```python
#!/usr/bin/env python3
# coding=utf-8
import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist
import numpy as np
Ka=1
Kp=1
class TurtlebotController:
    def __init__(self):
        rospy.init_node("TurtlebotController")
        self.vel_pub = rospy.Publisher("cmd_vel", Twist, queue_size=10)
        self.vel_msg = Twist()

        rospy.Subscriber("/scan", LaserScan, self.laser_callback)
        self.pillar_pos = [0.0, 0.0]

    def laser_callback(self, msg):
        ranges = np.array(msg.ranges)
        valid_ranges = ranges[np.isfinite(ranges)]

        if valid_ranges.size == 0:
            return

        mean_dist = np.mean(valid_ranges)
        std_dist = np.std(valid_ranges)

        threshold = 2
        lower_bound = mean_dist - threshold * std_dist
        upper_bound = mean_dist + threshold * std_dist

        outliers_indices = [i for i, dist in enumerate(ranges) if np.isfinite(dist) and (dist < lower_bound or dist > upper_bound)]
        outliers = [ranges[i] for i in outliers_indices]

        print("突出的距离值:", outliers)
        rate=rospy.Rate(30)
        for count in outliers_indices:
            ang = msg.angle_min + msg.angle_increment * count
            rospy.loginfo(f"Pillar is {ranges[count]:.2f}m away at {ang / np.pi * 180.0:.2f} degrees")


            x = ranges[count] * np.cos(ang)
            y = ranges[count] * np.sin(ang)

            self.pillar_pos.append([x, y])
            rospy.loginfo(f"Pillar's coordinate to Turtlebot is [{x:.2f}, {y:.2f}]")

            # 调整朝向并驱动turtlebot
            # if(x<0.5 or y<0.5):
            #     self.adjust_heading(0)
            #     self.adjust_speed(ranges[count])
            # else:
            self.adjust_heading(ang)
            self.adjust_speed(ranges[count])

            self.drive_turtlebot()
            rate.sleep()

    def adjust_heading(self, angle):
        rospy.loginfo(f"Adjusting heading to angle: {angle / np.pi * 180.0:.2f} degrees")
        self.vel_msg.angular.z = min(360,max(0,Ka*np.tan(angle)) )#根据角度调整转向


    def adjust_speed(self, distance):
        rospy.loginfo(f"Adjusting speed based on distance: {distance:.2f}m")
        self.vel_msg.linear.x = Kp*min(3, max(0.5, (distance-0.3)))

    def drive_turtlebot(self):
        self.vel_pub.publish(self.vel_msg)
        rospy.loginfo("Driving Turtlebot")

if __name__ == "__main__":
    controller = TurtlebotController()
    rospy.spin()
```