ME 3200 Mechatronics I Laboratory

Lab 6: Introduction to Photosensors

Introduction

Autonomous robots are often programmed to respond to specific kinds of external sources—such as light, sound, temperature, elevation, etc. In order to attract these robots to these sources, designers use specialized sensors that are sensitive to the desired source. In this laboratory exercise, you will learn how to use photosensors in conjunction with the Handy Board to detect the presence of infrared light. In addition, you will explore the characteristics of the photosensors that will be used for your project and how you might use these sensors to drive your robot to a source of infrared light. The techniques used in this lab to explore a sensor's capabilities are also applicable to other types of sensors—such as gas, sound, and temperature detection.

Background

Photosensors come in a variety of types, but the one thing that they all have in common is that they are used to detect the presence of some wavelength of light. Some are used to detect visible light; some detect ultraviolet light; the photosensors we use in lab, however, detect infrared (IR) light. The photosensors used in this lab are infrared detecting phototransistors as depicted in Figures 1 and 2 below.

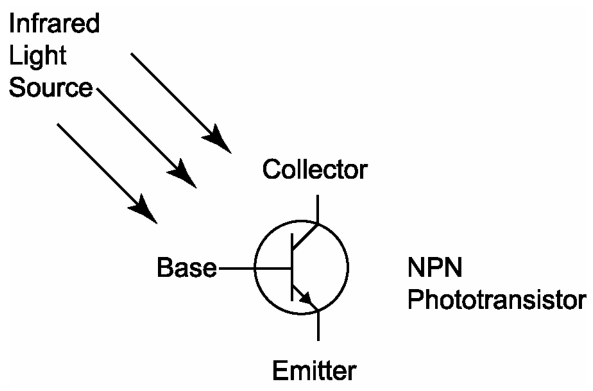

Figure 1: Schematic of an infrared detecting phototransistor.

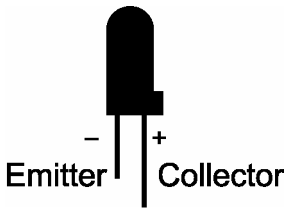

Figure 2: An infrared detecting phototransistor.

Figure 1 shows that, as the base of the phototransistor is exposed to infrared light, it "closes" a switch between the collector and emitter. If a voltage difference is applied across the IR sensor, current is allowed to pass from the collector to the emitter. The amount of current allowed to pass through the sensor is proportional to the amount of infrared light received by the base: more light produces a higher current. Because of this phenomenon, the phototransistor can be viewed as a variable resistor. When there is no infrared light present, the resistance is relatively high. When the IR sensor is saturated with infrared light, the resistance is much lower. This type of sensor is ideal for use with the Handy Board and its analog input ports.

IR Sensors and the Handy Board

Before using these IR sensors with the Handy Board, the user must understand how to properly connect the IR sensor to the Handy Board and how they behave when connected. Start by examining the IR sensor and determining which of its two leads are respectively the emitter and the collector. Figure 2 depicts the difference between the two; remember that the collector lead is longer than the emitter, and that the emitter side of the sensor has a flat face. Once these leads are identified, the IR sensor is connected to one of the Handy Board analog input ports according to Figure 3 below.

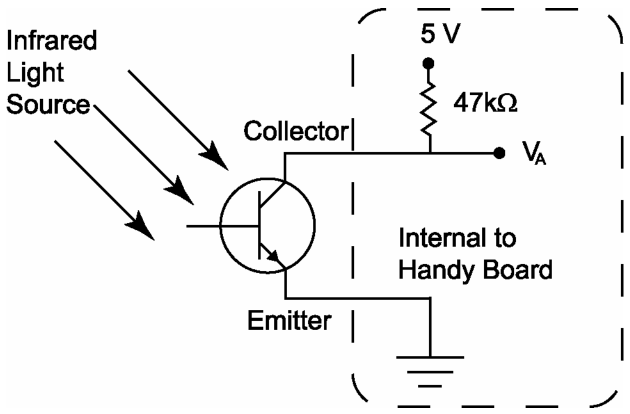

Figure 3: The IR sensor and Handy Board combining to form a voltage divider.

