Appendices

A Final code for Turtlebot

```
#!/usr/bin/env python
################################################################################
# Copyright 2018 ROBOTIS CO., LTD.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
################################################################################

# Authors: Gilbert #

import rospy
import time as t  # for sleep statements
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist
import numpy as np

LINEAR_VEL = 0.22
STOP_DISTANCE = 0.2
LIDAR_ERROR = 0.05
SAFE_STOP_DISTANCE = STOP_DISTANCE + LIDAR_ERROR
# radius of turtlebot is 10 cm and for safety measure LIDAR_ERROR is added.
EMERGENCY_STOP_DISTANCE = 0.15
MAX_ANGLE = 2.84
DEFAULT_TURN_ANGLE = 3.14/2


class Obstacle():
    def __init__(self):
        self._cmd_pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
        self.obstacle()

    def get_scan(self):
        scan = rospy.wait_for_message('scan', LaserScan)
        scan_filter = []

        left_lidar_samples_ranges = 60
        right_lidar_samples_ranges = 345
        front_lidar_samples_ranges = 15

        front_lidar_samples1 = scan.ranges[0:front_lidar_samples_ranges]
        front_lidar_samples2 = scan.ranges[right_lidar_samples_ranges:360]
        left_lidar_samples = scan.ranges[front_lidar_samples_ranges:left_lidar_samples_ranges]
        right_lidar_samples = scan.ranges[300:right_lidar_samples_ranges]
        front_left_lidar_samples = scan.ranges[14:22]
        front_right_lidar_samples = scan.ranges[338:346]

        scan_filter.append(left_lidar_samples)
        scan_filter.append(front_lidar_samples1)
        scan_filter.append(front_lidar_samples2)
        scan_filter.append(right_lidar_samples)
        scan_filter.append(front_left_lidar_samples)
        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
```

```
        while not rospy.is_shutdown():

            lidar_distances = self.get_scan()
            non_zeros(lidar_distances)

            # calculate mean of our slices:
            left_distance = np.mean((lidar_distances[0]))
            front_distance = np.mean(lidar_distances[1] + lidar_distances[2])
            right_distance = np.mean(lidar_distances[3])
            front_left_distance = np.mean(lidar_distances[4])
            front_right_distance = np.mean(lidar_distances[5])

            emergency_check(lidar_distances)

            # case 1: obstacle in front
            if (front_distance < SAFE_STOP_DISTANCE + 0.08):
                print('obstacle in front')

                # look ahead:
                if turtlebot_moving:
                    if (right_distance < left_distance):
                        while(front_distance < SAFE_STOP_DISTANCE + 0.08):
                            small_turns('left', 1.44*front_distance **
                                        (-0.29), 2.64*front_distance**2.42)
                            front_new = self.get_scan()
                            non_zeros(front_new)
                            emergency_check(front_new)
                            front_distance = np.mean(
                                front_new[1] + front_new[2])

                    elif(left_distance < right_distance):
                        while(front_distance < SAFE_STOP_DISTANCE + 0.08):
                            small_turns('right', 1.44*front_distance **
                                        (-0.29), 2.64*front_distance**2.42)
                            front_new = self.get_scan()
                            non_zeros(front_new)
                            emergency_check(front_new)
                            front_distance = np.mean(
                                front_new[1] + front_new[2])

            # case 2: obstacle to the left or right
            elif(left_distance < SAFE_STOP_DISTANCE or right_distance < SAFE_STOP_DISTANCE):
                if turtlebot_moving:
                    if(right_distance < left_distance):
                        small_turns('left', DEFAULT_TURN_ANGLE, 0.1)
                    else:
                        small_turns('right', DEFAULT_TURN_ANGLE, 0.1)

            # case 3: obstacle in "blindspot"
            elif(front_left_distance < SAFE_STOP_DISTANCE+0.15 or front_right_distance <
    SAFE_STOP_DISTANCE+0.15):
                if turtlebot_moving:
                    if (front_right_distance < front_left_distance):
                        print('blind vinkel hojre')
                        while(front_right_distance < SAFE_STOP_DISTANCE + 0.15):
                            small_turns('left', 0.34*front_right_distance **
                                        (-0.93), 1.7*front_right_distance**2.23)
                            front_right_new = self.get_scan()
                            non_zeros(front_right_new)
                            emergency_check(front_right_new)
                            front_right_distance = np.mean(front_right_new[5])

                    elif(front_left_distance < front_right_distance):
                        print('blind vinkel venstre')
                        while(front_left_distance < SAFE_STOP_DISTANCE + 0.15):
                            small_turns('right', 0.34*front_left_distance **
                                        (-0.93), 1.7*front_left_distance**2.23)
                            front_left_new = self.get_scan()
                            non_zeros(front_left_new)
                            emergency_check(front_left_new)
                            front_left_distance = np.mean(front_left_new[4])

            # case 4: no obstacles => go forward
            else:
                twist.linear.x = LINEAR_VEL
                twist.angular.z = 0.0
```

```
                turtlebot_moving = True
                self._cmd_pub.publish(twist)


def main():
    rospy.init_node('turtlebot3_obstacle')
    try:
        obstacle = Obstacle()
    except rospy.ROSInterruptException:
        pass


if __name__ == '__main__':
    main()
```