Line tracking sensors and algorithms

Posted on February 28, 2008, by Ibrahim KAMAL, in Robotics, tagged

Line tracking is a very important notion in the world of robotics as it give to the robot a precise, error-less and easy to implement navigation scheme.

As you may have seen, many robotics competitions promote this concept, by adding lines on the playground for the robot to follow, or sometimes, the sole purpose of the competition is to race with other robots following a line along a track.

In this tutorial, I am going to rely on the experience achieved by building the line sensors of the robots that participated to the robocon 2007 competition.

Number of cells in a sensor

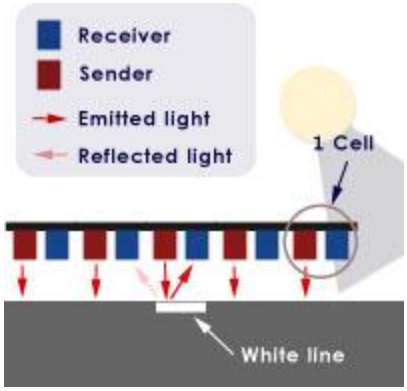

figure 1

As you can see in ***figure 1***, a line sensor is composed of a number cells and each cell is composed of a sender and a receiver. The particularity of this sender/receiver pair, is that it sends light that shall be reflected by the line to be detected but not by the eventually opaque background surrounding this line. Any sender/receiver pair that is able to make a difference between a line and the rest of ground (of a different color) can be used in a line sensor.

Usually, to make it easier on the designer of the sensor, there is an important contrast between the line and the ground (for example: white line on a dark blue ground), But in case there isn't enough contrast, there is a method to easily build a line sensor adapted to that specific situation, relying on old physics rules that states that a colored surface will absorb the light of different colors, and reflect the light of the same color. For example, If you want to build a line sensor to detect white lines drawn on a light blue floor, you can send red light, as the blue will absorb all of it, and the white line will reflect all of it. Actually this was the case in the playground of Robocon 2007 competition, there wasn't enough contrast between the white lines and the blue ground, so we had to use RED LEDs as senders instead of our preferred IR LEDs

So the first aspect that affects the precision and the quality of a line sensor, is the number of cells. Some roboticists use only two cells to know whether the line is at the left or at the right of the robot, but as you shall see later in the software part, this very poor source of information wont allow the controller to gradually guide the robot back on the track, instead you will notice that the robot will keep brutally turning right and left, but will never be able to smoothly follow the line. On the other hand, an eight cells line sensor will give a spectrum of relatively rich information to the controller, indicating whether the robot is very close to the line, close, far, or very far away. This variety of information will allow the controller to take actions that are proportional to the distance between the robot and the line, resulting in a smooth line tracking system.

Distance between the cells

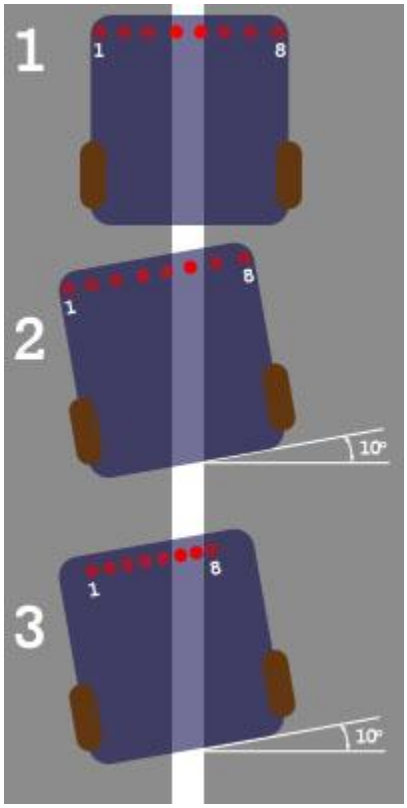

figure 2

The second aspect the be considered when building a line sensor, is the cell spacing (or the distance between a cell and the other). To understand the effect of cells spacing, consider the differential drive robot shown in ***figure 2***, with an eight cells line sensor, whose cells are numbered from 1 to 8 (from the left to the right). Three different situations are shown, In the first one, the cells 4 and 5 detect the line, indicating that the robot is perfectly centered on the line. In the first situation, the spacing between the cells is not very critical, but if the robot accidentally makes a 10° turn away from the line (second situation), you will notice that only the cell number 6 detect the line, which is the only indication that the controller will have about that 10° error. This means that, most probably, an error smaller than 10° wont even be noticed.

