**The Line Follower**

Last time in class we created several versions of a “line follower” program. Here is a review of what we did so that you can create your own LineFollower robot.

**The Original Idea**

We need a color sensor in front of our robot, pointing down. Of course that implies that we define two fields for the motors, and three fields plus one method for the light/color sensor.

The original algorithm is simple: place robot parallel to black line such that the light sensor is right on the edge of the line. Then: if the light sensor remains on the edge, go straight; if the light sensor is on the black portion, curve slightly *away* from the black line; if the light sensor is over the white portion, curve slightly *towards* the black line.

To make this work, we first needed to teach the robot “white” and “black” and store the corresponding light intensity values in fields named – what else – black and white. Finally, we computed the middle between those values and store it in a third field named zero. We will create a method that does all of this inside a new class called ‘LineFollower’:

 // Method to learn the values for 'black' and 'white'. Assumes fields ‘black’, ;white’,

 // and ‘zero’ are defined as well as a method ‘getLightValue()’. Fill in the code:

 **public** **static** **void** learnBW()

 {

 // Printing 'black' on the screen to get the user to place sensor on top of the black line

 // Waiting for user to press ENTER to continue

 // Getting the light value from the sensor and storing it in the field 'black'

 // Repeat this code to learn the value of 'white'

 // After getting 'black' and 'white', compute 'zero' as the average of those two values.

 // For additional info, you could (and should!) print these value on the screen.

 // In any case, wait for the user to press ENTER again to give them time to position the

 // robot with the sensor just on the edge of the black line before continuing.

 }

Here is the outline of the rest of the LineFollower class. It shows how to get the color sensor to switch into light intensity mode, but much of the rest of the code is for you to fill in:

**public** **class** LineFollower

{

 // Standard fields for the motors (make sure they match your connections)

 **public** **static** EV3LargeRegulatedMotor *motorLeft* = **new** EV3LargeRegulatedMotor(MotorPort.***A***);

 **public** **static** EV3LargeRegulatedMotor *motorRight* = **new** EV3LargeRegulatedMotor(MotorPort.***D***);

 // Standard framework for a sensor. Note that we are using "getRedMode" to switch our color

 // sensor into a light intensity sensor. In this mode it will no longer return a color ID

 // but instead a value that correspond to the light intensity measured by the sensor. Dark

 // colors such as black or brown mean low intensity (a small number) while bright colors

 // such as white or yellow will have higher intensity (larger values).

 **public** **static** EV3ColorSensor *lightSensor* = **new** EV3ColorSensor(SensorPort.***S3***);

 **private** **static** SampleProvider *lightProvider* = *lightSensor*.getRedMode();

 **private** **static** **float**[] *lightData* = **new** **float**[*lightProvider*.sampleSize()];

 // the values for 'black' and 'white' light intensities that we will teach our robot

 **public** **static** **double** *black* = 0;

 **public** **static** **double** *white* = 0;

 // The value between black and white, which gets computed later, and which we will interpret

 // as 'zero' color, i.e. neither white nor black.

 **public** **static** **double** *zero* = 0;

 // the usual method to query our sensor

 **public** **static** **double** getLightValue()

 {

 *lightProvider*.fetchSample(*lightData*, 0);

 **return** *lightData*[0];

 }

 **public** **static** **void** learnBW()

 { // as defined above }

 **public** **static** **void** main(String[] args)

 {

 // Leaning white and black values by calling the corresponding method

 *learnBW*();

 // Now starting a while loop as long as the ENTER button is up:

 // Get the current light value, subtract 'zero' from it, and store result. Then:

 //

 // if (result < 0)

 // we detected black: set speed of left, right motor to pull to the right

 // else if (result > 0)

 // we detected white: set speed of left, right motors to pull to the left

 // else

 // we are right on the edge of the black line: set speeds to go straight

 //

 // Engage both motors to move forward

 // End of while loop

 *motorRight*.stop();

 *motorLeft*.stop();

 }

}

Now, after you filled in the missing code (without errors, hopefully), it is time to test it. On the page below are some black line segments. Print them out, perhaps multiple copies of it, arrange the pieces using scotch tape to form longer segments, including some curves. Then use this line to test your program: first, teach it black and white, then place the robot parallel to your track with the sensor just on one edge of the black line. If everything works, your robot should now follow the line. If it seems to move *away* from the line, place the sensor on the other edge of the line.

If your robot can follow a straight line, but not any curves, adjust the speeds until it can handle the curves. The forward speed can’t be too large, while the difference in speed for pulling left or right needs to be sufficiently large. Make sure to use named constants in the final version of your code.

**Improved Version**

If your program works correctly, your robot should follow a line but if you observe it carefully, you will notice that it never actually drives straight, it always curves either left or right. That is because that portion of the code only executes if the light value is *exactly* zero which is rarely if ever the case. To allow for a bigger range, change the if-else statements in the main function like this:

 //

 // if (result < -CUT\_OFF)

 // we detected black: set speed of left, right motor to pull to the right

 // else if (result > CUT\_OFF)

 // we detected white: set speed of left, right motors to pull to the left

 // else

 // we are right on the edge of the black line: set speeds to go straight

for some suitable value for CUT\_OFF. Experiment: if CUT\_OFF is too large, your robot will move straight and not follow any curve at all; if it is too small, it will not got straight even when following a straight line. Note that CUT\_OFF could very well be a decimal number, even one that is less than 1.0.

Finding the right value for CUT\_OFF should speed up your robot quite a lot and “calm it down” nicely along the straight portions of your line.

**Proportional Version**

The improved version should work pretty well, but with a little thought we can improve on it even more while at the same time simplifying our code. Let’s review what the light sensor sees: it registers light intensity from black (low intensity) to white (high intensity). After we subtract the learned value of ‘zero’, we have that the result is negative over a darker portion of the surface and positive over a lighter one. In fact, the more negative the result, the ‘blacker’ the surface and the more positive it is, the whiter the surface. Thus, our robot should curve more the larger the result is, and the direction should depend on the sign of the result. Thus, we can replace the entire if – else if – else structure above simply by the following code:

// Get the current light value, subtract 'zero' from it, and multiply the difference

// by a constant KP (constant of proportionality). Store the result in a variable

// ‘correction’. Then:

*motorRight*.setSpeed((int)(SPEED – correction));

*motorLeft*.setSpeed((int)(SPEED + correction));

// engage both motors to move forward

Of course this requires that you define KP and SPEED as constants. Try, for example KP = 200 and SPEED = 300. Compile and test. If the robot does *not* follow the curve you should increase KP and/or decrease SPEED, if it wobbles too much, do the opposite.

So, I hope this gives you an idea about completing this challenge. If not, send me email with any questions.

**Print, Cut Out, and Tape Together**