

```

1#!/usr/bin/env python
2#####
3# Copyright 2018 ROBOTIS CO., LTD.
4#
5# Licensed under the Apache License, Version 2.0 (the "License");
6# you may not use this file except in compliance with the License.
7# You may obtain a copy of the License at
8#
9#     http://www.apache.org/licenses/LICENSE-2.0
10#
11# Unless required by applicable law or agreed to in writing, software
12# distributed under the License is distributed on an "AS IS" BASIS,
13# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14# See the License for the specific language governing permissions and
15# limitations under the License.
16#####
17
18# Authors: Gilbert #
19
20import rospy
21import time as t # for sleep statements
22from sensor_msgs.msg import LaserScan
23from geometry_msgs.msg import Twist
24import numpy as np
25
26LINEAR_VEL = 0.22
27STOP_DISTANCE = 0.2
28LIDAR_ERROR = 0.05
29SAFE_STOP_DISTANCE = STOP_DISTANCE + LIDAR_ERROR
30# radius of turtlebot is 10 cm and for safety measure LIDAR_ERROR is added.
31EMERGENCY_STOP_DISTANCE = 0.15
32MAX_ANGLE = 2.84
33DEFAULT_TURN_ANGLE = 3.14/2
34
35
36class Obstacle():
37    def __init__(self):
38        self._cmd_pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
39        self.obstacle()
40
41    def get_scan(self):
42        scan = rospy.wait_for_message('scan', LaserScan)
43        scan_filter = []
44
45        left_lidar_samples_ranges = 60
46        right_lidar_samples_ranges = 345
47        front_lidar_samples_ranges = 15
48
49        front_lidar_samples1 = scan.ranges[0:front_lidar_samples_ranges]
50        front_lidar_samples2 = scan.ranges[right_lidar_samples_ranges:360]
51        left_lidar_samples = scan.ranges[front_lidar_samples_ranges:left_lidar_samples_ranges]
52        right_lidar_samples = scan.ranges[300:right_lidar_samples_ranges]
53        front_left_lidar_samples = scan.ranges[14:22]
54        front_right_lidar_samples = scan.ranges[338:346]
55
56        scan_filter.append(left_lidar_samples)
57        scan_filter.append(front_lidar_samples1)
58        scan_filter.append(front_lidar_samples2)
59        scan_filter.append(right_lidar_samples)
60        scan_filter.append(front_left_lidar_samples)
61        scan_filter.append(front_right_lidar_samples)
62
63    return scan_filter
64
65    def obstacle(self):
66        twist = Twist()
67        turtlebot_moving = True
68
69        def small_turns(dir, angle, vel):
70            if(dir == 'left'):
71                twist.linear.x = vel
72                twist.angular.z = angle
73                self._cmd_pub.publish(twist)
74                rospy.loginfo('Turning left')
75                turtlebot_moving = False
76
```

```

77     elif (dir == 'right'):
78         twist.linear.x = vel
79         twist.angular.z = -angle
80         self._cmd_pub.publish(twist)
81         rospy.loginfo('Turning right')
82         turtlebot_moving = False
83
84     def emergency_check(l):
85         # calculate mean of our slices:
86         left_distance = np.mean((l[0]))
87         front_distance = np.mean(l[1] + l[2])
88         right_distance = np.mean(l[3])
89         front_left_distance = np.mean(l[4])
90         front_right_distance = np.mean(l[5])
91
92         # calculate mean of special case slices
93         special_case_right = np.mean(l[3][27:35])
94         special_case_left = np.mean(l[0][10:18])
95
96         if (front_distance < EMERGENCY_STOP_DISTANCE and left_distance < EMERGENCY_STOP_DISTANCE):
97             small_turns('right', MAX_ANGLE, 0.0)
98
99         elif (front_distance < EMERGENCY_STOP_DISTANCE and right_distance < EMERGENCY_STOP_DISTANCE):
100            small_turns('left', MAX_ANGLE, 0.0)
101
102        elif (front_distance < EMERGENCY_STOP_DISTANCE):
103            if (left_distance < right_distance):
104                small_turns('right', MAX_ANGLE, 0.0)
105            else:
106                small_turns('left', MAX_ANGLE, 0.0)
107
108        elif(special_case_right < SAFE_STOP_DISTANCE and special_case_left < SAFE_STOP_DISTANCE):
109            make_180()
110
111        elif(front_left_distance < SAFE_STOP_DISTANCE):
112            small_turns('right', MAX_ANGLE, 0.0)
113
114        elif(front_right_distance < SAFE_STOP_DISTANCE):
115            small_turns('left', MAX_ANGLE, 0.0)
116
117    # turns 180 degrees to the left
118    def make_180():
119        twist.linear.x = 0.0
120        twist.angular.z = MAX_ANGLE
121        self._cmd_pub.publish(twist)
