Hands-on Lab

Line Following

In a previous lab, the Lego NXT light sensor was introduced. This sensor doesn't measure color per se, but rather (reflected) intensity. By measuring the intensity of line's outermost edge, a robot can follow the line. Three concepts are sensor calibration, and then line following using bang-bang and proportional control approaches.

Concept 1: NXT Lego Light Sensor Calibration

With a black line on a white mat, the human eye may only distinguish 2 colors. However, a light sensor can actually measure gradients (i.e. a range of greyscale values) as it passes from the black line to the white background. The gradients depends on one's sensor, the mat's paper quality (e.g. glossy versus flat), and ambient light (e.g. fluorescent versus incandescent or sunlight).

Step 1: First construct a light sensor that mounts on the Domabot's front (see **Figure 1-1** below).

It's better if it's centered and about 2-mm (or the thickness of 2 pennies) from the ground. If the sensor is too close to the ground, the transmitted light will be obscured by the ground. The result is that the receiver won't capture the transmitted light well. If the sensor is too high from the ground, the transmitted light might become diffuse (i.e. sprayed out). Again, the receiver might not capture enough transmitted light.

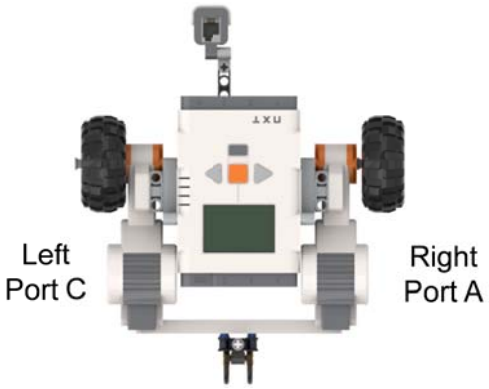

Figure 1-1: Mount NXT Light Sensor on Domabot front. Sensor should be about 2-mm off the floor

Step 2: Type and Compile the following NXT program (**Figure 1-2**) **lfCal1_0.nxc**

```
// FILE: lfCal1_0.nxc - Works!
// AUTH: P.Oh
// DATE: 09/30/22 09:55
// DESC: Domabot Lego NXT Light Sensor calibration for line following
//       Motor Right (Port A), Motor Left (Port C), Light Sensor (Port 3)
//       Domabot yaws back-and-forth to define min and max IR values
//       and display them and calculated threshold value
// REFS: lf0_1d1.nxc
// VERS: 1.0: ME 425 release
```

```
task main() {

  // Variable declarations ----------------------------------------------------
  bool orangeButtonPushed, rightButtonPushed, leftButtonPushed; // NXT buttons
  int irMin, irMax, irValue, irThresh; // [0, 100] light sensor values
  int speed, speedSlow, speedBase; // [0, 100] motor speed
  unsigned long endTime; // [ms] for calibration stopwatch

  // Variable initializations -------------------------------------------------
  irMin = 999; // Max light sensor value. Set to high value it'll never reach
  irMax = -1; // Minlight sensor value. Set to low value it'll never reach
  speedBase = 50; // Domabot's default speed.  Tune for faster/slower response
  speedSlow = 10; // Used to sweep from white to black and black to white line

  // Algorithm begins ---------------------------------------------------------
  TextOut(0, LCD_LINE1, "Orange Btn starts" );
  do{ // nothing until user pushes Orange button
    orangeButtonPushed = ButtonPressed(BTNCENTER, FALSE);
  } while(!orangeButtonPushed);

  SetSensorLight(IN_3);  // initialize IR sensor for 200 ms
  Wait(200);

  // Place light sensor on outside of black track as depicted below
  /*            Forward direction
                       ^
                       |
                  ///////
                  /////// Place Sensor Here
                  ///////
                  ///////  Outermost part
                  ///////  of black line
                  ///////
  */

  // Assumes IR sensor is on outer border of oval
  // Step 1: Turn left (CCW) calibration - will rotate IR sensor from white to black

  endTime = CurrentTick() + 2000; // 2000 msec stopwatch
  OnFwd(OUT_A, speedSlow); // Right motor forward, Left motor backward
  OnRev(OUT_C, speedSlow); // results in yawing slowly CCW from white to black
  while (CurrentTick() < endTime) {
     irValue = Sensor(IN_3);
     if (irValue > irMax) {
        irMax = irValue;
     } else if (irValue < irMin) {
        irMin = irValue;
     }  // end if-else
   }  // end while - and we now have max and min light sensor values
```

```
  // By now, the light sensor has yawed onto or past the black line
  /*            Forward direction
                       ^
                       |
                    ///////
   Sensor now here  ///////
                    ///////
      Inner part    ///////
    of black line   ///////
                    ///////
  */

  // Step 2: So yaw CW to rotate IR sensor from black to white
  endTime = CurrentTick() + 3000; // 3000 msec stopwatch
  OnFwd(OUT_C, speedSlow); // Left motor forward, Right motor backward
  OnRev(OUT_A, speedSlow); // results in yawing slowly CW from white to black
  while (CurrentTick() < endTime) {
     irValue = Sensor(IN_3);
     if (irValue > irMax) { // robot rotates CCW for 3 s gathering IR values
        irMax = irValue;
     } else if (irValue < irMin) {
        irMin = irValue;
     } // end if-else
  } // end while

  // Step 3: Calculate and display threshold value until user hits Right Button
  Off(OUT_AC);  // briefly stop
  PlaySound(SOUND_UP);
  irThresh = (irMin + irMax)/2; // average of min and max IR values
  ClearScreen();
  TextOut(0, LCD_LINE1, "Calibration values" );
  TextOut(0, LCD_LINE2, FormatNum("irMax = %d" , irMax) );
  TextOut(0, LCD_LINE3, FormatNum("irMin = %d" , irMin) );
  TextOut(0, LCD_LINE4, FormatNum("irThresh = %d" , irThresh) );
  TextOut(0, LCD_LINE6, "-> BTN to proceed" );
  do{
    rightButtonPushed = ButtonPressed(BTNRIGHT, FALSE);
  } while(!rightButtonPushed);
  ClearScreen();

  // User pushed Left Button, so exit gracefully
  Off(OUT_AC);
  PlaySound(SOUND_DOUBLE_BEEP);
  Wait(5000);
  StopAllTasks();
} // end main
```

