

```
1 #!/usr/bin/env python
2 import rospy
3 from std_msgs.msg import String
4 from std_msgs.msg import Float32
5 from sensor_msgs.msg import LaserScan
6
7 import random
8 import numpy as np
9 import matplotlib.pyplot as plt
10 import math
11
12 def listener():
13     rospy.init_node("listener", anonymous=True)
14     #rospy.Subscriber("/scan", LaserScan, callback_scan)
15     dot1 = snapshotCoordinates()
16     time.sleep(2000)
17     dot2 = snapshotCoordinates()
18     time.sleep(2000)
19     dot3 = snapshotCoordinates()
20
21     xCirc, yCirc, rCirc = getRotationCenterAndRadius(dot1, dot2, dot3)
22
23     axisToAxis = 0.26
24     lidarToBackAxis = 0.21
25     steeringAngle = getSteeringAngle(rCirc, axisToAxis, lidarToBackAxis)
26
27     rospy.loginfo("I heard and published {} - {}".format(rCirc, steeringAngle))
28
29
30 if __name__ == '__main__':
31     listener()
32
33 #sin(a)=gk/hyp -> sin(a)*hyp=gk
34 #cos(a)=ak/hyp -> cos(a)*hyp=ak
35 def getScatterPlot(lidarInput):
36     temp = np.empty((len(lidarInput), 2), dtype=float)
37
38     for i in range(0, len(temp)):
39         if lidar_input[i] == float('inf'):
40             temp[i][0]=0
```

```

41         temp[i][1]=0
42     else:
43         temp[i][0]=lidarInput[i]*math.sin(math.radians(i))
44         temp[i][1]=lidarInput[i]*math.cos(math.radians(i))
45
46     #Remove zeros (or infinities - this needs to be adjusted)
47     temp = temp[np.logical_or(temp[:,0]!=0, temp[:,1]!=0)]
48
49     return temp
50
51 #Distance from point (x_0,y_0) to a line by  $= -ax - c$  is given by:
52 #https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line#Line_defined_by_an_equation
53 #NOTE: The values b and m in this implementation refer to the standard line representation  $y = mx + b$ 
54 class LineModel():
55
56     def __init__(self, point1, point2):
57
58         #We can now calculate the slope m from the points. From that we get a
59         self.m = (point1[1]-point2[1])/(point1[0]-point2[0])
60         self.a = -self.m
61
62         #We can now rearrange the formula to  $c = -ax - by$  and insert one of our points to
        calculate c.
63         self.c = -self.a*point1[0] -point1[1]
64         #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
65         self.b = -self.c
66
67         #Now we prepare the denominators of the formula's fraction for later use (it only
        depends on constants)
68         self.denominatorPoints = pow(self.a,2)+1
69         self.denominatorDist = math.sqrt(self.denominatorPoints)
70
71     def dist(self, point):
72         #Here we just apply the formula as found in the link above
73         return abs(self.a*point[0] + point[1] + self.c)/self.denominatorDist
74
75     def closestPoint(self, point):
76         xVal = (point[0] - self.a*point[1] - self.a*self.c)/self.denominatorPoints
77         yVal = (-self.a*point[0] + (self.a**2)*point[1] - self.c)/self.denominatorPoints
78         return np.array([xVal, yVal])

```

```

79
80     def funcValue(self, xValue):
81         return (xValue*self.m+self.b)
82
83     def intersection(self, otherLine):
84         xVal = (self.b-otherLine.b)/(otherLine.m-self.m)
85         yVal = self.funcValue(xVal)
86         return np.array([xVal, yVal])
87
88 def ransac(whiteDotSet, percentageThreshold, distanceThreshold, iterations=20):
89     bestCandidateSet = None
90     antiSetOfBestSet = None
91     bestModel = None
92     #Percentage means percentage of dots inside the distance threshold of a model line
93     bestPercentage = 0
94
95     for j in range(0, iterations):
96         twoRands = np.random.randint(0, len(whiteDotSet), 2)
97         #Haven't found a better way to avoid choosing the same point twice
98         if twoRands[0] == twoRands[1]:
99             continue
100
101         dotA, dotB = whiteDotSet[twoRands[0]], whiteDotSet[twoRands[1]]
102
103         #A model for a line that goes through dotA and dotB
104         ln = LineModel(dotA, dotB)
105
106         def closeEnough(dot):
107             return ln.dist(dot) < distanceThreshold
108
109         #Using numpy feature "Boolean Indexing"
110         boolSet = np.apply_along_axis(func1d=closeEnough, axis=1, arr=whiteDotSet)
111         consensusSet = whiteDotSet[boolSet]
112
113         percentageActual = len(consensusSet)/len(whiteDotSet)
114
115         if (percentageActual > percentageThreshold and percentageActual > bestPercentage):
116             bestCandidateSet = consensusSet
117             antiSetOfBestSet = whiteDotSet[np.logical_not(boolSet)]
118             bestModel = ln
119             bestModel.dotsToDisplay = bestCandidateSet
120             bestPercentage = percentageActual

