Advanced Light Seeking

Lesson 12

Remember back to Lesson 8 where we learned how to read Wink's ambient sensors, and also Lesson 10 where we learned how to detect a barrier in front of Wink. In this lesson we will build on both of those past lessons and learn a few new things.

In this lesson we will put together several earlier concepts to build a complete light seeking behavior that works like the one in the Wink Demo behaviors. By the end you'll be able to do all kinds of fun things with light.

Wink's color eyes and Light Sensors...

In earlier lessons we have used Wink's ambient light sensors to see the light around him as well as barriers, but in all of the previous examples, his eyes were turned off. This is because Wink's ambient light sensors can see the light produced by Wink's eyes.

If we turn on Wink's eyes while reading the ambient light sensors, we will get a different reading compared to reading when the eyes are turned off. Wink's eyes change their brightness by blinking on and off very quickly. When Wink's eyes are dim, they are on for a very short time, then turned off for a while, then turned back on for a very short time. This on and off action is fast enough that your human eyes can't see it. But Wink's ambient sensors can see this blinking effect.

That is to say, if you turn on Wink's eyes using eyesBlue(100), then start reading the ambient sensors in your code, you will start getting values that are not constant. The values will bounce around depending on whether the sensor is read while Wink's eyes are momentarily turned on or if the reading happened while they were off. There is a mini computer inside each eye that handles this blink rate, so it is impossible to "synchronize" the reading of the light sensors with the blink rate of the eyes.

For the above reasons, you may think it is impossible to do anything useful with Wink's ambient light sensors while making his eyes light up. This is actually true, but there is a work around.

Simply turn the eyes off, then read the sensor, then turn the eyes back on.

```
void loop(){
 eyesOff();
 sensorValue = analogRead(AmbientSenseCenter);
 eyesOn();
}
```

} Briefly turn off Wink's eyes before reading an ambient sensor.

The sensor doesn't take very long to read, so your human eyes will hardly see the time the eyes are off. You may perceive a brief change of brightness or flicker, but it will appear as if the eyes are generally turned on the entire time.

There is one more thing to consider using this trick of briefly turning the eyes off before reading the ambient sensor.

We need to add an extra bit of delay between the time we turn off the eyes and the time we read the ambient light sensor. This is for two important reasons.

1. The first reason is that it takes a short time for the ambient light sensor to react to the change in light level. Because Wink's code runs very fast, if you attempt to read the ambient light sensor immediately following turning off the eyes, the sensor will be read before it has time to react to the new, lower light level.
2. The second reason is that Wink's eyes have their own internal computer. When they are commanded to change to a new brightness level (or turn off the eyes) it takes them a brief time to actually change the brightness of the eyes. This usually happens almost instantly but sometimes the eyes can take up to 4 milliseconds to actually turn off.

Normally the eyes will turn off and the sensors will react to the new lower light level after about 250 microseconds (which is 1/4 of a millisecond). Though sometimes it can take up to 4 milliseconds for the eyes to turn off due to the second reason above. This has to do with the internal clocking of the eyes themselves and is beyond this lesson. You may choose to use the shorter delay of 250 microseconds or the much longer delay of 4 milliseconds depending on what you are doing.

If you are running your code in a fast loop where the light sensor is constantly measured and your code responds to this in some way (like changing the speed of the motors) then an occasional reading that is incorrect won't cause much harm as the sensor will be quickly read again shortly following. But if you need to have more confidence that the reading is correct, and that it has not been interfered with by the eyes, use the longer delay of 4 milliseconds. This longer delay will be more visible to your human eyes but you can be much more confident that the value read by the ambient light sensor is correct and has not been effected by the eyes being on.

```
void loop(){
 eyesOff();
 delayMicroseconds(250);
 sensorValue = analogRead(AmbientSenseCenter);
 eyesOn();
}
```

} Use delayMicroseconds(250) before reading ambient light sensors

```
void loop(){
 eyesOff();
 delay(4);
 sensorValue = analogRead(AmbientSenseCenter);
 eyesOn();
}
```