Figure 1-2: Lego NXT Light Sensor calibration code for calibration code for **`lfCal1_0.nxc`**

Step 3: Place Domabot on black line, making sure it's light sensor is on the outermost edge of the black line. The execute **`lfCal1_0.nxc`** and record the displayed values

Code Explanation: Most of `lfCal1_0.nxc` is straight-forward. It begins with declaring and then initiating variables e.g. `irMin`, `irMax`, and `irThresh` – which we'll be using again in the future (e.g. bang-bang control `lfb1_0.nxc` and proportional control `lfp1_0.nxc`)

The first yellow-highlight introduces the NXC function `CurrentTick()`. Microcontrollers like the NXT Brick, have a built-in crystal. The NXT Brick's crystal produces a tick every millisecond and the value is stored as a 32-bit number. Thus the timer can store up to 2^{32}milliseconds. This equates to 49 days, 17 hours, 2 minutes, 47 seconds and 296 milliseconds – in other words, a long time. Here, `endTime = CurrentTime() + 2000` defines a 2 second timer. One sees that a `while` loop polls the crystal's value. If that value is under 2000 milliseconds, it repeats. If the value is greater than 2000 milliseconds, the `while` loop exits. In-between the while loop, the Lego NXT light sensor is recording values as the Domabot yaws clockwise (CW).

The second yellow-highlight is another timer but for 3000 milliseconds. Like the previous `while` loop, the light sensor values are measured as the Domabot yaws counter-clockwise (CCW).

The third yellow-highlight simply calculates the average of the minimum and maximum sensor readings and stores it in `irThresh`.

Concept 2: Domabot bang-bang control line following

The lecture notes detail the underlying theory, approach and pseudo-code.

Step 1: Write a bang-bang control NXC program called **`lfbb1_0.nxc`** – that builds on Concept 1's `lfCal1_0.nxc`. Translate the pseudo-code into NXC code to demonstrate bang-bang line following.

Concept 3: Domabot proportional control line following

The lecture notes detail the underlying theory, approach and pseudo-code.

Step 1: Write a proportional control NXC program called **`lfp1_0.nxc`** – that builds on Concept 1's `lfCal1_0.nxc` and `lfbb1_0.nxc`. Translate the pseudo-code into NXC code to demonstrate proportional control line following.