5.5.1.4. Wall Following

The Hybrid Avoid Obstacles and Go-to-Goal algorithm alone can not solve The Cul-de-sac Problem. We need a behavior that will cause the robot to move parallel to the obstacle and even move away from the goal if needed to move around a non-convex obstacle.

As with Hybrid Avoid Obstacles and Go-to-Goal, the robot heading (α) is calculated as the weighted sum of α_{ao} to avoid obstacles and a value called α_{wall} which will be designed to follow the turns of a wall at a fixed distance D_{wall}. Since the robot is expected to follow somewhat near a wall, α_{ao} will be larger than it was for the Hybrid algorithm, so we will use a larger threshold value to calculate the weighting variable h.

$$h = \begin{cases} \frac{|\alpha_{ao}|}{Max_{\alpha\_wall}} & \text{if } |\alpha_{ao}| < Max_{\alpha\_wall} \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

$$\alpha = h \cdot \alpha_{ao} + (1 - h) \cdot \alpha_{wall}$$

Thus, for $|\alpha_{ao}| \geq Max_{\alpha\_wall}$, we have pure collision avoidance:

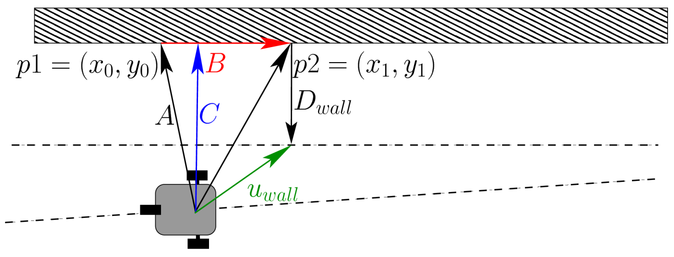
$$\alpha = \alpha_{ao}$$

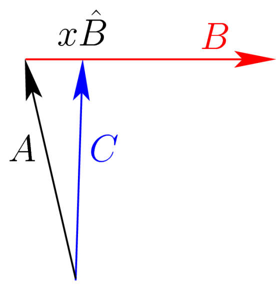
Note that when the wall ahead of the robot turns in towards the robot, the α_{ao} term will dominate the equation. When the wall is straight or turns away from the robot, the value of α_{wall} becomes critical to the heading angle calculation.

5.5.1.4.1. Perpendicular Vector Wall Follower

This algorithm uses a bit of linear algebra of vectors to find a point to drive to that keeps such that a wall is followed. It requires the robot to have five distance sensors (one in front and two to each side of the robot pointing at about 45 and 90 degrees to the robot).

Let the end points of the first two distance sensors on the side of the robot towards the wall be points $p1$ and $p2$ in the world coordinate frame. Given the pose of the robot, we can find a vector from the robot to point $p1$, $\boldsymbol{A} = p1 - [x\ y]^T$. We can also define a vector between points $p1$ and $p2$, $\boldsymbol{B} = p2 - p1$. Vector $\boldsymbol{B}$ generally defines the direction that the robot should travel; but more specifically, we wish to drive towards a point D_{wall} distance from the closest point along vector $\boldsymbol{B}$ to the robot. The closest point is $x\hat{\boldsymbol{B}}$ from point $p1$, where x is a scalar value that we need to find and $\hat{\boldsymbol{B}}$ is a normalized (unit vector) of $\boldsymbol{B}$. We will use a vector, $\boldsymbol{C}$, from the robot to $x\hat{\boldsymbol{B}}$ to find the target driving point.





$$\boldsymbol{C} = \boldsymbol{A} + x\hat{\boldsymbol{B}}$$

$$\hat{\boldsymbol{B}} = \frac{\boldsymbol{B}}{\|\boldsymbol{B}\|}$$

Since vectors $\boldsymbol{B}$ and $\boldsymbol{C}$ are to be perpendicular, the dot product between the vectors should have a value of zero.

$$\hat{\boldsymbol{B}}^{\boldsymbol{T}} \left(\boldsymbol{A} + x\,\hat{\boldsymbol{B}}\right) = 0$$

$$\hat{\boldsymbol{B}}^{\boldsymbol{T}}\,\hat{\boldsymbol{B}}\,x = -\hat{\boldsymbol{B}}^{\boldsymbol{T}}\,\boldsymbol{A}$$

Since $\hat{\boldsymbol{B}}$ is a unit vector, $\hat{\boldsymbol{B}}^{\boldsymbol{T}}\hat{\boldsymbol{B}} = 1$. Vectors of dot products may also be reversed if desired.

$$x = -\boldsymbol{A}^{\boldsymbol{T}}\,\hat{\boldsymbol{B}}$$

$$\boldsymbol{C} = \boldsymbol{A} - \boldsymbol{A}^{\boldsymbol{T}}\,\hat{\boldsymbol{B}}\,\hat{\boldsymbol{B}}$$

Now the vector, $\boldsymbol{u_{wall}}$ defining the robot driving direction is given by

$$\boldsymbol{u_{wall}} = \boldsymbol{p2} - \begin{bmatrix} x \\ y \end{bmatrix} - D_{wall}\,\frac{\boldsymbol{C}}{\|\boldsymbol{C}\|}$$