Once the sensor is in place, the internal voltage supply and pull-up resistor of the Handy Board combine with the sensor to form a voltage divider that determines the voltage, V_A, read by the Handy Board's eight-bit ADC (see Figure 3 above). To determine the infrared light level, use the `analog(p)`command—where `p` is an integer indicating which analog port to sample (refer to the number printed on the circuit board below the ports). The `analog(p)` command returns an integer will be returned that corresponds to the voltage, V_A, which is proportional to the effective resistance of the light sensor. In other words, in the absence of infrared light, the effective resistance of the sensor will be relatively high returning an integer near 255; when the sensor is saturated with infrared light, the integer returned will be very near zero. Recall that the analog-to-digital converter in the Handy Board will always return an integer value between zero and 255. It may not be very intuitive, but the integer returned from the `analog(p)`command is inversely proportional to the infrared level present. To make the IR readings more logical, you may prefer to always treat "`255-analog(p)`" as the reading. Then a reading of 255 corresponds to sensor saturation and 0 corresponds to no light.

Whenever IR sensors are used, it is important to consider which sources of infrared light are present in the operating environment that may influence the response of the sensor. The best infrared sources are light sources that generate a lot of heat; incandescent light bulbs and the sun are the best examples. Florescent lights, however, produce very little infrared light. Whatever infrared light sources are present, it is always good to determine how they will affect the behavior of the IR sensors.

Pre-lab Exercise

Examine the Interactive C code on the following pages. This code uses background processes to implement a light searching algorithm for a robot you will use in the lab.

On each line where you see the comment characters "// ****" write a description of the purpose or function of the code at that location (lines 48, 50, 53, 62, 72, 103, 106, 107, 113, and 118). Be prepared to show your interpretation of these sections to your TA at the beginning of lab.

```c
// Amount of time to let the robot run.
#define RUNTIME 30.0
// Direction control flags.
#define LEFT 0
#define RIGHT 1
// Motor power level definitions.
#define HIGH 80
#define MID 60
#define LOW 40

int irRead[5] = {0,0,0,0,0}; // Array used to store the IR sensor readings.
int hit = -1; // Variable used to indicate where a light source has been detected
int oldHit = hit;
int timeup = 0; // Flag used to signal when time has expired.

// Variables used to store the process ID's of the three main processes.
int timerPID;
int lightLocatorPID;
int controlMotorsPID;

// The timer function waits for a specified length of time and then switches the
// timeup flag.
void timer( float t ){
    float startTime = seconds();
    while( seconds() - startTime < t ){  // While time has not expired.
        defer();  // Relinquish control of the processor.
    }
    timeup = 1; // Once time has expired, change the flag, and exit the function.
} // timer( float t )

// The lightLocator function sets the variable hit to an integer representing
// the IR sensor that is detecting the most IR light.
// -1 indicates that no meaningful hit was detected.
// 0 through 4 indicate which sensor has a hit.
void lightLocator() {
    int i;        // A counter index.
    int minRead;  // The lowest IR reading.
    int maxRead;  // The highest IR reading.
    int maxIndex; // The index (sensor number) of the highest IR reading.

    while(!timeup){
        // Reset tracking variables.
        minRead = 256;
        maxRead = -1;
        maxIndex = -1;
        // Loop through each sensor to obtain the readings.
        for(i=0; i<=4; i++){
            irRead[i] = 255 - analog(i); // ****

            if( irRead[i]<minRead ){     // ****
                minRead = irRead[i];
            }
            if( irRead[i]>maxRead ){     // ****
                maxIndex = i;
                maxRead = irRead[i];
            }
        }
        // Process the readings to see if a meaningful hit was found.
        if( /* Enter a Boolean statement that recognizes a true hit. */ ){
            hit = maxIndex;
        }
        else{ hit = -1; } // ****
    }
} // lightLocator()