} Use delay(4) before reading ambient light sensors for a more accurate value

Advanced Light Seeking

Lesson 12

Let's revisit an example from the Light Sensors lesson. The example below does the same thing as the original example except the eyes are now turned on. Try this example and view the output in your Serial Monitor window.

```
int left,center,right;     //declare variables

void loop(){

  eyesOff();      //turn eyes off
  delay(4);       //wait 4 milliseconds before read sensors
  left = analogRead(AmbientSenseLeft);      //read left
  center = analogRead(AmbientSenseCenter);  //read center
  right = analogRead(AmbientSenseRight);    //read right
  eyesPurple(100);         //turn eyes back on

  Serial.print(left);      //left value
  Serial.print("\t");      //tab key
  Serial.print(center);    //center value
  Serial.print("\t");      //tab key
  Serial.print(right);     //right value
  Serial.println();        //print a new line character
  delay(500);              //delay 1/2 second

} //end of loop()
```

Wink_Ch12AdvLightSeek_Ex01

} Declare variables we will be using

} Turn off eyes and delay 4 milliseconds before reading the three sensors

} Read all three sensors

} Turn eyes back on immediately after reading sensors

} Print sensor values to the Serial Monitor window

The above technique can be used to operate Wink's eyes during light seeking and also barrier detection.

Building a better light seeking behavior...

Now that you know how to turn on Wink's eyes during light seeking, let's consider how we may design a light seeking behavior where Wink drives around seeking out light.

For this behavior, let's begin with Example 5 from Lesson 9 where you learned to use the "if" control structure. In that example, Wink remained in his own footprint while facing toward the brightest light. The final "else" in that example made the motors still if both eyes were seeing almost the same amount of light. Let's edit that example to make both motors move forward if the light is nearly the same on both sides. We will also lower the motor speeds a bit from 100 to 70 so Wink moves a bit more slowly while we experiment. Here's the previous example edited as a starting point for this new example. We are also adding code to turn on the eyes, and only turn them off during sensor reading.

Load up this example and use a light source to play with Wink for a while. It is important to experiment with this code on Wink before moving ahead so you can see how he acts.

```
int leftLight, rightLight;    //declare the variables

void loop(){
  eyesOff();      //make sure eyes are off
  delay(4);       //wait 4 milliseconds for sensors to settle
  leftLight = analogRead(AmbientSenseLeft);   //read left
  rightLight = analogRead(AmbientSenseRight); //read right
  eyesGreen(100); //turn eyes back on

  if (rightLight-10 > leftLight)    //if right is greater
  {
    motors(70,-70);                 //spin to the right
  }
  else if (leftLight-10 > rightLight)  //if left is greater
  {
    motors(-70,70);                 //spin to the left
  }
  else                              //otherwise...
  {
    motors(70,70);                  //drive straight
  }

  delay(100);     //wait 1/10th second before doing loop again

} //end of loop()
```

Declare the variables we will need

Eyes of and settle time

Read both sensors

Eyes back on

If rightLight is more than 10 greater than leftLight, then run code to make motors spin and rotate Wink to the right.

If leftLight is more than 10 greater than rightLight, then run code to make motors spin and rotate Wink to the left.

If neither of the above are true, then the light level must be fairly even. This is the "dead band". In this case, make the motors drive straight.

Delay before looping again

Wink_Ch12AdvLightSeek_Ex02

3 SKILL LEVEL

Advanced Light Seeking

Lesson 12

Now that you've spent some time experimenting, let's consider what is happening. If you read over the code, it seems to make sense that Wink will always seek the light, and he does. However, you've probably also noticed that when shining the light directly at him, he doesn't really track in on the light, he just wags side to side.

Let's think about why this may be happening. If one sensor reads a certain amount more than the other sensor, Wink will make a hard turn that direction. The next time the loop runs, he has over-shot the light a bit, and now the opposite is true, so he makes a hard turn back the opposite direction. This happens over and over and he never really "tracks" the light the way you would expect.

Also note that because we're giving the motors() function exactly opposite numbers, whenever Wink starts to turn, he actually stops moving forward and just spins back and forth in his own footprint. The only way he will drive forward is if the sensors are reading nearly identical values.

