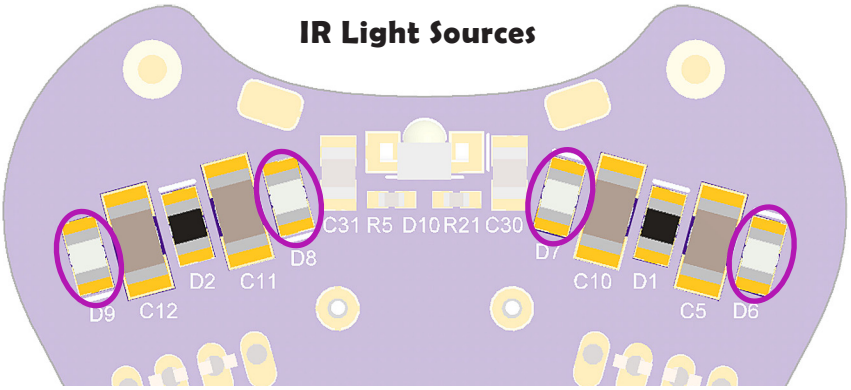
Bottom Sensor Basics

Lesson 14

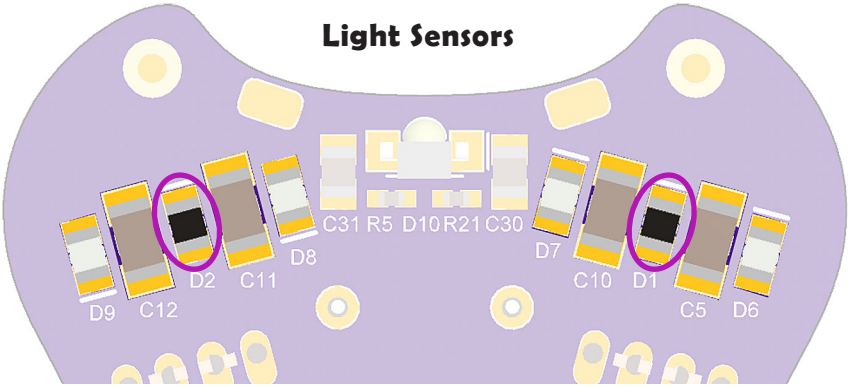
Lets start learning how Wink's bottom sensors work. He can use these sensors to see lines and measure when the surface he is driving on has changed.

Bottom Sensor Basics...

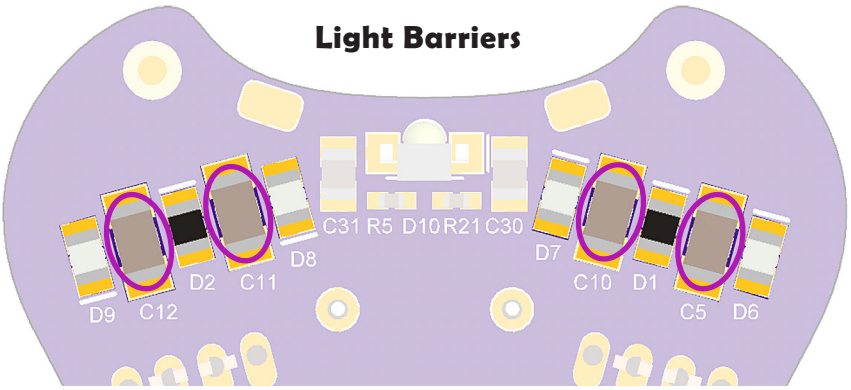
IR Light Sources



Light Sensors



Light Barriers



3
SKILL LEVEL

Bottom Sensor Basics

Lesson 14

Turn Wink over on his back side and see if you can identify the parts of the bottom sensors using the three pictures above. The bottom sensors are made up of three different parts.

IR Light Sources

The four IR Light Sources can be turned on in any combination with your code. These light sources shine an invisible infrared light downward onto the surface Wink is driving over. They are arranged with a pair on each side of the robot. Also notice there is an “inner pair” more toward the inside of the robot, and an “outer pair” more toward the outside. They can be used in different combinations for different kinds of sensing.

Light Sensors

There are two light sensors, one on each side. These are located between each pair of IR Light Sources. The Light Sensors measure the amount of light reflected from the surface. If you turn on an IR Light Source, you can measure how much light reflects. If Wink is over a white surface, a larger amount of light will reflect, and less light will reflect if he is over a darker surface.