```

```

121
122     return bestCandidateSet, antiSetOfBestSet, bestModel
123
124 def randomSubarray(arr, subarrayLen):
125     return arr[random.sample(range(0, len(arr)), subarrayLen)]
126
127 def getTwoWallsInClockwiseOrder(scatterPlot):
128     #The before/after edge might still be switched at this point!
129     _, notWallBeforeEdge, wallBeforeEdge = ransac(scatterPlot, percentageThreshold=0.2,
130     distanceThreshold=0.02, iterations=20)
131
132     _, _, wallAfterEdge = ransac(notWallBeforeEdge, percentageThreshold=0.2,
133     distanceThreshold=0.04, iterations=20)
134
135     #If the orientation of these three points is actually counter-clockwise, we need to
136     switch our lines
137     if orientation(wallBeforeEdge.closestPoint([0,0]), wallAfterEdge.closestPoint([0,0]),
138     [0,0]) < 0:
139         wallBeforeEdge, wallAfterEdge = wallAfterEdge, wallBeforeEdge
140
141     return wallBeforeEdge, wallAfterEdge
142
143
144 #Orientation of three dots (x1,y1), (x2,y2), (x3,y3): (y2-x1)(x3-x2) - (y3-x2)(x1-x2)
145
146 #Source: Slide 10 of http://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf
147
148 def orientation(dotA, dotB, dotC):
149     #Positive return value means clockwise orientation, negative means counter-clockwise
150     orientation, 0 means all in one line
151     return (dotB[1]-dotA[1])*(dotC[0]-dotB[0]) - (dotC[1]-dotB[1])*(dotB[0]-dotA[0])
152
153
154 def getRotationCenterAndRadius(dot1, dot2, dot3)
155     #From the Assignment we use:
156     #Distance to wall before edge is yi
157     #Distance to wall after edge is xi
158     #We can now note our three measurements
159
160     x1, y1 = dot1[0], dot1[1]
161     x2, y2 = dot2[0], dot2[1]
162     x3, y3 = dot3[0], dot3[1]
163
164
165     #Now we fit a circle to the three dots using center and radius form
166     #Source: https://www.qc.edu.hk/math/Advanced%20Level/circle%20given%203%20points.htm
167
168     #LGS:

```

```

157     # 2*(x2-x1)*x0 + 2*(y2-y1)*y0 = x2^2 + y2^2 -x1^2 -y1^2
158     # 2*(x2-x3)*x0 + 2*(y2-y3)*y0 = x2^2 + y2^2 -x3^2 -y3^2
159     #We can solve this LGS by matrix magic
160     coeffMatr = np.array([[2*(x2-x1), 2*(y2-y1)], [2*(x2-x3), 2*(y2-y3)]])
161     ordinateValues = np.array([x2**2+y2**2-x1**2-y1**2, x2**2+y2**2-x3**2-y3**2])
162     x0, y0 = np.linalg.solve(coeffMatr, ordinateValues)
163     r = math.sqrt((x1-x0)**2 + (y1-y0)**2)
164     return np.array([x0, y0]), r
165
166
167 def getSteeringAngle(lidarTurningRadius, axisToAxisLength, lidarToBackAxisLength):
168     #Is this really correct? check again
169     rRear = math.sqrt(r**2-lidarToBackAxisLength**2)
170     steeringAngle = math.atan(axisToAxisLength/rRear)
171     return math.degrees(steeringAngle)
172
173 def snapshotCoordinates():
174     dataRaw = rospy.wait_for_message("/scan", LaserScan)
175     data = dataRaw.ranges
176     scatteredData = getScatterPlot(data)
177     wallBeforeEdge, wallAfterEdge = getTwoWallsInClockwiseOrder(scatteredData)
178     distToWallBeforeEdge = wallBeforeEdge.dist([0,0])
179     distToWallAfterEdge = wallAfterEdge.dist([0,0])
180     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