If we make the loop run much faster, the sensors will be read more often and the motor speeds will be updated more often. Let's change the delay before reading the sensors to the much shorter 250 microseconds amount, then make the loop run much faster. We'll wait only 5 milliseconds between reading instead of the 100 milliseconds we are waiting in the previous example. (The companion code has the full code, I'm only showing you the two lines to edit below).

```
delayMicroseconds(250); //use shorter delay before sensor read

delay(5);               //much shorter loop delay
```

} Before reading sensors, change from `delay(4)` to `delayMicroseconds(250)`

} Change delay time near the bottom of loop

Wink_Ch12AdvLightSeek_Ex03

You will see this works much better. Mostly because the motors are being updated 20 times faster. You may however still notice that if you get the light very close to Wink, he still wants to just move side to side instead of moving forward.

In the motors() function, instead of making the motors run opposite speeds so Wink spins, lets give values of 70 and 20, so he will turn but both motors will keep moving forward at all times.

```
  if (rightLight-10 > leftLight)    //if right is greater
  {
    motors(70,40);                  //slow down right motor
  }
  else if (leftLight-10 > rightLight)  //if left is greater
  {
    motors(40,70);                  //slow down left motor
  }
```

} Change the -70 to 40

} Change the -70 to 40

Wink_Ch12AdvLightSeek_Ex04

"Control Loops"...

What we have created with our light seeking examples is called a "control loop". A control loop is a program that reads at least one input (like a sensor), then changes at least one output (like a motor) based on the input, then it repeats.

By experimenting with the previous examples, we have seen that making changes to different parts of the control loop can have significant effects on how the robot operates. By changing the speed of the control loop, we make the output (like the motor speed) update more often, so the amount of "over shoot" between readings is much less. We can also see that the amount of change to the motor speed, like using motors(70,40) instead of motors(70,-70) also has a significant effect.

If we make the change in motor speed very large, Wink will turn more sharply, but he will be more likely to over-shoot because he turns faster. But if we make the change much smaller as in the previous example, he won't over shoot so much because he is making a weaker turn. However this causes a new problem (that you may have already noticed) because he may not turn fast enough to actually follow the light.

Experiment with different turning strength numbers on your own for a while to see how different numbers change how Wink drives and responds to the light.

"Dynamic" Control Loops...

A "dynamic" control loop is a loop that changes over time based on something. In our previous example, the turn strength is set by your code and cannot be changed while the robot is running. No matter how much light is on one side or another, the turn is always the same. Sometimes when facing right into the light, where both sensors read almost the same amount, the turn should be less strong. But if you shine a light on the side of Wink, and one sensor is reading much more than the other sensor, Wink should make a stronger turn to try to face the light more quickly.

What we really want is for Wink to turn strong when needed, but tend to drive more straight with less turning force when a strong turn is not needed. We need to find a way to make this turn strength change automatically as the loop runs.

How can we do this?

If you're in a group, stop here and discuss what we could do to make this work better, or stop and think about it on your own for a while.

In our previous examples, the control loop only has three choices. Wink can turn at a certain strength to the left or to the right, or he can drive straight. In order to solve the problem, we really need this turn strength to change as the program runs. Sometimes Wink should turn more strongly and sometimes he should turn less strongly.

We could possibly do this by adding a lot more “else if” statements, but that’s not very efficient. It takes a long time to write the code and it is hard to edit.

What would be a much better solution is for the program to automatically adjust the motor speed based on the light level. When Wink is facing directly into a light, both the left and right ambient sensors should read about the same amount because an equal amount of light is falling on each sensor. But as you move the light source more to the side of Wink, the sensor on that side will start to read much higher than the sensor on the opposite side.

If the sensors are reading close to the same amount, we would want the motors to drive about the same speed. If one sensor is reading much higher than the other, then there must be a bright light on that side so Wink should make a strong turn that direction.

Planning the new program...

This new program is going to be a bit more complicated than the previous example, so let’s consider what we will need to do, then we’ll write some code that does what we want.

1. First we need to measure the left and right ambient light sensors. We will then need to figure out the difference between the two readings. We will probably need variables for all three of these.
2. We will need to make Wink start driving at some “normal” speed.
3. We will need to figure out which of the two sensors was reading the highest amount.
4. We will need to either speed up or slow down one or both of the motors based on the difference in the sensor readings from step 1. This way, as the difference in the sensor readings increases, the effect of the turn will also increase.

