

LESSON 8

Line Following

Line following is a very important and rewarding activity in robotic. There are many real-world applications for line following and understanding

There are several competitions focusing on line following. You can use one, two, three or even more sensors for line following. In this lesson, we will be using three light sensors. Before we attempt to do line follow, the line sensors should be assembled and tested.

Check the wiring connections on your Pi-Bot to ensure the HY Studio tracking sensor (line sensor) is correctly installed. The three center wires should be connected to the microprocessor, and the two outside wires should be connected to the breadboard, using five (5) 30cm wires, as discussed in Lesson 2: Building Your Pi-Bot.

Here is a brief overview of the line sensor array wiring:

- VCC – Red to breadboard VCC (5V Power)
- L – Yellow to STEM Board Pin 7
- C – Green to STEM Board Pin 8
- R – Blue to STEM Board Pin 10
- GND – Black to breadboard Ground

See figure 8.1 for the correct orientation of the line sensor. Make sure the 3 sensors are pointing down and the red LEDs are pointing up!

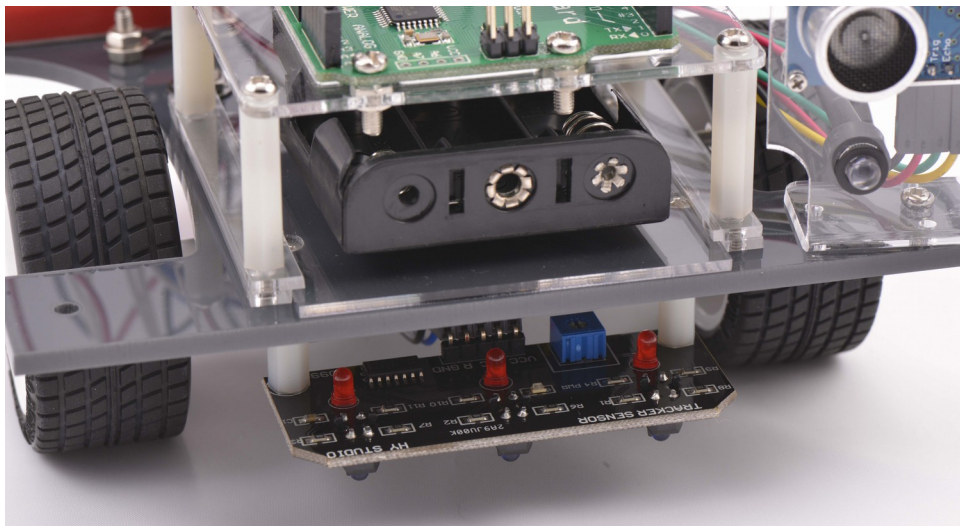


Figure 8.1

Your line sensor has been correctly installed and is now ready for testing!

Programming for Line Following

We will now discover how the line sensors will allow your Pi-Bot to detect and track a line.

Line following consists of two parts:

1. Determining the Position of your Pi-Bot to a Black Line

First, your Pi-Bot determines its position relative to a black line with the use of line sensors, located in the front. When the line sensor detects black, sensor outputs “1” indicating that most of the light is absorbed. When the line sensor detects white, the sensor output is “0”

By using three line sensors, your Pi-Bot will know whether it is on a black line (left and right sensors output “0” and the center sensor outputs “1”), to the left of a black line, or to the right of a black line.

2. Motor Control

With this information, your Pi-Bot will control the speed of each motor individually to try and get back over the line.

Figure 8.3 illustrates how the sensors detect a line.

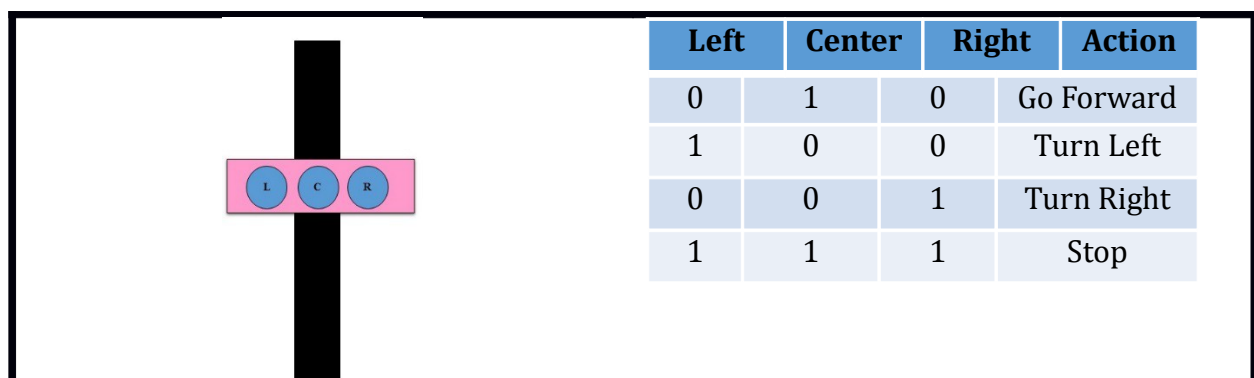


Figure 8.3

Run the program: line following 01.ino, as shown in figure 8.4.

