Andreas Döpkens, October 2014

[1.Rev. 2015-02-18]

There is no need to build a labyrinth,
when the entire universe is one.
(Jorge Luiz Borges)

The Wall Follower - directed by distance sensors

Driving a Bot through a Maze by just following the walls of the maze sounds easy. In a way it is not difficult to write the Program-Code, once you know what you are doing, meaning: once you understand the underlying Algorithm. From working experience with 15 years old students from high school we know, that eight to ten lines of C-Code can handle this job in a basic way pretty well. Having much fun, these really smart students just try a little bit here, try a little bit there, and after a couple of hours their Bots indeed do what they are supposed to do, namely: drive right-handed (or left-handed) along the walls through a 1,50 times 1,50 meters wooden Maze. If a Bot gets stuck, caused by whatever obstacles in their ways, the lines of code written by these students sometimes activate reverse gear to get rid of this problem, sometimes they turn the Bot on the spot until an exit is found out of the trap, and sometimes the Bots just sit and wait until someone human comes for help and removes the obstacle. Altogether, just by “playing” with the Bot, a simple and well working code can be created. But if you want to develop a more sophisticated solution especially with lack of redundancy, some intense thinking has to be done in advance. This article is meant to show you what has to be considered if you want to drive a Bot *effectively* and successfully through a Maze.

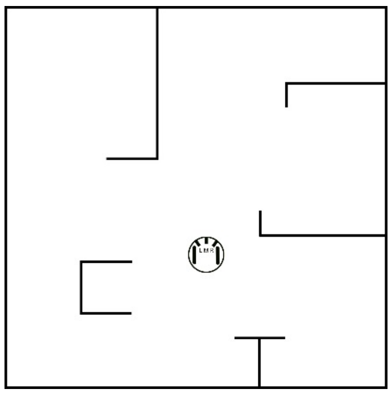


FIG. 1 „Little Universe“-Maze

Sensor Values and Their Meanings

The aCTino-Bot, a paradigm for any Teaching Bot in general, is equipped with three front distance-sensors **L**, **M**, and **R** (left, middle, right). Each of these sensors can have the following logical conditions:

+1 : “distance to obstacle/wall is fine”. There is a **MIN/MAX-Tolerance-Band** for the sensors before the walls of the Maze, aka ***Lead Trail Hysteresis*** (see FIG.2): a Sensor is nominated +1, if the measured value of the sensor lies between MIN (minimal required security distance of the Bot from the wall) and MAX (maximum allowed distance of the Bot from the wall). Therefore MIN is the lower value of the distance sensor, MAX the higher value.

0 : „obstacle/wall is far away” from the distance sensor of the Bot. The actual measured value of the sensor is higher (>) than MAX-Tolerance of the Hysteresis, and this means: action has to be taken (what the right action is will be discussed later in this article). Though the Bot is off Lead-Trail on the far side, this situation for the Bot is *uncritical*, it is not endangered to bouncing into the wall.

-1: “obstacle is too near” to the sensor. The measured value of the sensor is lower (<) than MIN-Tolerance, and this again means: action has to be taken. The Bot now is off Lead-Trail on the near side. This situation definitely is critical for the Bot, if it keeps driving without changing direction it will shortly crash into the wall.