Let’s start with a normal motor speed of 70. If the left sensor reads higher, we will subtract the sensor difference from the speed of the left motor, which will make Wink turn toward the left. We’ll do the opposite if the right sensor reads higher.

If you want a challenge, try to write this program yourself. You already know everything necessary to do this. At least consider or discuss with your group for a few minutes before moving forward.

If you try to write your own, here’s a quick tip. the abs() function returns the absolute value of anything inside the parentheses, which is helpful to make a negative number become positive.

```
sensorDiff = abs(leftSensor-rightSensor);
```

} This line of code automatically calculates the sensor difference based on the actual left and right sensor values. The abs() function deals with the case where the right sensor is larger, which will result in a negative number. The abs() function tells the difference between the sensors, but not which one is greater than the other.

Light seeking with motor speed based on the difference between the sensors...

```
int leftSensor, rightSensor;  //declare the variables
int sensorDiff;               //difference between sensors
int baseMotorSpeed = 70;      //declare and set base speed

void loop(){

  eyesOff();              //make sure eyes are off
  delay(4);               //longer 4ms delay before sensor read
  leftSensor = analogRead(AmbientSenseLeft);        //read left
  rightSensor = analogRead(AmbientSenseRight);      //read right
  eyesGreen(100);         //turn eyes back on

  sensorDiff = abs(leftSensor-rightSensor);  //get difference

  if(rightSensor>leftSensor){      //if the Right is greater
   motors(baseMotorSpeed,(baseMotorSpeed-sensorDiff)); //go right
  }
  else{                            //else
   motors((baseMotorSpeed-sensorDiff),baseMotorSpeed); //go left
  }

  delay(20);              // short delay before repeating

} //end of loop()
```

} Declare variables. baseMotorSpeed is set during variable declaration.

} Turn off eyes, wait sensor settle delay, then read both left and right sensors. Turn eyes back on right away so your human eyes see less flicker.

} Figure sensorDiff. The abs() function returns the absolute value of the result of subtracting rightSensor from leftSensor.

} Subtract sensorDiff from baseMotorSpeed to cause a turn to happen.

} This delay sets how fast the loop repeats.

Wink_Ch12AdvLightSeek_Ex05

Load up this example and see what happens. It looks like it's working pretty well to me, except that when Wink looks straight into a bright light from a flashlight, he still tends to sit still and quickly wag side to side. If I shine the light on him just right, I can actually make him slowly walk backward while wagging side to side.

Why is this happening?

Consider what we're doing to the motor speed inside the motors() functions. We are subtracting the sensor difference from a motor on one side. This sensor difference could become quite large if a bright beam of light is shining on one sensor. If a large number (like a few hundred) is subtracted from a small number (like 70), it will make the speed for one motor go backward, and maybe even the maximum speed backward (-255).

If we could somehow make sure the sensorDiff is never greater than the base motor speed, we could make sure the motor never goes backward. This would make sure Wink always moves forward no matter what.

I wonder if there is an easy way to limit how large or small a value can be inside a variable. That could be really helpful right now.

Arduino function reference...

You may wonder where we found the neat abs() function used in the last example. Sometimes you may have thoughts like, “I’d really like to be able to limit a variable like sensorDiff to 70. How can I easily do this?” It turns out the nice people at Arduino have published a handy function reference. Visit the official Arduino website at www.arduino.cc Then navigate to “Learning” then “Reference”. This is an EXCELLENT page to book mark in your web browser. This page lists all the official Arduino functions you can use in your code. Click on any of them to see how they work.

There is a useful function we can use called constrain(), which basically takes a number or variable, and “constrains” it to between a lower and upper limit. If you give it the number 100 and tell the function to constrain it between 0 and 10, it will return 10 as the result. It automatically raises or lowers the value far enough to make it equal to the lower or upper limit of the range you specify.

