# Adaptive monitor bias lighting with Arduino

**LAURA SCOCCIANTI**

*BSc in Computer Science at University of Trento*
*laura.scoccianti@gmail.com*
*http://laurascotch.github.io*

**Abstract:**

Bias lightning for PCs is something extremely important in order to care for our eyes as much as possible. Many people use their computer in complete darkness for large time periods, and the difference in luminance between the monitor and the surrounding dark environment creates a contrast that is difficult for eyes to cope with.

This simple project describes my personal solution to this problem: an Arduino controls a LED strip attached behind a monitor and sets its brightness and color based on the light condition of the surrounding environment.

## 1. Introduction

What is bias lighting for PCs? Simply put, it's a light (usually a LED strip) placed on the rear of a monitor that creates a sort of halo effect behind it. This is not a merely aesthetic solution: in fact it has been studied for reducing eye strain and fatigue when staying in front of a monitor for a large amount of time in low light condition.

What is really harmful to the eyes when working in front of a PC in low light condition is the sharp contrast between the monitor luminance and the surrounding darkness. This problem could be mitigated by using a desk lamp, but this solution is often not optimal: usually such lamps are either "of" or "off" and they may provide too much light, not to talk about maybe having a color that may not be relaxing for the eye (e.g. blue-ish neon lights).

To add on this, it often happens that we start working on our PCs in good light conditions, but then as hours go by, we find ourselves in darkness, maybe being too focused on our job to take the time to have a break and switch up some light. This in particular is what gave me the idea to develop this project.

The final product is a fully functional prototype of an adaptive monitor bias lighting: the brightness and color of the LEDs is set based on the room's light condition, that is gathered via a photoresistor.

## 2. Getting to know the scenario

### *2.1. Environment*

I developed this project in my room. The desk is placed in front of a white wall, that therefore faces the back of the monitor where I attached the LED strip. On the same wall, there's a big window. On the right, approximately 80cm above the table there's a light that is a bit too powerful and in fact often causes me headaches after hours of work.

The window is south facing and sun shines through it from early in the morning until 12:00 in winter and 13:30 in summer, when it gets covered by a building. During the afternoon, natural lighting gradually decreases and it's already insufficient at 15:30 - 16:00.

A conceptual visualization of the environment is show in Figure 1.

### *2.2. Data gathering*

The basic idea for the project is that we want the LEDs to emit a certain light given the condition of the room. In order to understand the luminosity values to use as threshold, I kept track of the luminosity of the room on a quite sunny day from 10:45 to 18:00. In order to do this I placed the

Fig. 1. Representation of the setting

photoresistor in the exact same place where it will be placed when the full project will be finished, that is just under the monitor, facing me and the rest of the room; this way, when there will be LEDs behind the monitor, they shouldn't influence too much the gathered luminosity values.

Figure 5 shows how natural light inside the room changes during the day, based on the gathered data.

### *2.3. Lighting bias*

Lighting bias is the phenomenon for which our perception of what's on screen is influenced by the background light, so it literally bias the brain's interpretation. To give a concrete example, shining an orange light behind the monitor will make us perceive everything on-screen as slightly more orange. This works with every other color.

Because of this, it is useful to have the color of the LEDs adapt to the time of the day, rather than (just) the brightness: I will make it shift gradually from a reassuring and calming light blue towards warmer colors. I opted for this solution because using warmer lights at the end of the day is more resting for the eyes. Moreover, there are studies that states that the violet/blue light emitted from digital devices delays sleep.

The colours I choose to show are depicted in Figure 5.

## 3. The hardware

The hardware for this project is:

- Arduino UNO (or Leonardo, Nano, Micro, no real difference - to be fair I used a Leonardo because my UNO is already busy)
- 1x photoresistor
- 1x 220 Ω resistor
- NeoPixel LED Strip (WS2812 chip)

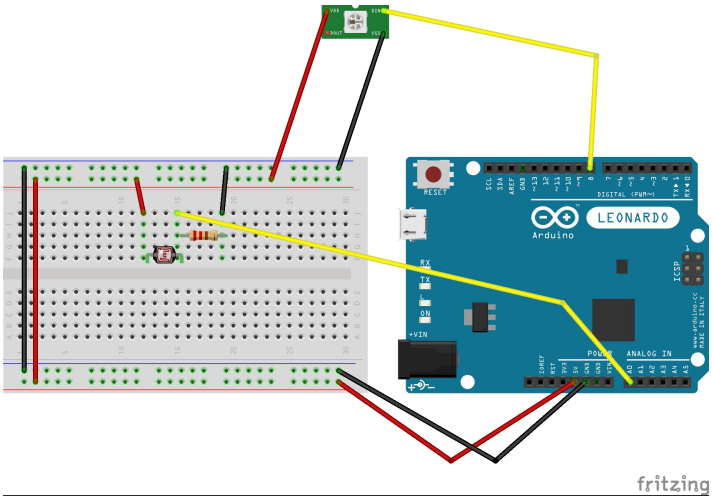



Fig. 2. Circuit representation

The following picture shows the schematics.