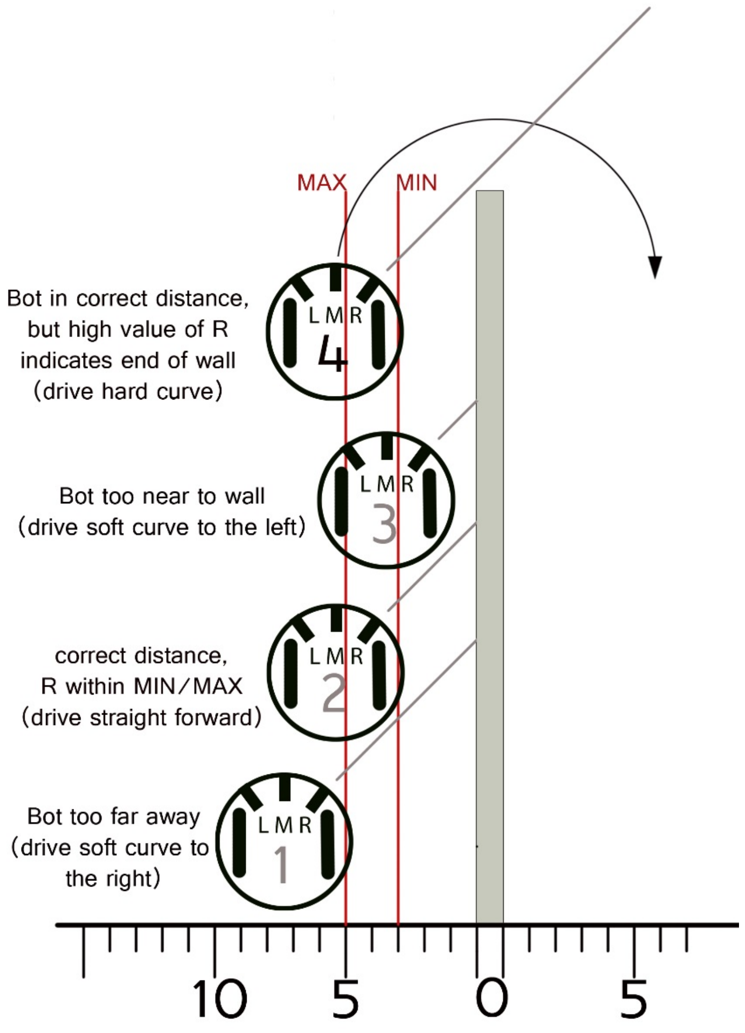

MAX
MIN
Bot in correct distance, but high value of R indicates end of wall (drive hard curve)
L M R
4
Bot too near to wall (drive soft curve to the left)
L M R
3
correct distance, R within MIN/MAX (drive straight forward)
L M R
2
Bot too far away (drive soft curve to the right)
L M R
1
10
5
0
5

FIG. 2 Lead Trail Hysteresis

Different Cases of Bot Positions

As the Figure below shows there are five different cases to distinguish, that each concern the “tour de Bot” through the Maze.

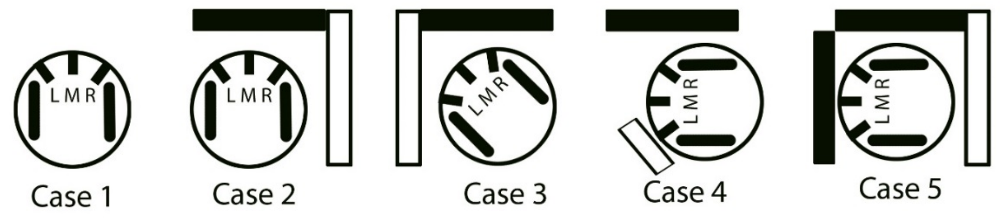


FIG. 3 Five Cases of Bot Positions Relative to Obstacle/Wall

Whatever the “legal” actual position of the Bot may be at the moment, in a *strict* right-hand driven Maze the value of L at no time is of any importance **(L = don’t care)**. In a certain way, we don’t need the distance sensor L at all in a right-hand driven Maze; technically we could remove L from the Bot without losing necessary function. But sometimes the going gets tough for the Bot (e.g. due to high speed the Bot cannot stop in time and therefore exceeds the minimum distance to a wall, or maybe another Bot in the same Maze gets too near to our Bot et cetera). In any of these cases it might be useful to have also sensor L to evaluate these “unplanned” situations for the Bot. This means: the logical conditions for the Bot’s navigation through the Maze is determined by M and R primarily, nevertheless L has a secondary function as will be described below in the *Sensors Case Table*. But first we are going to take a close look at the Logical M-R Table that gives an overview over the different Cases depending on the two sensors R and M. This table also helps to find redundancy that should be avoided in the Program-Code.

Table 1   Logical M-R Table

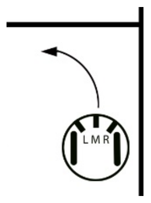
Logical M-R-Table for the different conditions of the sensors M and R		and R=		
		0	1	-1
if M=	0	C1 (B, A)	C4	C3
	1	C2	C2	C2
	-1	C5	C5	C5

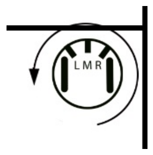
Read the table above like: if (M==0 && R==1) { /* see Case 4 (C4) below about what action has to be taken */ }

In the Table above the different Cases for the possible Bot positions in the Maze are shown. Each of these Cases requires a specific action for the Bot to enable it driving successfully through the Maze. What the specific actions and behaviors are, is described in the Sensors Case Table below.