```

```
// The controlMotors function controls the motors based on the value of "hit"
// which is continuously updated by lightLocator().
void controlMotors()
{
    while(!timeup){ // As long as time is left, continue updating the motor commands.
        if ( hit != oldHit){ // ****
            oldHit = hit;
            if(hit == -1){ // No hit; what should you do?
                defer();
            }
            if(hit == 0){  // Turn left quickly
                defer();
            }
            if(hit == 1){  // Turn left slowly
                defer();
            }
            if(hit == 2){  // Go forward
                defer();
            }
            if(hit == 3){  // Turn right slowly
                defer();
            }
            if(hit == 4){  // Turn right quickly
                defer();
            }
        }
    }
    // Once time has expired kill the engines and end the function.
    alloff();
} // controlMotors()

// Main simply starts processes for the controlling functions above.
void main()
{
    while(1){
        printf("Press start to  chase light.\n");
        while( !start_button() ){  } // ****

        printf("Chasing!!\n");
        reset_system_time();  // ****
        timeup = 0;           // ****

        timerPID = start_process( timer( RUNTIME ) );
        lightLocatorPID = start_process( lightLocator() );
        controlMotorsPID = start_process( controlMotors() );

        while(!timeup && !stop_button() ){  // ****
            // Add debugging commands here as needed.
            defer();
        }

        // ****
        kill_process(controlMotorsPID);
        kill_process(lightLocatorPID);
        kill_process(timerPID);
        alloff();
        beep();
    }
} // main()

```

Laboratory Exercise

Equipment needed:

- A Handy Board
- An RJ11 cord
- A 12 Volt, 500mA DC transformer
- A flashlight
- A CutieBot equipped with infrared photsensors.
- A paper compass (see the link on the Lab Handouts page of the class website.)

1) Interface your Handy Board with your computer using the same procedure from the *Introduction to the Handy Board* lab, place the Handy Board in interactive mode, and start Interactive C.

2) Insert the collector of an IR sensor into an analog input port, and insert the emitter into the ground bus. Refer to Figure 2 above to help determine which lead is the anode and which is the cathode.

3) Use the `analog(p)`command to determine the ambient infrared light level in the lab. Record the reading in the space below:

4) Using the flashlight, saturate the IR sensor with light and run the `analog(p)`command again. Record the sensor reading below:

5) Write a program using the sample code below that averages five light readings of the center IR sensor of the CutieBot, normalizes the average (from 0 to 1), and prints the result to the LCD screen. Load the program into the Handy Board and verify that it works appropriately.

   Note: The CutieBot uses five IR sensors distributed in increments of 22.5° along the front of the CutieBot. The IR sensors are connected to analog ports 0 (leftmost) through 4 (rightmost) of the Handy Board of the CutieBot.

```
int i;                                  /* i is a counter index */
int avg=0,avg2=0;                       /* average center ir value*/
float center;                           /* value of the center ir */
int num=5;                              /* the number of readings taken */

void main() {
  printf("Press start to read center ir");
  while(!start_button()) {}             /* wait until start is pressed */
  while(1) {                            /* run in a continous loop */
    if(start_button()) {                /* take readings if start is pushed */
      for(i=1; i<=num; i++) {           /* loop for num times */
        avg += 255 - analog(2);         /* sums the value of the num runs */
      }
      avg /= i;                         /* divide by i to take the average */
      avg2 = (int)((float)avg/.255);    /* rounds answer to 3 sig. figs. */
      center = (float)avg2/1000.;       /* a float with range of 0 - 1 */
      printf("C=%f\n",center);          /* print the results */
      avg = 0;                          /* reset the value of average */
    }
  }
}
```

6) Using a CutieBot, the program you wrote in step 5), a flashlight, and the paper compass provided by the TA determine the IR detector's angular response characteristics.

   a) Place the CutieBot on the floor on top of one of the paper compasses with the center IR directly over the 90° point and so that the wheel axles are parallel to the 0° - 180° line.

   b) Set the flashlight on the floor at least five feet from the CutieBot with the beam shining down one of the lines in the floor.

   c) Using the paper compass, a seam between the tiles of the floor as a guide, and the program you wrote in Step 5), record the IR level at each mark on the compass. Record these data in the table below (these data will not necessarily be symmetric).

