P:Y:R

Program Your (Botball) Robot

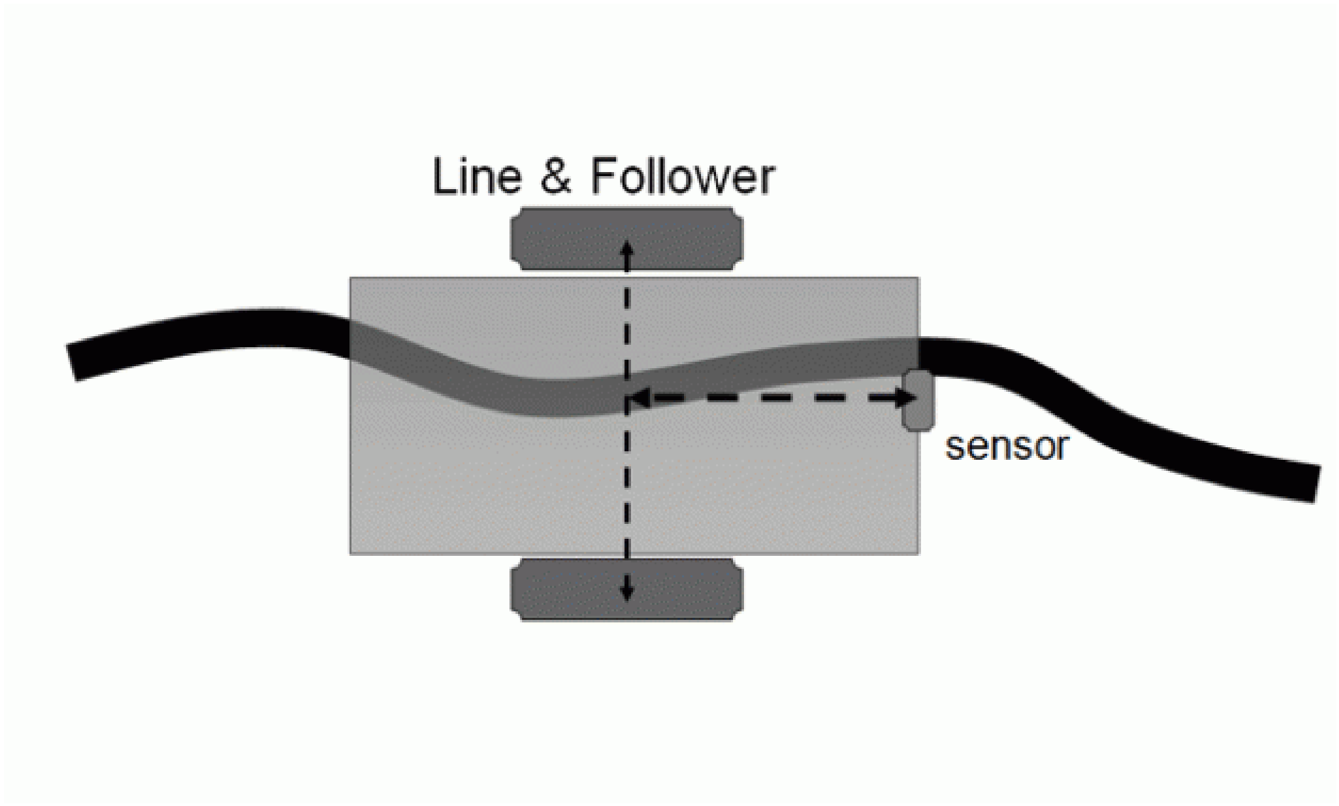
Line Following Introduction

Many introductions to robotics programming include an example of “Line Following” code using one or two sensors. This may seem appropriate since the sensing is simple and the code can be written with very few lines. However, this behavior is *not* simple and describing the task is not easy. [good programming must start with a well-defined objective.] If you search the web for examples you will find many: from reports by 5th grade students using a Mindstorms kit, to descriptions of major open competitions and various winning strategies. An unconstrained requirement to “do rapid, accurate line following” is an invitation to a whole range of topics, most of which are beyond the concepts for beginners.

Let’s try to explain the **basic line-following behavior using a single reflectance, or ‘proximity’ sensor on a two-wheel, drive-steered robot**, assuming a single smoothly-curving black line (3/4” electrical tape) on a white background. The task implies a behavior where the robot moves forward while guiding itself to follow the curve of the line.

A drawing showing the sensor, pointed down at the path and its location in relation to the drive wheels is important to understanding the task.