Light Barriers

The Light Barriers are used to prevent the IR Light Sources from shining directly into the Light Sensors. Because some light escapes the sides of the IR Light Sources, if we didn’t have the Light Barriers present, the Light Sensors would pick up some of this sideways light. This would reduce their sensitivity to light reflected from the surface. With the Light Barriers in place, almost all of the light measured by the Light Sensors is light that actually reflected from the running surface.

Bottom Sensor Operation

The bottom sensors work the same way as the IR Headlight in our barrier detection example from a previous lesson. In order to measure how bright or dark a surface is, we can turn off all the IR Light Sources, then measure both Light Sensors. We can then turn on both the inner IR Light Sources and then measure the Light Sensors, then off the inner IR Light Sources and turn on the outer IR Light Sources and again measure both Light Sensors. If we subtract out the amount of light from the first set of readings (when all IR Light Sources were off), we are left with four values which indicate the relative brightness or darkness of the surface under each of the IR Light Sources.

By looking at these four values, we can determine if any of Wink’s sensors are over a dark area, like a line.

Bottom Sensor Code...

Lets write up some code to carry out the process we described above.

```
int leftOuter, leftInner, rightInner, rightOuter;
int leftLineSensorValueOff, rightLineSensorValueOff;

void loop(){

  Serial.print(leftOuter);Serial.print("\t"); Serial.print(leftInner);Serial.print("\t");
  Serial.print(rightInner);Serial.print("\t");Serial.println(rightOuter);
  delay(200);

  //make sure all bottom IR light sources are off before we begin
  digitalWrite(LineLeftOuter,LOW);
  digitalWrite(LineRightOuter,LOW);
  digitalWrite(LineLeftInner,LOW);
  digitalWrite(LineRightInner,LOW);
  delayMicroseconds(500);   //short delay to allow sensors to stabilize

  //measure sensors with IR light sources turned off
  leftLineSensorValueOff=analogRead(LineSenseLeft);
  rightLineSensorValueOff=analogRead(LineSenseRight);

  //turn on outer light sources, then re-read the sensors
  digitalWrite(LineLeftOuter,HIGH);          //turn on outer IR light sources
  digitalWrite(LineRightOuter,HIGH);
  delayMicroseconds(500);   //short delay to allow sensors to stabilize

  //measure with outers on, subtract the dark reading from above
  leftOuter = analogRead(LineSenseLeft)-leftLineSensorValueOff;
  rightOuter = analogRead(LineSenseRight)-rightLineSensorValueOff;

  //off outer sources, on inner sources, then re-read the sensors
  digitalWrite(LineLeftOuter,LOW);           //turn off outer IR light sources
  digitalWrite(LineRightOuter,LOW);
  digitalWrite(LineLeftInner,HIGH);          //turn on inner IR light sources
  digitalWrite(LineRightInner,HIGH);
  delayMicroseconds(500);   //short delay to allow sensors to stabilize

  //measure with inners on, subtract the dark reading from above
  leftInner = analogRead(LineSenseLeft)-leftLineSensorValueOff;
  rightInner = analogRead(LineSenseRight)-rightLineSensorValueOff;

  //turn inner light sources back off
  digitalWrite(LineLeftInner,LOW);           //turn off inner IR light sources
  digitalWrite(LineRightInner,LOW);
}
```

Wink_Ch14BottomSenseBasics_Ex01

Woah. That's a long block of code! You should be getting comfortable reading comments in code by now, so I'll encourage you to read over the block of code above and study the comments. You should be able to see what is happening.

Can you imagine writing that entire block of code each time you want to measure the bottom sensors? Luckily we learned how to use functions in the previous lesson. We'll turn this code into a function in the next example so we can easily re-use it.

Load the previous example into Wink and open your serial monitor window. You should see a set of four values being printed, which correspond to the readings of each of the four bottom sensors. Get a piece of white paper and draw a bold black line on it. You can also draw a black square and color it in. Now experiment sliding Wink over the white and black areas while observing how the values change. When a sensor is over white paper you should see values between 700 and 1000, and when a sensor is over a black area, you should see values less than 100. You will also notice these values change quickly when moving between light and dark areas.