The resistor is connected to analog pin `A0`, while the LED strip is connected to pin `8`. The Arduino is powered via USB.

## 4. The software

The code is based on three variables: `lightVal`, `lightAvg` and `lightSet`.

`lightVal` is the variable where the raw luminosity data is read and stored. This value is gathered every 2 seconds and is added to a queue of 8 items.

`lightAvg` stores the average value of the last 8 measurements, which are stored in the already said queue.

`lightSet` is the luminosity value used to set the LEDs. It is calculated every 7 cycles, with the following formula:

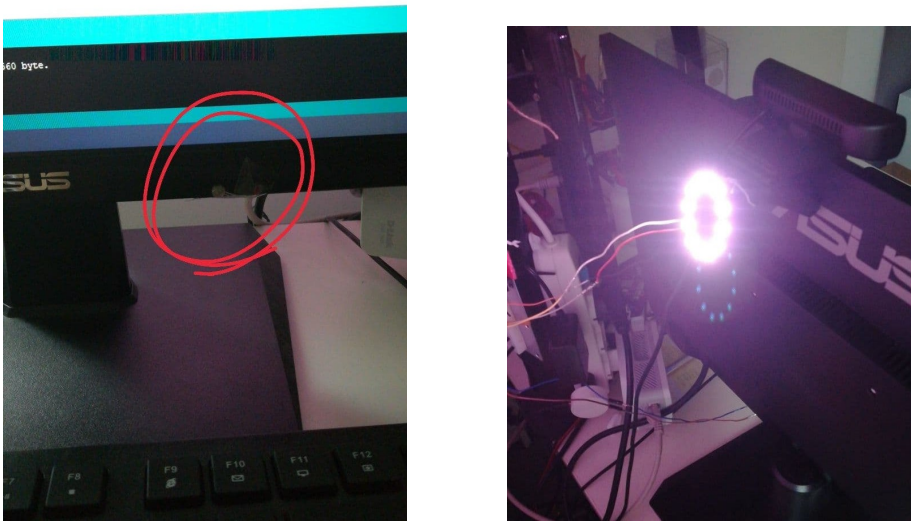
$$lightSet = 0.0107 \cdot lightAvg^2 + 0.3532 \cdot lightAvg + 0.4365$$

Why can't we just use the "raw" average? Because the LEDs being switched on slightly influence the luminosity measurement and the "raw" value would cause them to repeatedly switch on/off or between different colours. After some observations, I was able to model the difference between the measured luminosity and the actual one using that formula, obtained with the `polyfit` function in Python.

The size of the queue (8) and the number of cycles (7) before updating the LED strip are chosen so that:

- There's always some older value in the queue that influences the average between the updates
- 7 cycles are enough to have a stable estimate of the light needed in "natural" conditions, but are also reactive enough in the case that a light in the room is switched on (or off).

## 5. Photos of the prototype



(a) The photoresistor (b) The LED strip

Fig. 3. These two photos show how the LEDs are mounted behind the monitor, while the photoresistor is in front of it, so to not have it influenced by the light