Constrain works like this...

```
result = constrain( startingValue, lowerLimit, upperLimit );
```

} Using constrain

So after we get our sensorDiff value, we could use constrain() to make sure the value we subtract from the ‘slow’ motor is no more than the motor speed of 70. Like this...

```
sensorDiff = constrain( sensorDiff, 0, 70 );
```

} Constrain sensorDiff to between 0 and 70

This would work fine, but as long as we’re here, we should realize that we’ll probably be adjusting our baseMotorSpeed variable a few times. If we write the constrain() as above, we’ll have to constantly change the “70” value. So instead of writing “70”, let’s just use the baseMotorSpeed variable directly. Like this...

```
sensorDiff = constrain( sensorDiff, 0, baseMotorSpeed );
```

} Constrain sensorDiff to between 0 and whatever we have baseMotorSpeed set to

We can add this line of code to our previous example right after the sensorDiff is calculated. The entire new example is on the next page.

```
int leftSensor, rightSensor;  //declare the variables
int sensorDiff;               //difference between sensors
int baseMotorSpeed = 70;      //declare and set base speed

void loop(){

  eyesOff();              //make sure eyes are off
  delay(4);               //longer 4ms delay before sensor read
  leftSensor = analogRead(AmbientSenseLeft);        //read left
  rightSensor = analogRead(AmbientSenseRight);      //read right
  eyesGreen(100);         //turn eyes back on

  sensorDiff = abs(leftSensor-rightSensor);  //get difference
  sensorDiff = constrain( sensorDiff, 0, baseMotorSpeed );

  if(rightSensor>leftSensor){      //if the Right is greater
   motors(baseMotorSpeed,(baseMotorSpeed-sensorDiff)); //go right
  }
  else{                            //else
   motors((baseMotorSpeed-sensorDiff),baseMotorSpeed); //go left
  }

  delay(20);              // short delay before repeating

} //end of loop()
```

Wink_Ch12AdvLightSeek_Ex06

} Use constrain() to make sure sensorDiff is not greater than the baseMotorSpeed

This works much better. I'd say Wink is now doing a pretty good job of seeking the light. As you play with this example, keep a few things in mind.

1. Remember that Wink's ambient sensors are most sensitive seeing light along the running surface. They are less sensitive if you shine light from directly above him, and the eyes do shadow the sensors a bit. To see the best reaction, place your flashlight flat on the running surface and point it toward Wink. You will find he responds much better.
2. Wink can see sunlight very well. If you have a window or open door nearby, you will notice he really likes to go toward it. Your light source needs to over-power this external light. If your room is really bright, you may need to close your blinds so the room is darker. Wink is less sensitive to some flashlights (especially LED type flashlights) but normally this is not a problem.
3. You may notice that under even lighting Wink tends to want to always steer a certain direction. When the sensors are manufactured, there is a very slight variance between them. So even in perfectly even light, you may notice that one sensor always reads slightly higher than the other. You can test this with Serial.print functions. If you notice the right sensor is always 5 or 10 counts higher for example, you can add an extra line of code after the sensor reading that subtracts the 5 or 10 counts from the reading so they're more equal in even lighting.

Adding more variables to the control loop...

So Wink is following the light pretty well now. It's fun to play with at this point, but we could do a few more things to give him a bit more personality.

Let's pretend that Wink lives on light - like light is his food. Maybe we assume that when there is no light, Wink will rest by sitting still, and when he begins to see some light, he tries to follow it. We can also say that as he sees more and more light, that he runs faster and faster to try to catch it.

Can we do a few more things to our program to make this happen?

We can measure the center ambient sensor as well as the left and right, and use the amount of light on the center sensor as the basis for his motor speed. We can also say that if the light on the center sensor is below a certain amount, he sits still to conserve energy.

Try to work this up on your own if you like. It's a good challenge.

I put my own example on the next page by itself because it's rather long. Here is a list of some of the things I did that may help in writing your own example.

1. I've added the variable and step of reading the center sensor.
2. I've calculated a "speed boost" based on the reading of the center sensor. Remember motor speed can only go up to 255, but the sensor can read up to 1024. If I just add the sensor reading to the motor speed, it will very quickly max out. To bring the sensor reading down into a range more compatible with the motor speed, I've divided the center sensor reading by 5, then constrained that result to a max of 250. I then figure the motor speed we are actually going to use by adding this speed boost to the base motor speed. I have also set the base motor speed much lower than before (20 in this case) because the "speed boost" is going to make up much of the final motor speed. You can play with the amount you divide the speed boost and where you constrain it to see what different effects these have.