Angle [degrees]	IR Level
0	
10	
20	
30	
40	
50	
60	
65	
70	
75	
80	
85	
90	

Angle [degrees]	IR Level
95	
100	
105	
110	
115	
120	
130	
140	
150	
160	
170	
180	

7) Create a polar plot of the data collected in step 6)c) above; Excel does not support a polar plot format so you will need to create the plot in Matlab.

   a) To create this plot, use Excel to make a file where column 1 lists the angle and column 2 lists the normalized IR readings; save the file in .dat format.

   b) Open Matlab and set the current directory to the folder in which you saved your data file in the previous step.

   c) Enter the following commands into the command window.

```
load yourfilename.dat
polar(yourfilename(:,1)*pi/180, yourfilename(:,2))
```

   The plot will come up in a new window labeled Figure 1. Save the plot to your disk and compare your plot with Figure 2 of the IR sensor datasheet (a link to the datasheet is included on the Lab Handouts page of the class website).

8) Place the flashlight directly in front of the CutieBot at varying distances (between six inches and eight feet) and record the IR sensor readings from your program at each distance in the table below. Comment on the reliability of the sensor to determine the distance to a light source.

Distance											
IR Reading											

Note: In the remaining steps of this procedure you will use the Interactive C code available on the class website under Lab Handouts entitled "irFinder.c" to create a program that makes a CutieBot chase an infrared light source. The "irFinder.c" program utilizes background processes to create a robust and capable program that (a) imposes a time limit on how long the CutieBot will chase a light source, (b) continuously monitors the state of the IR sensors, and (c) continuously updates motor commands that direct the CutieBot toward the light. Remember that the IR sensors are numbered from 0 (leftmost) through 4 (rightmost).

9) Open `irFinder.c` in the text editor of Interactive C and alter the conditional argument of the `if()` statement on line 60 of the `irFinder` code to include a Boolean statement that will recognize whether a true hit has been found. Consider the following examples of infrared readings to help formulate your Boolean statement; think about which of these sets really has a "hit", and how to determine what a true hit is:

Ex.	**255 – Analog Input Channel**					**Result**	
	0	**1**	**2**	**3**	**4**	**Hit?**	**Why?**
A	5	6	5	7	5		
B	13	150	50	18	12		
C	150	253	180	50	40		
D	250	255	253	250	254		

10) Add a `printf()` statement in the `while(!timeup)` loop of the `main()` function that continuously prints the value of hit while the program is "Chasing".

11) Save the modified code to a floppy disk and download it to your CutieBot's Handy Board.

12) Reset the Handy Board (turn off then on again) and then use a flashlight to verify that the value returned for hit is appropriate depending on where the light is relative to the sensors.

13) When you are confident the `lightLocator()` function is working appropriately, alter the `controlMotors()` function to include motor statements that will turn the CutieBot appropriately depending on the value of hit.

    Consider creating additional functions that will provide appropriate motor commands; appropriate function headers could include:

    ```
    turn(int direction, int power);
    goForward(int power);
    ```

14) Continue to modify your code until the CutieBot will accurately follow the light of a flashlight and stop after time has expired. Remember that removing the printf() statements will improve the program performance.

15) Demonstrate your CutieBot's behavior to your TA, clean-up your lab station, and answer the questions at the end of the lab.

Questions

1. Based on your experiments, how much variability in light sensor readings would you expect for repeated measurements of a constant light source? Are there any advantages/disadvantages of averaging multiple readings from the same IR sensor?

2. Based on the CutieBot IR sensor configuration, are the IR sensors good sensors to determine the distance from a light source in the environment? Why or why not?

3. Compare the angular resolution of the IR sensor as stated in the datasheet with your experimental results. How are they similar/different? Why?

4. Why does the Handy Board return a small number with high IR light levels and a large number with low IR light levels?

5. What is(are) the benefit(s) of using background processes when you program mobile robots with the Handy Board.