With a single sensor, the guiding signal is the difference between the black line and the white background. The objective would be to maintain a ‘gray’ level indicating that the sensor is at the edge of the line, following either the right or left edge as the robot moves forward. Let us assume a configuration as shown from a top-down view:



Two dimensions are important to guiding the robot, the wheel-base, or axle, and the distance from the sensor to the axle. As the robot turns slightly, one wheel moves ahead of the other, and the sensor moves sideways, toward or away from the line, in proportion to ratio of the dimensions D/A, where D is the distance from the axle, and A is the length of the axle. (If the sensor is closer to the axle, the sideways motion will be smaller.)

Now we can define the behavior in more detail: The robot must be controlled to move forward *and* to turn as directed by the proximity sensor. If the sensor can only indicate “on” or “off” the line, the response is always to turn toward, or away from the line (this control is referred to as “bang-bang”). The turn is the tricky part and requires a control loop to rapidly decide the magnitude and direction. The rate of turning can be adjusted, at sample intervals, in response to the curvature of the line with a magnitude determined (by design and in advance) by the speed forward, the angle of the line change, and the D/A ratio (or simply a right/left decision) based upon each sample value. Finding the optimum control is the subject of a course in feedback control, but a simple, workable method can be found by trial and error: If you have too much turning the robot will be moving back-and-forth across the line faster than it is moving forward. And furthermore, it may jump beyond the line edge if the turning movement (or line change) is too large during one sample interval. If you have too little turning the robot will follow slight line curves, but not keep up with a sharper curve, and thus will lose the line edge.

Various non-linear strategies can be tried to allow more drastic turns only when needed, such as: ***1***.pivoting or spinning if the line edge isn’t found in one or two samples (measured by sample counts or clocks) ***2***. slowing the forward motion if the line edge hasn’t been found quickly ***3***. Turning a fixed amount and moving forward a fixed maximum amount if the line edge hasn’t been found quickly. Etc. [This, along with the infinite range of D/A ratios and speeds, and the variables in line extremes, leads to many opportunities for creative designs and also many ‘Line Following’ competitions.]

Another issue, which the developer will soon be faced with, is the context of the desired line-following behavior. In a Botball competition line-following is only the selected means to the end of arriving at a scoring object or a target location. How does the robot transition to and from line-following? To answer this question a method for creating the 'start' and 'stop' of the line-following behavior must be developed. Often an additional sensor is sampled to decide.

Below is a simple program to calibrate & test a ***timed*** line follower with a single sensor and bang-bang control (deciding simply to turn right or left at a fixed rate):

```
// Timed line follow 8/8/12; mod for CBC or Link 8/20/15 tlg
 //assume left motor is #3, right motor is #0
 //proximity sensor is analog(3) [an 8 bit CBC A/D; for Link substitute 'analog8(3)
 //D/A ratio for test robot is 1/4;
 // the sensor is much closer to the axle than in the diagram in the introduction.
 //2-d add loop counter to 180 (~ 5 sec with sample loops on CBC), then end
 //2-e add back the clock reading & print run time
 /*4-a add start mode: -run straight until sense line, then turn ~60 deg, run
  straight at 1/2 SP for a short distance, to merge w. line, then begin line-
  following loop for a fixed # of samples */
```

```
 #define SP 583 //maximum forward speed to which 'del' can be added for turns
 #define del 416 // trial delta in speed needed to turn on a test line
 #define left 3 // motor port #'s
 #define rt 0
```

```
int main()
 {
 int mid =110, i=0; // average sensor level between dark & light for 8 bit A/D
 // for Link multiply mid by 4 ('analog(3)' is 10 bits) or substitute 'analog8(3)'
 float startt; // time ref
 printf("start pointing at line, then press B -4a\n");
 while(b_button()==0); //wait for B press
 while(analog(3)<mid){//move forward until line is sensed
    mav(left,SP);
    mav(rt,SP);
    }
 mav(rt,-SP/2); mav(left,SP/2);//spin to the right
 msleep(1900); ao(); // turn ~ 60 deg
 //while(1); //test for angle & set sleep time above to spin 60 deg
 mav(rt,SP/2); mav(left,SP/2);
 msleep(2000); //go straight for a short distance to merge w. line
 startt = seconds();
 while (i<180) // count # of samples to get to end of line
    {
     if (analog(3) < mid )// too white
       { // lower speed forward on left, raise on right: turn left
        mav(left,SP-del); mav(rt,SP+del);
       }
     if (analog(3) >= mid ) // too black
       { // lower speed forward right, raise on left, turn right
        mav(left,SP+del); mav(rt,SP-del);
       }
     i++;
    //if (seconds()-startt > 23.)break ; //alternate test for end time (23 sec)
    }
 ao();
 printf("finish ,%d loops - %.2f sec",i, seconds()-startt);
 //show samples & time for run; a measure of the sample rate
 }
```

Line Following Development