   To divide in C, use the forward slash like this: `/` To multiply in C, use the `*` asterisks symbol.

   Remember that when you do division on an "int" type variable (which is what we're using for sensorDiff), there will be no decimal left over. Any decimal remainder will be dropped, because an "int" type variable cannot store a decimal.
3. I've added a while() loop that stops motors and turns off the eyes when the center sensor reading is very low, so Wink will rest when no light is present.
4. I've used the new "motorSpeed" variable in the functions that actually drive the motors. Having "motorSpeed" as its own variable, you can do other math to this value in your code without having to do the math inside the motors() functions themselves. This works the same both ways, but makes the code more easy to read and edit by calculating motorSpeed on its own lines before the motors() functions.

```
int leftSensor, rightSensor, centerSensor;
int speedBoost, motorSpeed;
int sensorDiff;
int baseMotorSpeed = 20;
int restingLightLevel = 10;

void loop(){

  eyesOff();
  delay(4);
  leftSensor = analogRead(AmbientSenseLeft);
  rightSensor = analogRead(AmbientSenseRight);
  centerSensor = analogRead(AmbientSenseCenter);
  eyesCyan(100);

  sensorDiff = abs(leftSensor-rightSensor);
  sensorDiff = constrain( sensorDiff, 0, baseMotorSpeed );

  speedBoost = centerSensor / 5;
  speedBoost = constrain(speedBoost,0,250);
  motorSpeed = baseMotorSpeed + speedBoost;

  while(centerSensor < restingLightLevel){
    eyesOff();
    motors(0,0);
    centerSensor = analogRead(AmbientSenseCenter);
    delay(100);
  }

  if(rightSensor>leftSensor){
   motors(motorSpeed,(motorSpeed-sensorDiff));
  }
  else{
   motors((motorSpeed-sensorDiff),motorSpeed);
  }

  delay(20);

} //end of loop()
```

Wink_Ch12AdvLightSeek_Ex07

} Base motor speed under low light and the "resting" light threshold. Adjust this restingLight level for your own room. If your room is brighter and Wink won't stop moving, raise this amount a bit until you find the level where he will stop moving.

} Read all three sensors.

} Get sensorDiff and constrain it. You can change "baseMotorSpeed" to a hard number like "40" to allow Wink to make stronger turns.

} Divide down centerSensor for speedBoost then constrain it. Add speedBoost to baseMotorSpeed to get final motorSpeed used below.

} Sit still with eyes off while centerSensor is less than restingLightLevel from above

} Set driving motor speeds

} Delay before loop repeats

3 SKILL LEVEL

A few final touches...

We've almost got it. The previous example is working really well. I can think of a couple more things that will make it even better then you're on your own.

I think it would be interesting to make Wink's eyes get brighter the closer he gets to the light. I also think it would be fun to have his eyes turn red and his sound chirper "scream" as Wink attacks the light.

```
int leftSensor, rightSensor, centerSensor;
int speedBoost = 0;
int sensorDiff, motorSpeed;
int baseMotorSpeed = 20;
int restingLightLevel = 10;
int screamLight = 900;

void loop(){

  eyesOff();
  delay(4);
  leftSensor = analogRead(AmbientSenseLeft);
  rightSensor = analogRead(AmbientSenseRight);
  centerSensor = analogRead(AmbientSenseCenter);
  eyesCyan(speedBoost);

    //.. removed to save space on this page (see companion code)

  if(centerSensor > screamLight){
   eyesRed(250);
   digitalWrite(Beeper, HIGH);
  }
  else{
   digitalWrite(Beeper, LOW);
  }

  if(rightSensor>leftSensor){
   motors(motorSpeed,(motorSpeed-sensorDiff));
  }
  else{
   motors((motorSpeed-sensorDiff),motorSpeed);
  }

  delay(20);

} //end of loop()
```

Wink_Ch12AdvLightSeek_Ex08

speedBoost is now declared as 0 so it is zero the first time through the loop. We've added screamLight threshold at 900.