Study the example below. We've shortened it on this page so it will fit. Open up this example from the companion code for this lesson and study it. We've just taken the large block of code from above and made a function called readLines() from it. This next example does exactly the same thing as the first example, except you can now simply call readLines() to update the variables leftOuter, leftInner, rightInner, and rightOuter.

```
//global variables for line sensor results
int leftOuter, leftInner, rightInner, rightOuter;

void loop(){

  readLines();    //read bottom sensors, update variables

  //print the readings to the serial monitor window
  Serial.print(leftOuter);Serial.print("\t");
  Serial.print(leftInner);Serial.print("\t");
  Serial.print(rightInner);Serial.print("\t");
  Serial.println(rightOuter);
  delay(200);

}

void readLines(void){
  //this section omitted in the written lesson. Open and
  //view the sketch Wink_Ch14BottomSenseBasics_Ex02 from
  //the companion code to study the actual function.
}
```

} We now simply call the readLines() function instead of writing the long block of code in the first example.

} Code to read lines now moved to its own function for easy re-use.

Wink_Ch14BottomSenseBasics_Ex02

3 SKILL LEVEL

Trapped in a circle...

Now lets try a simple example that uses the bottom sensors to do something interesting. Lets take a piece of white paper and draw a large circle on it using a thick black marker. We will let Wink drive around inside this circle, and every time he senses the line, he will stop, reverse, turn, and drive again. This should result in him being "trapped" inside the circle.

This should be an easy one to write on your own. You know the bottom sensors will read a high value when over the white paper, and they will quickly drop to a low value when you cross the line. All we need to do is read the sensors over and over, and use an if/else control structure to make Wink stop, back up, and turn when he sees a line.

We'll evaluate the left and right sides separately and make the turn correspond to moving away from the side that went to the lower value - this way he will always tend to steer away from the line.

The important parts are listed below, but open the companion code file to see the entire sketch to really study how it works.

Note: If your Wink doesn't "see" the line, try making the line thicker and darker. You can also lower the motors speed a bit. Try something lower like 40.

```
int lineThresh = 500;      //darkness for a line to be "seen"
void loop(){

  motors(70,70);           //go forward
  readLines();             //read line sensors

  if (leftOuter < lineThresh){
    eyesRed(50);
    motors(-80,-80);       //go backward
    delay(200);            //... for 200 ms
    motors(80,-80);        //right turning motion
    delay(100);            //... for 100 ms
    delay(50);             //short delay
  }

  else if (rightOuter < lineThresh){
    eyesRed(50);
    motors(-80,-80);       //go backward
    delay(200);            //... for 200 ms
    motors(-80,80);        //left turning motion
    delay(100);            //... for 100 ms
    delay(50);             //short delay
  }
}
```

Wink_Ch14BottomSenseBasics_Ex03

} If a sensor reads lower than this (a darker surface) then Wink will "see" the line and react to it.

} Drive forward while continually reading line sensors.

} Do this reaction if the leftOuter sensor reads below the lineThresh limit set above.

} Do this reaction if the rightOuter sensor reads below the lineThresh limit set above.

3 SKILL LEVEL

Making the example more dynamic...

This example works well, but we can make a few minor improvements to it. The example above assumes a white piece of paper with a nice dark line drawn. It assumes the paper will always be brighter than 500 and the line will always be darker than 500. But what if we drive Wink on a grey piece of paper with a line that is a lighter color? He may not necessarily react the same way. We can make the behavior more “dynamic” (which means it can adjust to more situations) with a few changes.

Let’s discuss our options first, then we can make the required changes to our code.

No matter what the color of the paper or the line, we know the readings of the line sensors will be lower when a line is seen (that is assuming the paper is a brighter color than the line). Instead of making 500 a hard limit for this, let’s instead say that if the line sensor values drop by a certain amount, that we consider this “seeing” a line.

In the previous example, if the blank paper with no line has a brightness of 800, and a when a line is crossed the reading drops to 580, the line won’t be “seen”. But if instead, we say that a line is “seen” if the blank paper reading drops by more than 200, then the line would be seen.

To make this work, we’ll need to take at least one initial reading in our code while Wink is not over a line, and save this value in memory. This reading will represent the brightness of the paper. We’ll call this initial value “baseline”, as this will be the “baseline” brightness level of the paper. We’ll read all four bottom sensors, add them all up, then divide by four, which will average them. We’ll save that result in a variable called “baseline”.

We also need to decide how far the value should drop before we assume a line has been “seen”. We’ll declare this as another variable called “dropThresh” and give it a value at the top of our code. This way we can easily adjust it later. Keep in mind that making this value very low may result in Wink reacting to a line that doesn’t exist, and making this value very large may make him not react to the line at all.