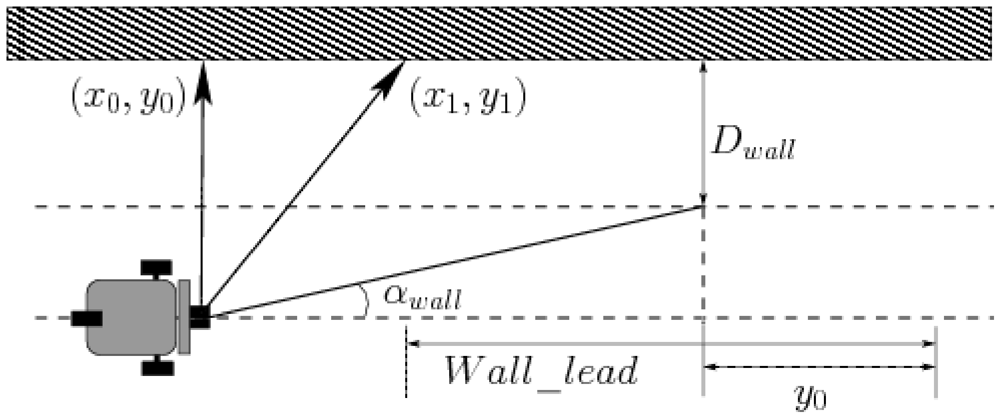
$$\alpha_{wall} = atan2(u_{wall}(2), u_{wall}(1))$$

The robot's current position is subtracted because $p2$ is in the global coordinate frame. As mentioned before, this driving vector should be blended with the vector for avoiding obstacles.

5.5.1.4.2. Virtual Triangle Wall Follower

This algorithm is one that I developed in 2015. It is based on trigonometry functions to give a smooth turning behavior. The algorithm forms a model of a right triangle where sonar measurements determine the lengths of the sides opposite and adjacent to the angle α_{wall}.

The calculation uses the x, y coordinates from the first two sonar measurements on the side of the wall.



Right Triangle Model for Wall Following

Use the *atan2* function to calculate α_{wall}:

$$\alpha_{wall} = atan2((y_1 - D_{wall}), (x_1 + Wall\_lead - y_0))$$

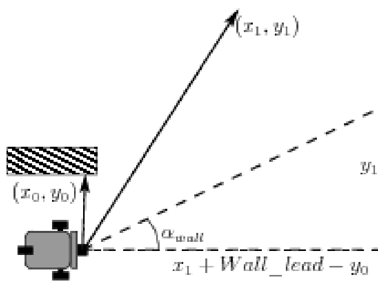
The length of the second sonar measurement is an indicator of the near future path provided by the wall. Thus the right triangle model begins with (x_1, y_1) on the adjacent and opposite sides of the triangle. The vector to (x_1, y_1) always spans the same angle, but the length of the measurement relates to the distance from the wall and turns in the wall ahead.

The value of D_{wall}, which is a tunable constant, indicates how far from the wall we want to be. Thus $y_1 - D_{wall}$ is the length of the opposite side of the triangle. If $y_1 - D_{wall} = 0$, then the robot is following the wall as desired.

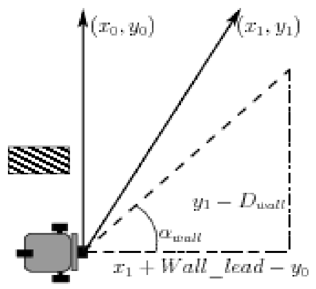
The adjacent side length is calculated from three variables.

- Like y_1, x_1 has valuable information about the contour of the wall. It will be shorter if the wall turns in towards the robot and longer if it turns away. When the robot nears the end of a wall that turns away from the robot, both y_1 and x_1 will become much larger.
- The second variable, $Wall\_lead$ is a tunable constant to effectively adjust the gain of the controller. The initial value was set at 2 meters. It should be increased if the robot is turning too aggressively and reduced if the turning is too sluggish.

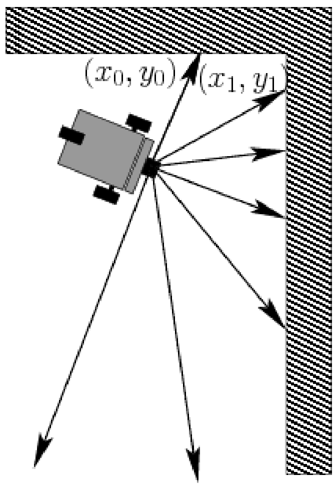
  The minimum value of $Wall\_lead$ can be calculated by considering the maximum value that we want α_{wall}, $\pi/4$ seems reasonable. Taking the maximum values for the two sonar measurements and their angles, when $D_{wall} = 0.4$, $Min\{Wall\_lead\} = 1.772$, so our value of 2.0 seems reasonable.
- The value of y_0 tells us how far the robot currently is from the wall. When the robot reaches the end a point where the wall turns away from the robot, y_0 will suddenly become larger. Since this value is subtracted from length of the adjacent side of the model right triangle, this increased size of y_0 will cause the robot to turn. Thus, when the wall turns away from the robot, y_0 can be considered a gating factor to keep the robot from turning until robot goes past the end of the wall. Correspondingly, when the robot is close to the wall, the small value of y_0 will cause the robot to turn away from the wall.



(x_1, y_1)
$y_1 - D_{wall}$
(x_0, y_0)
α_{wall}
$x_1 + Wall\_lead - y_0$



(x_0, y_0)
(x_1, y_1)
$y_1 - D_{wall}$
α_{wall}
$x_1 + Wall\_lead - y_0$



(x_0, y_0)
(x_1, y_1)