Eye brightness is now based on speedBoost

Make eyes red and turn on sound chirper to "scream" if centerSensor level is greater than the screamLight threshold level. Else turn sound chirper off. Eyes will automatically re-set to the normal color the next time through loop().

The digitalWrite() Arduino function accepts a pin name, and setting it "HIGH" turns it on, and setting it "LOW" turns it off. The pin name for Wink's sound maker is "Beeper".

3 SKILL LEVEL

Advanced Light Seeking

Lesson 12

It's working great! Here is one final "extra credit" example that takes it one step further with some math.

Things are working really well except for one thing that is bothering me. You'll notice that when shining a light near the side of Wink, he doesn't turn very fast. This is because we are constraining sensorDiff to baseMotorSpeed, which is quite low in this example. This means that Wink can only have a turn strength of 20, which isn't very strong.

One simple way to make the turn strength stronger is just to use a hard number like "80" in the sensorDiff constrain (instead of baseMotorSpeed as it is now), but we can do something even more dynamic here.

I'd like him to turn stronger when the light level is lower, this way he'll quickly rotate to face any light, but once he starts to rush toward the light, I want the turn strength to be lower so he doesn't wag side to side as much.

This will take a few steps of math but we can figure it out. We'll create yet another new variable called turnBoost. When the light level is low, I want to give a boost of an extra 50 to turn strength so he can turn more quickly. But when the light level is high (the center sensor is reading toward the top of the range), I want the boost to be very low - like zero.

This is a range of 50 over a possible light sensor range of about 1000. So if we divide the center light sensor value by 20, then subtract that amount from 50, we should have about the number we're looking for.

```
turnBoost = 50 - (centerSensor / 20);
turnBoost = constrain( turnBoost, 0, 50 );
```

This math should make turnBoost near 50 when light level is low, and near zero when light level is high.

```
if(rightSensor>leftSensor){
 motors((motorSpeed+turnBoost),(motorSpeed-sensorDiff-turnBoost));
}
else{
 motors((motorSpeed-sensorDiff-turnBoost),(motorSpeed+turnBoost));
}
```

Add the turnBoost to the motors() function, so we drive the motorSpeed, minus the sensorDiff, minus an additional amount of turnBoost. We will also add a positive turnBoost to the opposite motor. Do this for motors() in both cases.

We'll subtract turnBoost from the 'slow' side, but also add it to the 'fast' side, that way Wink wants to spin a bit more. You'll notice he's a bit more twitchy now so you can play with the max amount in the turnBoost constrain, or divide the centerSensor by a larger number, which will make turnBoost say smaller and thus have less effect.

I've made the edits to the following example. Remember the entire Arduino code can be found in the Companion Code archive.

I would suggest you play with the numbers and see if you can make a new light seeking behavior of your own. There are thousands of ways you could make Wink chase light (or run away from light?). You've learned enough now to really make that happen!

```
int leftSensor, rightSensor, centerSensor;
int speedBoost = 0;
int turnBoost;
int sensorDiff, motorSpeed;
int baseMotorSpeed = 20;
int restingLightLevel = 10;
int screamLight = 900;

void loop(){

  eyesOff();
  delay(4);
  leftSensor = analogRead(AmbientSenseLeft);
  rightSensor = analogRead(AmbientSenseRight);
  centerSensor = analogRead(AmbientSenseCenter);
  eyesCyan(speedBoost);

    //.. removed to save space on this page (see companion code)

  turnBoost = 50 - (centerSensor / 20);
  turnBoost = constrain( turnBoost, 0, 50 );

  if(centerSensor > screamLight){
   eyesRed(250);
   digitalWrite(Beeper, HIGH);
  }
  else{
   digitalWrite(Beeper, LOW);
  }

  if(rightSensor>leftSensor){
   motors((motorSpeed+turnBoost),(motorSpeed-sensorDiff-turnBoost));
  }
  else{
   motors((motorSpeed-sensorDiff-turnBoost),(motorSpeed+turnBoost));
  }

  delay(20);

} //end of loop()
```

} Declare new turnBoost variable

} Add new code to calculate turnBoost based on centerSensor

} Add new turnBoost value to the existing motors() functions for both cases

Wink_Ch12AdvLightSeek_Ex09

3 SKILL LEVEL