But in the third situation, the cells are closely collated together, and you can notice that with the same 10° deviation from the line, the sensor's cells 6 and 7 detected the line, leaving some other possible states in between the perfectly centered position and the 10° deviation. In other words, the closer are the cells from each others, the more will be the resolution of the sensor.

The same effect can be observed by changing the distance between the sensor and the center of steering. In general, It is important to always try to keep the sensor as far as possible from the center of steering, which is the back of the robot in a differential steering one, because this will also help to amplify the deviation detected by the sensor, resulting in a better response from the controller.

Building the sensor

There are many electronic components that can be used to build the sender/receiver cells of a line sensor. Two of them are discussed in this article, showing the advantages and disadvantages of each one, and showing how to implement each one of them in an electronic circuit.

IR LEDs

This method relies on our famous IR proximity sensor with some modification. It has the advantage of being cheap and easy to implement, but unfortunately need an important contrast between the line and the ground. Refer to the this tutorial for more information.

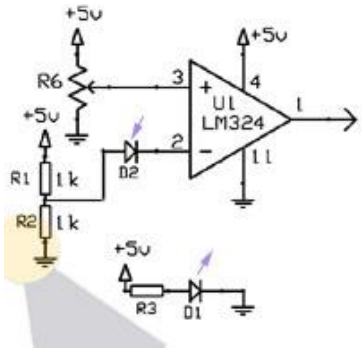

figure 3.A

LDRs and LEDs

When you need to adapt to low contrast situations, as discussed before, this is the most common alternative. You chose the most suitable color of LED for sending the light, then, the LDR will pick up the reflected light, but it's slower to respond than IR LEDs.

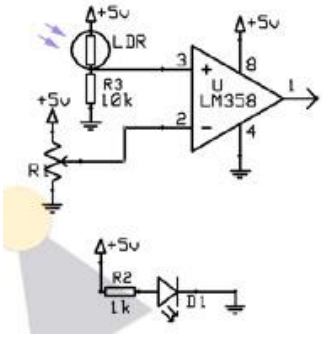

figure 3.B

D_1: Emitter LED **D_2**: Receiver LED — **R_6**: Sensitivity adjustment — **D_1**: Emitter LED — **R_1**: Sensitivity adjustment

After a lot of experiments, I personally recommend the LDR based line sensor because it can be easily adapted to many different environments by adjusting the sensitivity using the potentiometer R_1 or by changing the color of the LED D_1.

Here is the electronic circuit of the LDR based line sensor we used in our robots in the Robocon 2007 competition. As you can see it is composed of eight cells, each one resembling the cell in ***figure 3.B***. There are many reasons to choose to build a sensor with exactly eight cells, no more, no less: Eight can provide enough precision, it connects directly to one port of the microcontroller, and is represented by one

single Byte of data, making it easier to implement in the programming and in the memory of an 8 bit micro controller.

