**Wall Following Algorithm**

The first problem is to find, where is the wall. You can use laser rangefinder and find minimal values (as in Braitenberg vehicle tutorial). This way you have the distance from the closest obstacle and the angle relative to the robot.

To succesfully follow the wall, you need to keep the same distance and angle (π/2 or −π/2 depending on position of the wall - the left or right side of the robot). You can use two PID like controllers (which will change angular velocity) to maintain these values. Wall following is implemented in the NodeWallFollowing class.

### Processing input and output

Input to this algorithm are the data from laser rangefinder and output is Twist message.

You need to know the angle of the minimal value from ranges in laser scan data. To calculate it use this equation

where

φis angle of i-th element of vector,

i is index of element in vector,

l is lenght of the vector and

θis angle incrementation (angle\_increment).

If the minimal value was determined from the whole ranges vector, the robot could start following the oposite wall in tight spaces. To ensure this does not happen, find minimum only from the appropriate half of ranges vector. Finding the minimal value is done in a loop so you can adjust its start and end according to these equations:

Where *d* is direction (1 for following wall on left side of robot, −1 for wall on right side), is an index of the element in the vector where the loop starts**;** is an index of the element in the vector where the loop ends and *l* is a lenght of the the vector.

### PD controller for direction

You need to calculate the error value first:

*-*

where

is error value, is the distance of the closest obstacle (minimal value from the ranges vector) measured from laser scan position and is the desired distance from the wall (reference value).

You can use discrete PD controller to keep desired distance from the wall:

where is equal to ;

To simplify calculation you can assume, that is a constant value (in reality it is not, but small diferences will not have significant effect on the controller).

is intervention from PD controller;

is time of n-th iteration;

is proporcional constant;

is derivative constant.

### P controller for angle For controlling the angle of the closest wall suffices a simple proportional controller. Calculate error value first:

where is the error value;

is the angle of the closest wall and dis a direction (1 for following wall on left side of robot, −1 for wall on right side);

Now you can use simple P controller to calculate the intervention:

where is intervention from PD controller, is time of n-th iteration and is proportional constant. The final angular velocity could be calculated as a sum of interventions from PD and P controller:

If the linear velocity would be constant, robot would not manage to turn in corners in time or the velocity would have to be too low. This could be solved by checking the distance of a point in front of the robot and if it is too close, robot slows down (or stops linear movement completely). Than it has enough time to turn around.

Another issue may occur when the wall ends. The robot might not be able to turn around in time and it could go forward and start following another wall. You can solve it by checking angle of the closest obstacle and if the absolute value of that angle is bigger than 1.75(which means the obstacle is already the behind the robot), lower the linear velocity.

**寻墙算法**

第一个问题是确定墙在哪里。使用激光雷达来确定最小值。通过这种方法可以发现离机器人最近障碍物的角度和距离。

为了成功地进行寻墙，需要保证机器人离墙固定的距离和角度（π/2或-π/2取决于墙的位置——机器人的左边或右边）。可以使用两个PID控制器来对这两个参数进行控制。墙跟踪是在 NodeWallFallowing 类中实现的。

**算法的输入和输出**

算法的输入是激光雷达的数据，输出是Twist消息类型。我们必须确认scan数据中的最小值对应的角度范围，可通过如下公式计算得出：

其中，是第i个元素的角度;i是向量中元素的索引;l是向量的长度;θ是角度增量。

如果从整个距离向量中确定最小值，机器人可能在狭窄的空间中开始跟随对面墙。为了避免这种情况发生，只从适当的范围向量的一半找到最小值。找到最小值是在一个循环中完成的，所以你可以根据以下公式调整它的开始和结束:

其中，

d是朝向，（-1是右侧，1是左侧）;

是向量中元素的索引，代表循环开始;

是向量中元素的索引，代表循环结束;

代表向量的长度。

**使用PD控制器来控制方向**

首先需要计算误差值：*-*

其中，

是误差值;

是最近障碍物的距离,即向量中距离值的最小值;

是期望的离墙距离;

使用PD控制器来保证离墙的期望距离：

其中， 等价于

为了简化计算，可以假设是一个常数值(实际上不是，但微小的差异不会对控制器产生显著影响)。

是PD控制器的干预值;

是比例参数、是微分参数;

是第*n*次迭代的时间;

### 使用P控制器来控制角度

为了控制最近墙的角度，只需一个简单的比例控制器,误差值定义如下：

其中，是误差值;

### 是最近墙的角度，并指示方向(机器人左侧跟随墙为1，右侧墙为−1); 现在你可以使用简单的P控制器来计算干预值:

其中，是PD控制器的干预值，是第n次迭代的时刻，是比例参数。最终角速度可以计算为PD和P控制器干预之和:

如果机器人的速度是恒定的，可能在角落处无法及时转弯掉头或者速度很低。这个问题可以通过检查机器人前方的一个点的距离来解决。如果距离太近，机器人就会减速（或者完全停止直线方向的运动）。

另一个可能发生问题的地方在墙结束的时候。机器人可能不会及时地转向，可能会向前走，开始跟踪其它的墙。我们可以通过检查最近障碍物的角度来解决这个问题，如果这个角度的绝对值大于1.75,这意味着障碍物已经在机器人的后面，降低线速度。