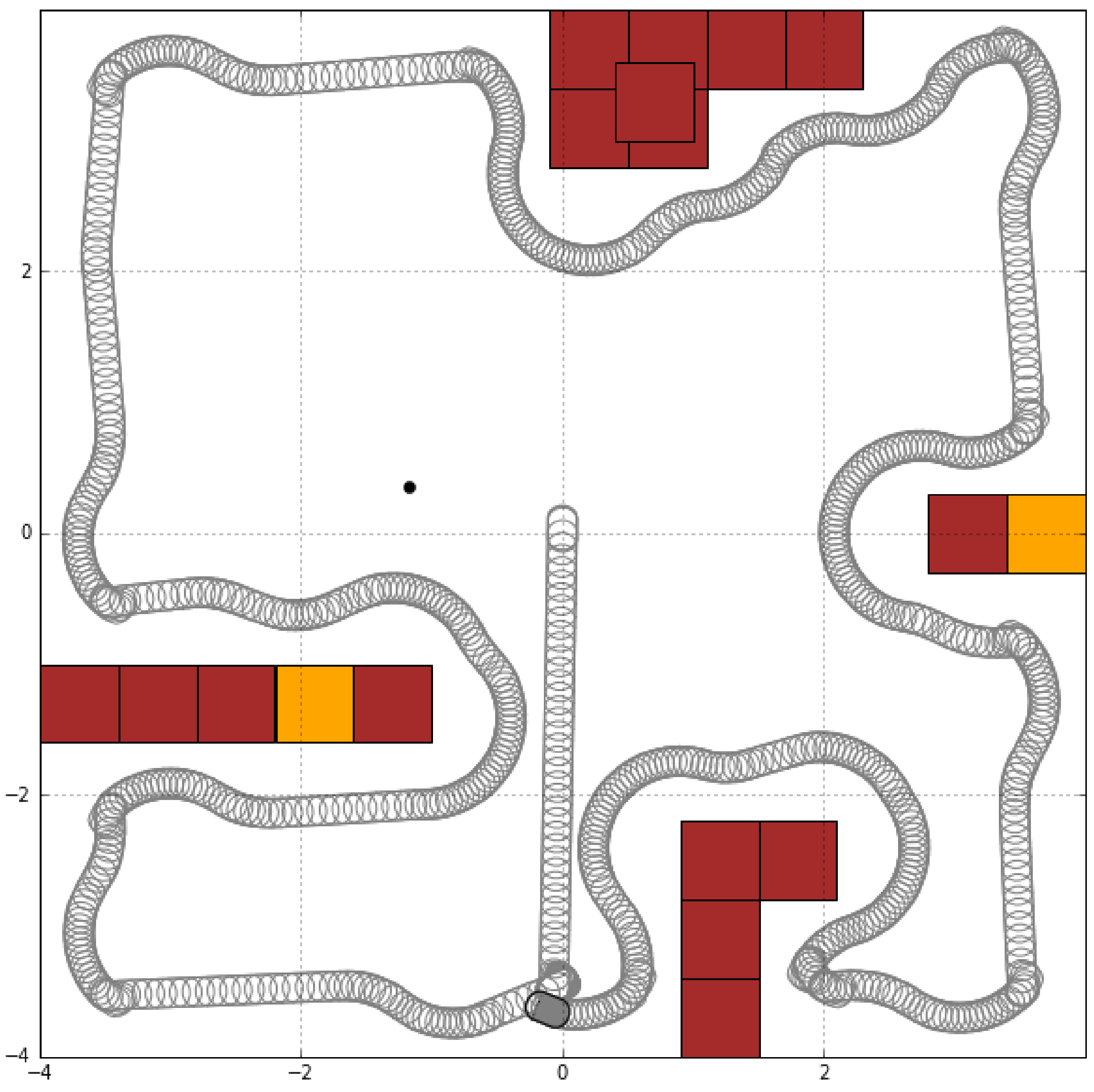
The resulting trajectory of the robot can be plotted.

In [10]:

```
%matplotlib inline
import trajectory
```

In [11]:

```
trajectory.plot()
```



Did it work? If so, congratulations, you have completed the work of this week!

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/robotprogramming