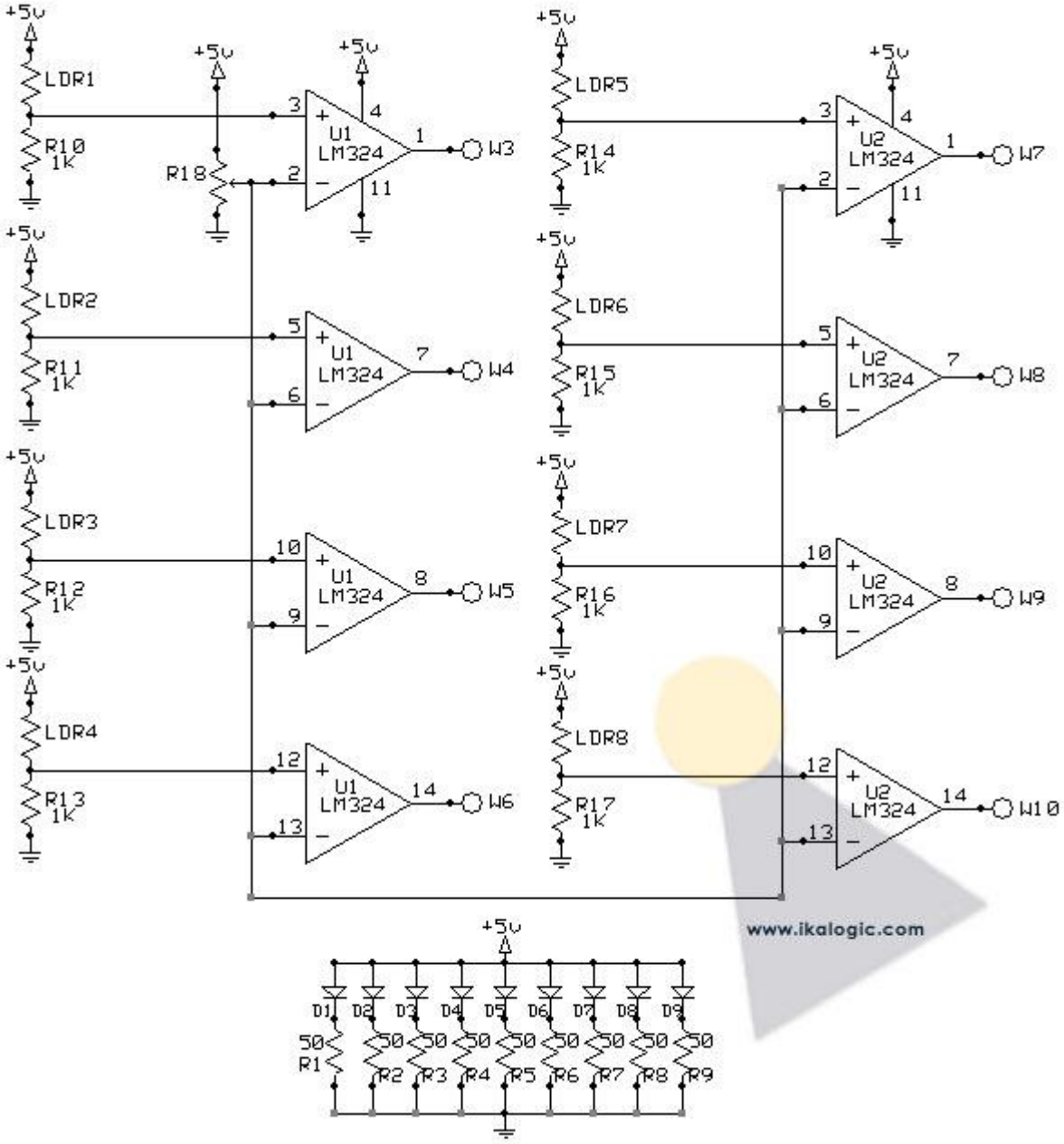

figure 3.C

The wire connections W_3 to W_{10} are the outputs of the eight cells of the sensor.

The value of R_1 to R9 cannot be lower than 50 ohm, actually this value is very low and that's why the sensor sinks a lot of current. You may try to use larger values first, like 220 ohm, then if the intensity of the light is not en ought, reduce it gradually.

You will also notice that there are nine sender LEDs (not eight), that's because the the LEDs and the LDRs are positioned in such a way that each LDR has one led on its right and another on its left (as you can see in ***figure 3.D***). The purpose of this technique is to make sure all LDRs share the same reflected light intensity, and this way, only one potentiometer can be used to calibrate all of them.

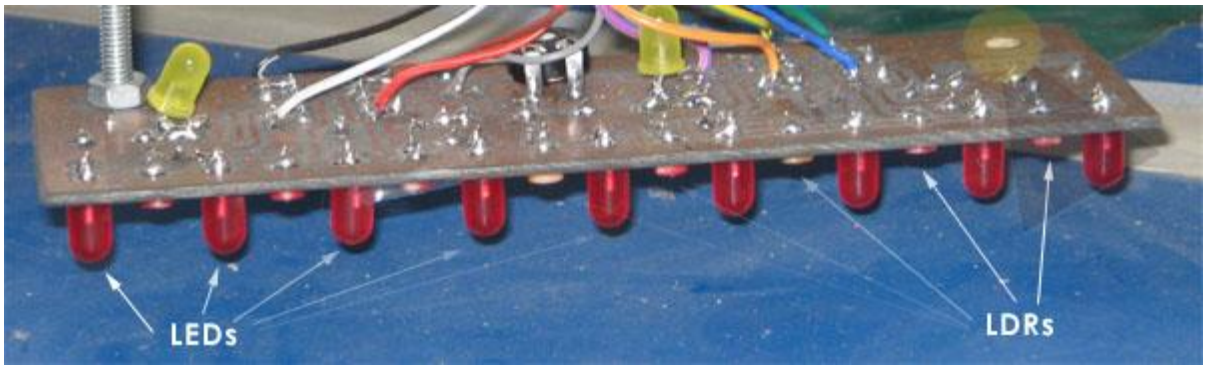

figure 3.D

Proportional Control Algorithms

Now that your sensor is working and is providing a correct reading of the line underneath it, you still need to develop some algorithms to use the data collected from the line sensor. The quality of those algorithms is as important – if not more important – than the quality of the sensor it self. Its those software procedures that will give to the robot the ability to smoothly and correctly track lines in a grid of lines and intersections, perform 90° turns and many others moves that can be implemented in such a lines grid.