122        rospy.loginfo('Turnings 180 degrees')
123        t.sleep(1.105634)
124        turtlebot_moving = False
125
126    # function converts list of tuples to list of lists and then converts zeros to 3.5:
127    def non_zeros(l):
128        # convert to list
129        for i in range(len(l)):
130            l[i] = list(l[i])
131
132        # if element zero, set to 3.5
133        for i in range(len(l)):
134            for j in range(len(l[i])):
135                if l[i][j] == 0:
136                    l[i][j] = 3.5
137
138    while not rospy.is_shutdown():
139
140        lidar_distances = self.get_scan()
141        non_zeros(lidar_distances)
142
143        # calculate mean of our slices:
144        left_distance = np.mean((lidar_distances[0]))
145        front_distance = np.mean(lidar_distances[1] + lidar_distances[2])
146        right_distance = np.mean(lidar_distances[3])
147        front_left_distance = np.mean(lidar_distances[4])
148        front_right_distance = np.mean(lidar_distances[5])
149
150        emergency_check(lidar_distances)
151
152    # case 1: obstacle in front

```

```

153     if (front_distance < SAFE_STOP_DISTANCE + 0.08):
154         print('obstacle in front')
155
156     # look ahead:
157     if turtlebot_moving:
158         if (right_distance < left_distance):
159             while(front_distance < SAFE_STOP_DISTANCE + 0.08):
160                 small_turns('left', 1.44*front_distance **
161                             (-0.29), 2.64*front_distance**2.42)
162                 front_new = self.get_scan()
163                 non_zeros(front_new)
164                 emergency_check(front_new)
165                 front_distance = np.mean(
166                     front_new[1] + front_new[2])
167
168         elif(left_distance < right_distance):
169             while(front_distance < SAFE_STOP_DISTANCE + 0.08):
170                 small_turns('right', 1.44*front_distance **
171                             (-0.29), 2.64*front_distance**2.42)
172                 front_new = self.get_scan()
173                 non_zeros(front_new)
174                 emergency_check(front_new)
175                 front_distance = np.mean(
176                     front_new[1] + front_new[2])
177
178     # case 2: obstacle to the left or right
179     elif(left_distance < SAFE_STOP_DISTANCE or right_distance < SAFE_STOP_DISTANCE):
180         if turtlebot_moving:
181             if(right_distance < left_distance):
182                 small_turns('left', DEFAULT_TURN_ANGLE, 0.1)
183             else:
184                 small_turns('right', DEFAULT_TURN_ANGLE, 0.1)
185
186     # case 3: obstacle in "blindspot"
187     elif(front_left_distance < SAFE_STOP_DISTANCE+0.15 or front_right_distance <
188           SAFE_STOP_DISTANCE+0.15):
189         if turtlebot_moving:
190             if (front_right_distance < front_left_distance):
191                 print('blind vinkel højre')
192                 while(front_right_distance < SAFE_STOP_DISTANCE + 0.15):
193                     small_turns('left', 0.34*front_right_distance **
194                                 (-0.93), 1.7*front_right_distance**2.23)
195                     front_right_new = self.get_scan()
196                     non_zeros(front_right_new)
197                     emergency_check(front_right_new)
198                     front_right_distance = np.mean(front_right_new[5])
199
200             elif(front_left_distance < front_right_distance):
201                 print('blind vinkel venstre')
202                 while(front_left_distance < SAFE_STOP_DISTANCE + 0.15):
203                     small_turns('right', 0.34*front_left_distance **
204                                 (-0.93), 1.7*front_left_distance**2.23)
205                     front_left_new = self.get_scan()
206                     non_zeros(front_left_new)
207                     emergency_check(front_left_new)
208                     front_left_distance = np.mean(front_left_new[4])
209
210     # case 4: no obstacles => go forward
211     else:
212         twist.linear.x = LINEAR_VEL
213         twist.angular.z = 0.0
214         turtlebot_moving = True
215         self._cmd_pub.publish(twist)
216
217 def main():
218     rospy.init_node('turtlebot3_obstacle')
219     try:
220         obstacle = Obstacle()
221     except rospy.ROSInterruptException:
222         pass
223
224 if __name__ == '__main__':
225     main()

```