Table 2 Sensors Case Table

Sensors Case Table	
Case 1	M=0 R=0 (L=X) **State B: Very Beginning** At the beginning of the program no obstacles are allowed in front of the Bot that could impede its drive. The Bot must be enabled to go straight forward until a wall of the Maze is found by either the sensors M or R. This requirement of course also goes for sensor L. **State A: Afterwards in The Program** If later in the program the logic condition of sensor R is 0 there can be two different reasons for it: ***A1:*** As Fig.1 shows the Bot is continuously striving to drive parallel to a wall of the Maze in order to stay within the Lead Trail Hysteresis (MIN-MAX boundary). Whenever in a right-hand Maze the Bot gets too far away from the wall, this is indicated by sensor R (the value of R is *a little higher* than MAX). The appropriate action to this situation is to get the Bot carefully a little closer to the wall again by driving a **soft curve** (omega app. -5 to -10°/s) **to the right** until the Bot is back inside Lead-Trail Hysteresis. (The counterpart of this State A1 in Case 1 is Case 3) Note: Experience has shown that Omega of the soft curve should not be a fixed value but dynamic to the actual value of R. To do a proper weighing a formula like SOFT_TURN(1+x(R-MAX)) will help (x being something like 0,1 … 0,3 [1/cm]).
	A2: Another situation for sensor R to show a value too high is the following: The bot could have reached the end of a right-hand wall, and now has to drive a **hard curve** (omega app. -70°/s) **to the right** until either M or R detect a new obstacle/wall again (see Fig.1). In this situation A2 the value of R is *not a little higher* than MAX like in A1, it is *very much higher* (as an estimated value let's say more than two times MAX) as shown by the dotted line in

	Fig.1. Note (see Note in Case 1A1): HARD_TURN(1+x(R-MAX))
Case 2	M=1 R=X (L=X) If M=1 then the logic condition of R is of no importance, it is “don’t care” (as Logic-M-R-Table shows, the logic condition of R never changes, it always is C2). R could be 1 relative to the right-hand wall, or it could be 0 relative to the corner or right-hand wall, and it doesn’t matter whether it is -1 (too near to a right hand wall). The Bot has to drive a **smooth curve** to the left.  *FIG. 4 Bot Driving Smooth Curve* Note1 (see Note in Case 1A1): SMOOTH_TURN(1+0,5(R-MAX)) Note 2: although R in this Case is generally considered *don’t care*, experience has shown that there should be some Priority, in which a “critical” sensor R (=-1) should rule over a “fine” M (=1). That means if R=-1 the Bot is not supposed to drive a smooth curve but rotate on the spot.
Case 3	M=0 R=-1 (L=X) This Case 3 reflects the counterpart of A1 in Case 1. While the Bot strives to drive alongside the wall within the MIN-MAX Range, it constantly has to correct its path. Now it gets too far away from the wall (=A1 of C1) then it gets too close (=C3) again. In A1 as a course of action the Bot had to softly turn right a little bit to get it back into the Lead-Trail Hysteresis because it got too far away from the wall, now in C3 its track has to be corrected by driving a **soft curve to the left**, because it got to near to the wall. Note1 (see Note in Case 1A1): SOFT_TURN(1+x(R-MAX)) with x=0,2/cm Note 2: R in this Case is considered *critical* (-1). But here there should be made a difference between just critical and “very critical” meaning R<MIN/2. If this is the case a soft curve to the left is not sufficient, a *rotate on the spot* is necessary.

	The situation described above implies, that there is no "white wall" (see Fig.3) and therefore L=0. **If however L=-1||1** (there is a white wall) the circumstances are different: In a right-hand Maze this situation can only result from some accident: somehow the Bot got stuck (maybe by somebody's "teasing" action). Now there is little or no chance to turn the Bot smoothly or softly to the left (doing so even endangers the Bot's R and L sensors to get even closer to the wall). So the only way for the Bot to get out of this trap is: it has to **rotate on the spot** (on its kinematic center, -> V=0 in the VOmega-interface) until at least R=1 again. Of course the Bot also could drive backwards, but this behavior is real dangerous, because there could be another white wall like in Case 4, and most notably the Bot's got no "eyes" at its back.)  *FIG. 5 Bot Rotating on the Spot*
Case 4	M=0 R=1 (L=X) In a right-handed Maze this situation is just fine. No action to correct the Bot's position has to be taken. So, **if L=0** everything is fine, the Bot just has to **go straight forward**. To facilitate this behavior, Omega in the VOmega-Interface has to be set to Zero! But what if someone put some obstacle like the white wall "diagonally/triangle-like" there just to tease the Bot? **If L=-1** (which means there is an obstacle very near to the left of the Bot) there is no escape for the Bot. The only possible behavior is: **Stop Bot and wait until obstacle is removed**.
Case 5	M=-1 R=X (L=X) This Case is similar to Case 3, except for the fact that not the R-sensor is critical, but the M-sensor. As is easily to be seen in Fig.3 there is only one reasonable behavior for the Bot: it can only rotate around its kinematic center to the left until set free again.