Proportional Control, which is usually used in line following algorithms, means that the intensity of the rotation of the robot towards the line is proportional to the distance between that robot and the line. In other words, if the center of the robot is positioned exactly on the line, the rotation of the robot will be equal to zero, but if the robot gets deviated from the center of the line, the intensity of the rotation will gradually increase, until it reaches maximum intensity if the line is completely out of reach. This proportional Algorithm will prevent the robot from oscillating to the right and to the left of the line while trying to follow it.

What I mean by the intensity of rotation, is the speed at which the wheels will turn (in a differential steering robot) or the angle of the front wheel (in a car-like steering robot).

This may be true in theory, but in practice, due to the non-linearity of the behavior of DC motors, and many others sources of error that cannot be clearly defined, the robot would still oscillate while trying to track the line, and would sometimes fail, because the error would eventually increase instead of decreasing. That's why the proportional control scheme have to be tailored for each robot, depending on it's moment of inertia, on the type of motor, on it's weight and on many other factors. After lot of testing, the graph in ***figure 4.A*** shows a control scheme that proved to work correctly on most differential steering robots.

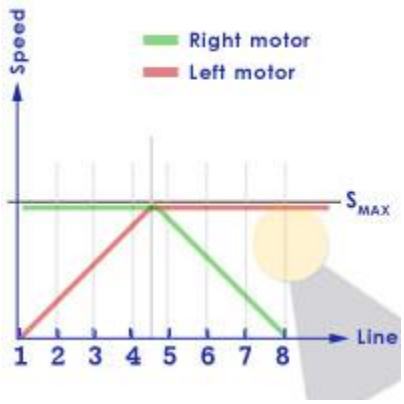

figure 4.A

Figure 4.A represents a relation between the speed that should be applied on the right and left wheels of a differential steering robot and the position of the line relative to the center of the robot. As you can see, for an 8 cell line sensor, the line is considered to be at the center of the robot when it reads 4.5, while it is considered to be totally at the left when the first cell of the sensor is detecting the line.

The only thing you may have to to do, is to define the value of Smax suitable to your robot. The easiest way to do this is by trial and error. You will probably notice that High values of Smax will result in very fast response, but with a lot of oscillations.

An important question is how to obtain analog readings from such a digital output line sensor? The answer is we actually don't obtain real analog signals, we just calculate an average of the position of the line, when more than one cells detect the line. For example, when cells number 4 and 5 detect the line, the average of 4 and 5 is 4.5, and we will consider this value as the reading of the line to be used in the graph of the ***figure 4.A***. Depending on the thickness of the line being tracked, you can optain a multitude of readings between a integer and the other.

In order to precisely control the speed of the motors in a differential drive robot, you need to adapt what is called closed loop speed control of DC motors, which is explained in detail in this tutorial.

For a 8051 microcontroller programmed in C, here is an example source code of a function named **follow_line()** which when called, reads the value of the sensor which is connected to port 0, calculates the average then deduces the required speed of the right and left wheels to smoothly adjust the robot to the line.

```
ollow_line(){

max_speed = 8;

half_speed = 4;

line_to_speed_factor = (max_speed) / 4.5;
```

```
                    //The line sensor is connected to P0
    if (P0 != 0    ){     //Keep the old line reading in case the line is
lost
    old_line = P0;
    }
    new_line = P0;
    l1 = P0_0;        //Store the values of each cell of the 8 cells of the
    l2 = P0_1;        //line sensor in the variables l1 to l8.
    l3 = P0_2;
    l4 = P0_3;
    l5 = P0_4;
    l6 = P0_5;
    l7 = P0_6;
    l8 = P0_7;
    fwd();         //Call a function that orders the robot to move forward


    if (P0 == 0){    //In case the line is out of reach, rely on the last
valid
        if (old_line > 45){    //reading to decide whether to pivot right or
        pivot_left();        //left to reach the line again.
          req_right_pulses = max_speed;
        req_left_pulses = max_speed;
        }else{
        pivot_right();
          req_right_pulses = max_speed;
        req_left_pulses = max_speed;
        }
    }else{
```

```
        if(old_line != new_line){
        //Calculate the average reading of the line.
            line = (l1) + (l2*2) +(l3*3)+(l4*4)+(l5*5)+(l6*6)+(l7*7)+(l8*8);
            line = line / (l1+l2+l3+l4+l5+l6+l7+l8);
            //Calculate the required right and left speed
            //according to the graph.
            req_right_pulses_ = floor((line*line_to_speed_factor)+0.5);
            req_left_pulses_ = floor(((9-line)*line_to_speed_factor)+0.5);

            if (req_left_pulses_ > max_speed){
                req_left_pulses = max_speed;
            }else{
                req_left_pulses = req_left_pulses_;
            }

            if (req_right_pulses_ > max_speed){
                req_right_pulses = max_speed;
            }else{
                req_right_pulses = req_right_pulses_;
            }
        }
    }
}
```