One other quick note. You may have noticed while viewing the serial data above, that the values can change by as much as 200 to 300 counts if you rock Wink forward or backward against his front and rear felt running pads. This is because the sensors will read a lower value as they move further from the surface (when Wink rocks backward). This will be important as we first react to a line, then after backing and turning, we start driving forward again. If we quickly re-read the bottom sensors immediately after we start moving, Wink’s nose may still be elevated from the surface because he has rocked backward during the turning movement. This may cause him to immediately false trigger a line response again. To get around this, after we do our reverse and turn move, we’ll start driving forward for a brief time (about 50 milliseconds should be enough) to allow his nose to re-settle to the normal height before reading our sensors again.

Attempt this on your own first. It shouldn’t be too hard. Then open the companion code sketch named Wink_Ch14BottomSenseBasics_Ex04. Some important parts of this new sketch are on the next page, but some parts are omitted to save page space. The companion code shows the complete working sketch.

```
int baseline;              //holder for our “baseline” reading
int dropThresh = 200;      //drop threshold to “see” a line
int motorSpeed = 70;       //motor speed

//other global variables for line sensor results
int leftOuter, leftInner, rightInner, rightOuter;

void setup(){
  hardwareBegin();
  playStartChirp();
  delay(2000);             //wait 2 seconds to put Wink down
  readLines();             //read line sensors
  baseline = (leftOuter+leftInner+rightInner+rightOuter)/4; //get baseline
}

void loop(){
  eyesYellow(50);                  //turn on eyes
  motors(motorSpeed,motorSpeed);   //go forward
  readLines();                     //read lines, update global variables

  if (leftOuter < (baseline-dropThresh)){
    eyesRed(50);
    motors(-80,-80);       //go backward
    delay(200);            //... for 200 ms
    motors(80,-80);        //right turning motion
    delay(100);            //... for 100 ms
    motors(motorSpeed,motorSpeed);  //causes Wink to rock back forward
    delay(50);
  }

  else if (rightOuter < (baseline-dropThresh)){
    eyesRed(50);
    motors(-80,-80);       //go backward
    delay(200);            //... for 200 ms
    motors(-80,80);        //left turning motion
    delay(100);            //... for 100 ms
    motors(motorSpeed,motorSpeed);  //causes Wink to rock back forward
    delay(50);
  }

}
```

Wink_Ch14BottomSenseBasics_Ex04

} Declare variables

} Wait 2 seconds so you can put Wink down on blank paper. Then read lines, then calculate “baseline” from average of all 4 sensor readings.

} Drive forward while continually reading the line sensors.

} Drive forward for 50 ms before leaving the “if” to read lines again.

} Drive forward for 50 ms before leaving the “if” to read lines again.

3 SKILL LEVEL

Challenge...

So that's the basic idea of how the bottom sensors work. In a future lesson we're going to discuss the idea of actually following a line, but you know enough at this point that you can probably figure this out on your own.

There are a few basic ways you can make line following work. Here are a few tips if you'd like to experiment on your own.

You can use a simple threshold method as we have in this lesson. If a sensor drops below a certain value, you adjust motor speeds to steer away from the line briefly then begin moving forward again. You'll probably want to use the inner set of sensors for this. This will cause Wink to wag side to side as he bumps into the line from side to side. You will need to keep up your sampling rate (the how fast you call readLines) and make sure your speed adjustments to the motors aren't too extreme or Wink will over shoot and drive right off the line. This will be easier to control if Wink is driving a slower speed.

Another method, slightly more complex but results in smoother movement, is to directly translate the drop in value of a sensor to a speed change of the motors. For example, read the sensors, and if the value drops by 50 on the right side (because Wink has steered too far to the left, causing the right side sensor to cross the line), you can simply subtract 50 from the motor speed of the right side, which will cause him to steer back toward the right.

For this to work you can get a baseline reading as we have already done, then run a loop where you read the sensors, and whatever value the sensors read below the baseline, you subtract this from the motor speed on that side. There is no if/else needed in this case, as the readings are constantly translated directly into a change in motor speed.

If Wink is reacting too strongly to this change in speed, you may want to add a multiplier that reduces this difference by a certain amount before being applied to motor speed. For example, maybe you take the drop in brightness and multiply it by 0.3, then use the result of that calculation to subtract from the motor speed. This should result in Wink being 70% less responsive to the line.

We'll go into this in a full lesson in the future, but if you're up for a challenge give it a go on your own. Good luck!