Robotik 6. Übungsblatt von Pascal Müller, Friedrich Müller, Fabian Casares

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
from std_msgs.msg import Float32
from sensor_msgs.msg import LaserScan

import random
import numpy as np
import matplotlib.pyplot as plt
import math

def listener():
    rospy.init_node("listener", anonymous=True)
    #rospy.Subscriber("/scan", LaserScan, callback_scan)
    dot1 = snapshotCoordinates()
    time.sleep(2000)
    dot2 = snapshotCoordinates()
    time.sleep(2000)
    dot3 = snapshotCoordinates()

    xCirc, yCirc, rCirc = getRotationCenterAndRadius(dot1, dot2, dot3)

    axisToAxis = 0.26
    lidarToBackAxis = 0.21
    steeringAngle = getSteeringAngle(rCirc, axisToAxis, lidarToBackAxis)

    rospy.loginfo("I heard and published {} - {}".format(rCirc,steeringAngle))


if __name__=='__main__':
    listener()

#sin(a)=gk/hyp -> sin(a)*hyp=gk
#cos(a)=ak/hyp -> cos(a)*hyp=ak
def getScatterPlot(lidarInput):
    temp = np.empty((len(lidarInput),2),dtype=float)

    for i in range(0,len(temp)):
        if lidar_input[i] == float('inf'):
            temp[i][0]=0
```

```
            temp[i][1]=0
        else:
            temp[i][0]=lidarInput[i]*math.sin(math.radians(i))
            temp[i][1]=lidarInput[i]*math.cos(math.radians(i))

    #Remove zeros (or infinities - this needs to be adjusted)
    temp = temp[np.logical_or(temp[:,0]!=0, temp[:,1]!=0)]

    return temp

#Distance from point (x_0,y_0) to a line by = -ax - c is given by:
#https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line#Line_defined_by_an_equation
#NOTE: The values b and m in this implementation refer to the standard line representation y=m*x +b
class LineModel():

    def __init__(self, point1, point2):

        #We can now calculate the slope m from the points. From that we get a
        self.m = (point1[1]-point2[1])/(point1[0]-point2[0])
        self.a = -self.m

        #We can now rearrange the formula to c= -ax -by and insert one of our points to calculate c.
        self.c = -self.a*point1[0] -point1[1]
        #NOTE: This b is not the same as in the formula! we ignore the b in the formula becuase it is 1. This b here is the offset of the function
        self.b = -self.c

        #Now we prepare the denominators of the formula's fraction for later use (it only depends on constants)
        self.denominatorPoints = pow(self.a,2)+1
        self.denominatorDist = math.sqrt(self.denominatorPoints)

    def dist(self, point):
        #Here we just apply the formula as found in the link above
        return abs(self.a*point[0] + point[1] + self.c)/self.denominatorDist

    def closestPoint(self, point):
        xVal = (point[0] - self.a*point[1] - self.a*self.c)/self.denominatorPoints
        yVal = (-self.a*point[0] + (self.a**2)*point[1] - self.c)/self.denominatorPoints
        return np.array([xVal, yVal])
```

```

    def funcValue(self, xValue):
        return (xValue*self.m +self.b)

    def intersection(self, otherLine):
        xVal = (self.b-otherLine.b)/(otherLine.m-self.m)
        yVal = self.funcValue(xVal)
        return np.array([xVal, yVal])

def ransac(whiteDotSet, percentageThreshold, distanceThreshold, iterations=20):
    bestCandidateSet = None
    antiSetOfBestSet = None
    bestModel = None
    #Percentage means percentage of dots inside the distance threshold of a model line
    bestPercentage = 0

    for j in range(0,iterations):
        twoRands = np.random.randint(0, len(whiteDotSet),2)
        #Haven't found a better way to avoid choosing the same point twice
        if twoRands[0] == twoRands[1]:
            continue

        dotA, dotB = whiteDotSet[twoRands[0]], whiteDotSet[twoRands[1]]

        #A model for a line that goes through dotA and dotB
        ln = LineModel(dotA, dotB)

        def closeEnough(dot):
            return ln.dist(dot)<distanceThreshold

        #Using numpy feature "Boolean Indexing"
        boolSet = np.apply_along_axis(func1d=closeEnough, axis=1, arr=whiteDotSet)
        consensusSet= whiteDotSet[boolSet]

        percentageActual=len(consensusSet)/len(whiteDotSet)

        if(percentageActual>percentageThreshold and percentageActual>bestPercentage):
            bestCandidateSet = consensusSet
            antiSetOfBestSet = whiteDotSet[np.logical_not(boolSet)]
            bestModel = ln
            bestModel.dotsToDisplay=bestCandidateSet
            bestPercentage = percentageActual
```