Note that this code is not stand-alone, it is a part of more complicated program that contains the the closed loop speed control and many other functions allowing the robot to navigate according to a specific path. for example, the values ‘req_left_pulses’ and ‘req_right_pulses’ have to be fed to the closed loop speed controller.

You will also notice that the speed is calculated in two steps, the first result is stored in ‘req_right_pulses_’ then the final result is stored in ‘req_right_pulses’. This is because the graph in ***figure 4.A*** is composed of two independent linear relations, the first is for the readings from 1 to 4.5, and the

other relation is for the rest of the readings, 4.5 to 8, (and the same applies to the 'req_left_pulses' variable). This is just an example, there are many ways to implement such a graph into a microcontroller program, it's up to you to see the most suitable method according to the architecture and organization of your program.

Navigation through lines and intersections

figure 5.A

Now that you know how to make your robot follow a line, you can use that same sensor to allow it navigate through a grid of horizontal and vertical lines as the one in ***figure 5.A***, using the same 8 cells sensor.

The main clue to an errorless navigation in such a maze, is to be able to precisely detect intersections. To do that, first you have to analyze the nature of those lines, the angle of intersections, and the different readings of the line sensor when crossing intersections. Actually, you have to adapt your code to each and every playground you expect you robot to navigate on.

After a lot of testing we developed this simple technique to detect intersections, whatever the way the robot crosses it.

As you can see in ***figure 5.B***, three different situations are shown, in each one of them, the robot crosses an intersection, coming from a different angle. The cells of the line sensor that detect the line are designated by bright red spots, while cells that don't detect it are designated by dark red spots.

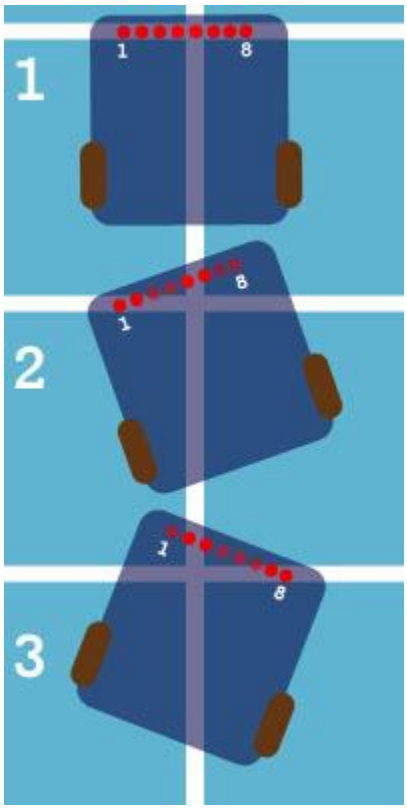

figure 5.B

What we tried to do is to find what is common between those three different possibilities, and the following rule was developed to detect intersections:

'If one of the end cells (one or eight) detects the line while one or more of the last 4 cells at the other end also detect the line, then the sensor is crossing over an intersection'

In other words, for an intersection to be validated, the reading of the sensor must be as follow:

Cell number 1 detect the line AND one or more of the cells 5 to 8 detect the line

OR

Cell number 8 detect the line AND one or more of the cells 1 to 4 detect the line

Then you have to develop the code that will analyze the readings of the sensor, count intersection, and guide your robot through the desired path, which can be done with a multitudes of methods. The choice of the method to guide a robot, and precisely localize it in a map can be very difficult task, even if you are using line following algorithms. Some methods will even involve a combination of dead reckoning and line following to achieve more accurate results. Generally, it's your job to design the navigation scheme which

is most suited to the environment of the robot. It's important to note that robot navigation is subject to many research and is still in an intensive development phase in the robotics labs around the world.

I hope this article covered the main aspects required to construct a simple robot navigation system based on line following algorithms and helped to introduce some of the scientific principle behind the operation of such a system.

Source: http://www.ikalogic.com/line-tracking-sensors-and-algorithms/