This is a simple program that uses just the outer two detectors on the line following sensor. The middle detector is ignored. When neither sensor detects the black line, it assumes that the sensor is centered over the line and a fast speed is applied to both motors. If either of the sensors detects the line, the program corrects by applying a fast speed to the motor opposite the detected line and a slow speed to the motor closest to the detected line. This will cause your Pi-Bot to always drive into the direction of the line. For mild curves and straight lines, this method works very well. For sharp curves and corners, your Pi-Bot may get off track and have difficulty getting back to the line.

This program provides a good starting point for you to develop your simple controller. You may need to adjust the fast and slow speeds depending on the curvature and width of your line.

To begin, manually move your Pi-Bot over a black line. Do not turn the motor control power switch yet! As the sensors cross over the line, you should see the red LEDs on the board turn on and off.

Note: If the red LEDs do not turn on or off as the sensors cross the line, you may need to adjust the sensitivity of the sensors, using the built-in potentiometer. If nothing turns on, check your wiring connections.

With that being successful, place your Pi-Bot over the black line and turn on the motor control power switch.

Your Pi-Bot should track and follow the black line!

```

// line_following_01.ino

const int Line1 = 7;    // Left Line Sensor
const int Line2 = 8;    // Center Line Sensor
const int Line3 = 12;   // Right Sensor

const int In1 = 3;      // In1
const int In2 = 5;      // In2
const int In3 = 6;      // In3
const int In4 = 11;     // In4
const int batPin = A0;

#define num 6

void setup()
{
    // initialize the pins
    pinMode(In1, OUTPUT);
    pinMode(In2, OUTPUT);
    pinMode(In3, OUTPUT);
    pinMode(In4, OUTPUT);
    pinMode(Line1, INPUT);
    pinMode(Line2, INPUT);
    pinMode(Line3, INPUT);
}

void loop()
{
    int fast, slow;
    int speed1;
    int speed2;
    int L1, L2, L3;
    fast = 125;
    slow = 20;
    while(1==1)
    {
        L1 = digitalRead(Line1);
        L2 = digitalRead(Line2);
        L3 = digitalRead(Line3);
        if (L1 == 1)
        {
            speed1 = fast;
            speed2 = slow;
            forward(speed1, speed2);
        }
        else if (L3 == 1)
        {
            speed1 = slow;
            speed2 = fast;
            forward(speed1, speed2);
        }
        else
        {
            speed1 = fast;
            speed2 = fast;
        }
        forward(speed1, speed2);
    }
}

void reverse(int speed1, int speed2)
{
    analogWrite(In4, speed1);
    analogWrite(In3, 0);
    analogWrite(In2, speed2);
    analogWrite(In1, 0);
    return;
}

void forward (int speed1, int speed2)
{
    analogWrite(In4, 0);
    analogWrite(In3, speed1);
    analogWrite(In2, 0);
    analogWrite(In1, speed2);
    return;
}

void stopNow()
{
    analogWrite(In4, 0);
    analogWrite(In3, 0);
    analogWrite(In2, 0);
    analogWrite(In1, 0);
    return;
}

```

Figure 8.4: Program – line_following_01.ino

Improve the ability of your Pi-Bot using the following exercises.

Exercise 1

- Write a program to follow a line with just one light sensor (use L, C, or R).

Exercise 2

- Compare the performance of the single detector output to the program: line following 01.ino, as shown in figure 8.4.

More advanced programming can be created to allow your Pi-Bot to operate more precisely. The previous program does not utilize the center sensor

Run the program: line following 02.ino, as shown in figure 8.4.

This is an example of a more sophisticated line following algorithm. While your Pi-Bot will not run any faster using this program, it is more reliable and can easily track around sharp corners.

The program continuously samples each of the three line sensors and stores the latest N samples for each detector where N can be any integer value between 2 and 50. The samples are stored in three arrays, one for each detector. After each set of samples are read, the program places the samples on the top of the arrays and shifts the current samples down for each array. The samples at the bottom of the arrays are discarded. After the arrays are shifted down and the latest samples are placed on the top of the arrays, a weighted sum is generated from the arrays for each of the detectors.

Basically, instead of having an on/off value for each detector, there are values related to how long your Pi-Bot has been off the line. The longer it has been off the line, the larger the differential speed is between the wheels to get your Pi-Bot back on track.