```python

    return bestCandidateSet, antiSetOfBestSet, bestModel

def randomSubarray(arr, subarrayLen):
    return arr[random.sample(range(0,len(arr)),subarrayLen)]

def getTwoWallsInClockwiseOrder(scatterPlot):
    #The before/after edge might still be switched at this point!
    _, notWallBeforeEdge, wallBeforeEdge = ransac(scatterPlot,percentageThreshold=0.2, distanceThreshold=0.02, iterations=20)
    _, _, wallAfterEdge = ransac(notWallBeforeEdge,percentageThreshold=0.2, distanceThreshold=0.04, iterations=20)

    #If the orientation of these three points is actually counter-clockwise, we need to switch our lines
    if orientation(wallBeforeEdge.closestPoint([0,0]), wallAfterEdge.closestPoint([0,0]), [0,0])<0:
        wallBeforeEdge, wallAfterEdge = wallAfterEdge, wallBeforeEdge

    return wallBeforeEdge, wallAfterEdge

#Orientation of three dots (x1,y1), (x2,y2), (x3,y3): (y2âĹŠy1) (x3âĹŠx2) âĹŠ (y3âĹŠy2) (x2âĹŠx1)
#Source: Slide 10 of http://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf
def orientation(dotA, dotB, dotC):
    #Positive return value means clockwise orientation, negative means counter-clockwise orientation, 0 means all in one line
    return (dotB[1]-dotA[1])*(dotC[0]-dotB[0]) - (dotC[1]-dotB[1])*(dotB[0]-dotA[0])

def getRotationCenterAndRadius(dot1, dot2, dot3)
    #From the Assigment we use:
    #Distance to wall before edge is yi
    #Distance to wall after edge is xi
    #We can now note our three measurements
    x1, y1 = dot1[0], dot1[1]
    x2, y2 = dot2[0], dot2[1]
    x3, y3 = dot3[0], dot3[1]
    #TODO: Input real measurements here

    #Now we fit a circle to the three dots using center and radius form
    #Source: https://www.qc.edu.hk/math/Advanced%20Level/circle%20given%203%20points.htm

```

```
    #LGS:
    # 2*(x2-x1)*x0 + 2*(y2-y1)*y0 = x2^2 + y2^2 -x1^2 -y1^2
    # 2*(x2-x3)*x0 + 2*(y2-y3)*y0 = x2^2 + y2^2 -x3^2 -y3^2
    #We can solve this LGS by matrix magic
    coeffMatr = np.array([[2*(x2-x1),2*(y2-y1)],[2*(x2-x3),2*(y2-y3)]])
    ordinateValues = np.array([x2**2+y2**2-x1**2-y1**2,x2**2+y2**2-x3**2-y3**2])
    x0, y0 = np.linalg.solve(coeffMatr,ordinateValues)
    r = math.sqrt((x1-x0)**2 + (y1-y0)**2)
    return np.array([x0, y0]), r


def getSteeringAngle(lidarTurningRadius, axisToAxisLength, lidarToBackAxisLength):
    #Is this really correct? check again
    rRear = math.sqrt(r**2-lidarToBackAxisLength**2)
    steeringAngle = math.atan(axisToAxisLength/rRear)
    return math.degrees(steeringAngle)

def snapshotCoordinates():
    dataRaw = rospy.wait_for_message("/scan", LaserScan)
    data = dataRaw.ranges
    scatteredData = getScatterPlot(data)
    wallBeforeEdge, wallAfterEdge = getTwoWallsInClockwiseOrder(scatteredData)
    distToWallBeforeEdge = wallBeforeEdge.dist([0,0])
    distToWallAfterEdge = wallAfterEdge.dist([0,0])
    return np.array([distToWallAfterEdge, distToWallBeforeEdge])
```

A) Wir finden die Wand mit dem Ransac Algorithmus und bestimmen dann die kürzeste distanz mit dist(point).

B) getRotationCenterAndRadius(dot1, dot2, dot3) brechnet den Radius wobei getTwoWallsInClockwiseOrder entscheidet welche Wand welche ist wenn das Auto seine Position ändert.

C) getSteeringAngle() gibt dann den Steering Winkel zurück