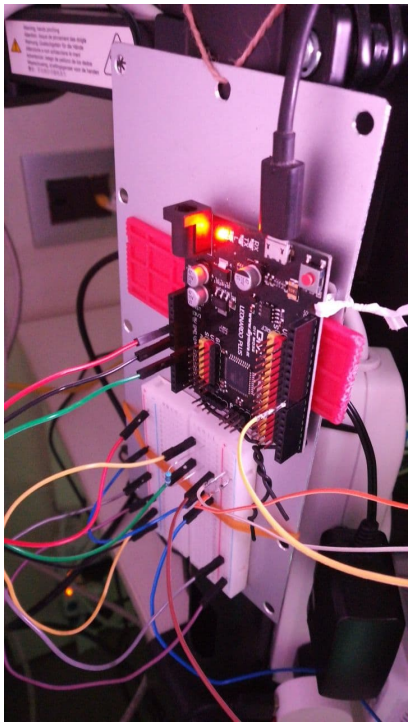


Fig. 4. The "brain": Arduino and breadboard

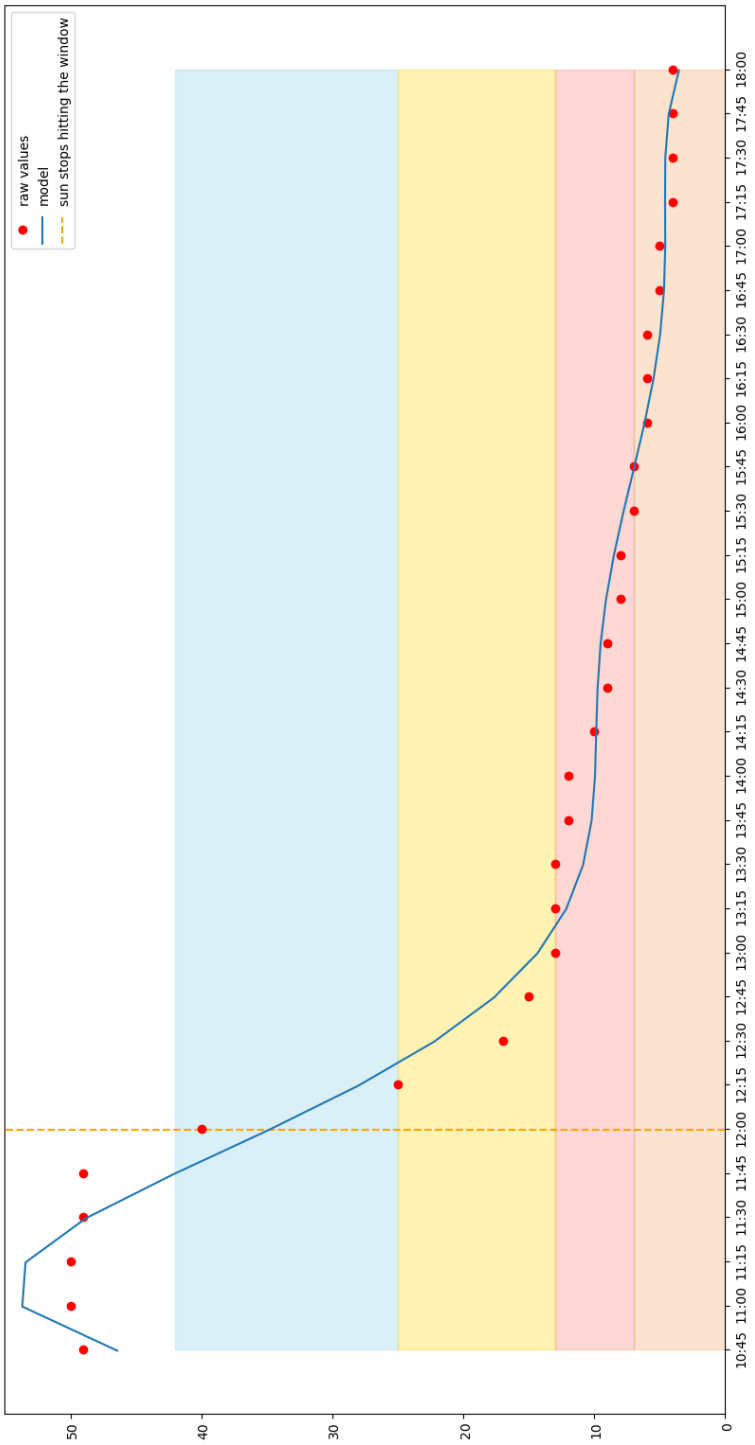



Fig. 5. Room luminosity during day

## 6. Code

```
#include <Adafruit_NeoPixel.h>
#include <ArduinoQueue.h>

ArduinoQueue<int> lumin(8);
Adafruit_NeoPixel led1 = Adafruit_NeoPixel(12, 8, NEO_GRB + NEO_KHZ800);

const int PERIOD = 2;
const int CNT = 7;

int count = CNT;
int lightVal;
float lightAvg;
int lightSet;

float lastAvg;

bool first_avg = true;
bool need_change = false;

uint32_t pale_red = led1.Color(255, 80, 70);
uint32_t pale_orange = led1.Color(255, 111, 20);
uint32_t light_blue = led1.Color(54, 68, 93);
uint32_t orange = led1.Color(255, 100, 0);
uint32_t black = led1.Color(0, 0, 0);
uint32_t white = led1.Color(255, 255, 255);

void setup() {
  led1.begin();
  rainbow(10); // check that LEDs are working
  //delay(500);
  led1.fill(black); // turn off LEDs
  led1.show();
  pinMode(A0, INPUT); // photoresistor
  lightSet = analogRead(A0);
  setLight(lightSet);
}

void loop() {
  count--;
  lightVal = analogRead(A0);
  if(lumin.isFull()){
    lumin.dequeue();
  }
  lumin.enqueue(lightVal);
  lightAvg = average();
  if(count == 0){
    if(first_avg){
      lastAvg = lightAvg;
      first_avg = false;
      // following model not suited for big values (>100)
      lightSet = (int)((0.0107*pow(lightAvg,2)) + (0.3532*lightAvg) + 0.4365);
      if(lightSet<0) lightSet = 0;
      setLight(lightSet);
    } else {
      // I want the new average to be fairly different to
      // the last one used to set the LEDs
      if(lightAvg - lastAvg > 1.5 || lightAvg - lastAvg < -1.5){
        // following model not suited for big values (>100)
        lightSet = (int)((0.0107*pow(lightAvg,2)) + (0.3532*lightAvg)+0.4365);
        lastAvg = lightAvg;
        if(lightSet<0) lightSet = 0;
```

```
        setLight(lightSet);
      }
    }
    count=CNT;
  }
  // check ambient light every PERIOD seconds
  delay(PERIOD * 1000);
}

float average(){
  int tmp;
  int sum = 0;
  int s = lumin.itemCount();
  int q[s];
  for(int i=0; i<s; i++){
    tmp = lumin.dequeue();
    q[i] = tmp;
    sum = sum + tmp;
  }
  for(int i=0; i<s; i++){
    lumin.enqueue(q[i]);
  }
  return (float)sum/s;
}

void setLight(int l){
  uint32_t c;
  if(l > 42){
    c = black;