In the program, N is set to 8. The speed value for each motor is controlled by the ratio of the weighted sum of the side detectors to the center detector. This algorithm is very sensitive to the motor voltage supplied by the 3 AA batteries. As the battery voltages lower with use, or if you are using rechargeable NiCads batteries, the performance changes. To minimize this effect, the program reads the motor battery voltage using the analog input and adjusts the PWM voltage accordingly. **THEREFORE, YOU MUST RUN A JUMPER FROM THE BREADBOARD, H-BRIDGE PIN 15 (MOTOR BATTERY POWER) TO A0 OF THE ANALOG INPUT.** Without the jumper, the program will not work correctly. Also, this will not fix dead batteries. If your batteries cannot supply enough current to run the motors, no programming will help! As you familiarize yourself with the program, change the values to experiment. You can use this program as a starting point or just as a learning tool to develop your own (and better) control program.

```

// line_following_02.ino

const int Line1 = 7;    // Left Line Sensor
const int Line2 = 8;    // Center Line Sensor
const int Line3 = 12;   // Right Sensor

const int In1 = 3;      // In1
const int In2 = 5;      // In2
const int In3 = 6;      // In3
const int In4 = 11;     // In4
const int batPin = A0;

#define num 6

void setup()
{
    // initialize the pins
    pinMode(In1, OUTPUT);
    pinMode(In2, OUTPUT);
    pinMode(In3, OUTPUT);
    pinMode(In4, OUTPUT);
    pinMode(Line1, INPUT);
    pinMode(Line2, INPUT);
    pinMode(Line3, INPUT);
    Serial.begin(9600);
}

void loop()
{
    int rLED[num], cLED[num], lLED[num];
    int lSum, cSum, rSum;
    int fast;
    int speed1;
    int speed2;
    float scale1, scale2, scale3;
    boolean debugPrint;
    int L1, L2, L3;
    int batVoltage;

    for(int i=0; i<num; i++)
    {
        rLED[i]=0;
        cLED[i]=1;
        lLED[i]=0;
    }
    rSum = 0;
    cSum = 0;
    lSum = 0;
    //***** Turn on debugging here *****
    //debugPrint = true;
    debugPrint = false;
    //*****
    delay(100);
    forward(0, 0);
    while(1==1)
    {
        batVoltage = analogRead(batPin);
        L1 = digitalRead(Line1);
        L2 = digitalRead(Line2);
        L3 = digitalRead(Line3);
        if ((L1 == 1) && (L2 == 1) && (L3 ==1))
        {
            speed1 = 0;
            speed2 = 0;
            forward(speed1, speed2);
            if(debugPrint)
            {
                Serial.println("In Lift Mode");
            }
        }
        else if ((L1 != 0) || (L2 != 0) || (L3 !=0))
        {
            rSum = 0;
            cSum = 0;
            lSum = 0;
            for(int i=0; i<num-1; i++)
            {
                rLED[i]=rLED[i+1];
                cLED[i]=cLED[i+1];
                lLED[i]=lLED[i+1];
            }

            rLED[num-1]=L1;
            cLED[num-1]=L2;
            lLED[num-1]=L3;
        }
    }
}

```

```

        for(int i=0; i<num; i++)
        {
            rSum = rSum + rLED[i] * (8-i);
            cSum = cSum + cLED[i] * (8-i);
            lSum = lSum + lLED[i] * (8-i);
        }

        if(debugPrint)
        {
            Serial.println("");
            Serial.print(L1);
            Serial.print(", ");
            Serial.print(L2);
            Serial.print(", ");
            Serial.println(L3);
            Serial.print(rSum);
            Serial.print(", ");
            Serial.print(cSum);
            Serial.print(", ");
            Serial.println(lSum);
        }

        scale1= (float) rSum + (float) cSum;
        scale2= (float) lSum + (float) cSum;
        scale3= (float) lSum + cSum + rSum;

        fast= 80000. / (float) batVoltage;

        if(rSum > lSum)
        {
            speed1 = fast;
            speed2 = scale2/scale1 * fast;
        }
        else
        {
            speed2 = fast;
            speed1 = scale1/scale2 * fast;
        }

        forward(speed1, speed2);

        if(debugPrint)
        {
            Serial.print(scale1);
            Serial.print(", ");
            Serial.println(scale2);
            Serial.print(speed1);
            Serial.print(", ");
            Serial.println(speed2);
            Serial.print("battery Voltage = ");
            Serial.println(batVoltage);
            Serial.print("fast = ");
            Serial.println(speed2);
            delay(500);
        }
    }
}

void reverse(int speed1, int speed2)
{
    analogWrite(In4, speed1);
    analogWrite(In3, 0);
    analogWrite(In2, speed2);
    analogWrite(In1, 0);
    return;
}

void forward (int speed1, int speed2)
{
    analogWrite(In4, 0);
    analogWrite(In3, speed1);
    analogWrite(In2, 0);
    analogWrite(In1, speed2);
    return;
}

void stopNow()
{
    analogWrite(In4, 0);
    analogWrite(In3, 0);
    analogWrite(In2, 0);
    analogWrite(In1, 0);
    return;
}

```

Figure 8.4: Program – line_following_02.ino

As you can see, changing the control program causes your Pi-Bot to behave very differently!

Exercise 3 (Advanced)

- Combine the line following with the collision detection. Your Pi-Bot should follow a black line, until an obstacle is placed in front of it, at which time it stops.

Did your Pi-Bot perform successfully? If so...

Congratulations! You have successfully programmed your Pi-Bot!