  }
  if(l < 43){
    led1.setBrightness(255);
    c = light_blue;
  }
  if(l < 25){
    led1.setBrightness(255);
    c = orange;
  }
  if(l < 13){
    led1.setBrightness(255);
    c = pale_red;
  }
  if(l < 7){
    led1.setBrightness(255);
    c = pale_orange;
  }
  uint32_t cc = led1.getPixelColor(0);
  if(led1.getPixelColor(0) != c){
    colorFade(c);
  }
}


void colorFade(uint32_t finalColor){
  uint8_t b = finalColor & 0xFF;
  uint8_t g = (finalColor >> 8) & 0xFF;
  uint8_t r = (finalColor >> 16) & 0xFF;  // separate into RGB components
  uint8_t origR, origG, origB;
  uint32_t orig = led1.getPixelColor(0);
  uint32_t tmp;
  origB = orig & 0xFF;
  origG = (orig >> 8) & 0xFF;
  origR = (orig >> 16) & 0xFF;  // separate into RGB components
```

```
  while((origR != r) || (origG != g) || (origB != b)){
    if(origR<r) origR++; else if(origR>r) origR--;
    if(origG<g) origG++; else if(origG>g) origG--;
    if(origB<b) origB++; else if(origB>b) origB--;
    tmp = led1.Color(origR, origG, origB);
    led1.fill(tmp);
    led1.show();
    orig = led1.getPixelColor(0);
    origB = orig & 0xFF;
    origG = (orig >> 8) & 0xFF;
    origR = (orig >> 16) & 0xFF;  // separate into RGB components
    delay(30);
  }
}

// May use these effects some time in the future, who knows
void rainbow(uint8_t wait) {
  uint16_t i, j;

  for(j=0; j<256; j++) {
    for(i=0; i<led1.numPixels(); i++) {
      led1.setPixelColor(i, Wheel((i+j) & 255));
    }
    led1.show();
    delay(wait);
  }
}

uint32_t Wheel(byte WheelPos) {
  if(WheelPos < 85) {
   return led1.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
  } else if(WheelPos < 170) {
   WheelPos -= 85;
   return led1.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  } else {
   WheelPos -= 170;
   return led1.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
}
```

## References


1. Power Practical, “What is TV and Monitor Bias Lighting?” https://powerpractical.com/pages/what-is-tv-bias-lighting.
2. Windows Central, “Bias lighting is an easy and affordable upgrade every PC user should make” https://www.windowscentral.com/bias-lighting-your-pc-monitor.
3. University of Oxford, “Lighting colour affects sleep and wakefulness” https://www.ox.ac.uk/news/2016-06-09-lighting-colour-affects-sleep-and-wakefulness.