**DIY Chemistry lab: building a spectrophotometer**

*by Robert J. LeSuer*

This article describes how to build a simple photometer that can be used, for example, to quantify the amount of food coloring in beverages. It uses the Wolfram Language to collect, visualize and analyze the results.

**Introduction**

I was in the beverage isle of the supermarket one day and was looking at the soft drinks. One of the drinks, a "Watermelon Punch" had listed on it Red 40 as one of the ingredients. Red 40, also known as allura red AC or E129, was phased out of the UK by the Food Standards Agency in 2009 because some research showed that it may induce hyperactivity in children. The food dye is still allowable in the US market, but if questions have been raised about the safety of the dye, I'd like to know how much of it I'm consuming.

I'm an analytical chemistry professor and one of the fun parts about my job is that I get paid to teach students how to play with instruments: we use them, we break them, we repair them and sometimes we even build them. One major area of analytical chemistry is called spectroscopy, which uses light (or more broadly, the electromagnetic spectrum) to explore the properties of atoms, molecules and materials. A spectroscopic instrument commonly found in chemistry labs is the visible spectrophotometer, which helps a scientist learn how materials interact with light of different colors.

My goal is to create a simple Raspberry Pi based spectrophotometer that can help me quantify the amount of Red 40 in the Watermelon Punch. Since I may want to turn this project into a laboratory experiment for my chemistry students, I'd like to keep the hardware requirements simple and the programming requirements accessible to students who may have never programmed before.

**Supplies**

All of the electronic parts should be available at your local electronics store or an online retailer. I purchased my items either at Newark (www.newark.com) or Adafruit (http://www.adafruit.com/)

* Green LED (20 mA max)
* CdS photoresistor (Adafruit product no. 161, Newark product no. 95F9039)
* 100 uF capacitor

The science equipment I borrowed from my personal lab, but on line retailers such as Edmund Scientifics (www.scientificsonline.com) and the Labware section of Amazon.com are good sources for these items

* spectrophotometer cuvettes
* graduated cylinder (10 to 25 mL worked well for me)

The soft drink I used is from Snapple (Watermelon Punch) and to create my standard solutions I used McCormick egg dye.

**Instrument design**

The basic parts of a spectrometer are a source, the sample and the detector. For this project I'm only interested in measuring one color (red) so I can simplify the source by using a green LED. The detector I'll use in this experiment is a CdS photocell that has a resistance dependent upon the amount of light shining on it. Because the Raspberry Pi doesn't have an analog input, I'll be using some suggestions from Adafruit (http://learn.adafruit.com/basic-resistor-sensor-reading-on-raspberry-pi) for making analog measurements. I'll connect the photocell to a capacitor and "ping" the circuit - essentially measuring how long it takes for the capacitor to charge. Since the charging time is influenced by the resistance in the circuit, this time will be related to the amount of light hitting the photocell. To keep everything in one place, I'll use some Legos - which make for a relatively cheap and robust optical bench. Not shown (since it makes for a boring photo) is that the whole circuit is placed under a box; we only want our detector sensing light from the LED, and therefore we need to block out light from all other sources.

[Circuit diagram]

[Photo of setup]

**The Science**

White light, such as that from the sun, is composed of all the colors of the rainbow. Objects have colors based on how the molecules that make up those objects interact with the light. Typically, materials can do one of three things with light: absorb it, reflect it, or let it pass through (transmit it). When we see an object, say a red-colored soft drink, the molecules that make up the soft drink are absorbing all the colors *except for red*, which it transmits and reflects. If there are a lot of molecules that absorb those colors, then the drink will appear dark red; conversely, if there are only a small number of molecules that absorb, then the drink will appear light red. There is a relationship, called Beer's Law, which shows that the amount of light absorbed is proportional to the concentration of the absorbing species, and it is this law that makes spectrophotometry so powerful.

To perform this type of analysis, I need a calibration curve, which shows the relationship between the instrument output and the concentration of Red 40 dissolved in water. I took one drop of commercial red 40 dye and diluted it to 10 milliliters (mL). Since one drop is approximately 50 microliters (0.050 mL), this was a 200x dilution. The color of the Watermelon Punch looks like it falls between 1000x and 5000x dilutions of the dye.

[calibration solutions]

One way to make these solutions is to take 1 mL of the 200x solution and dilute it to 5 mL (making a 200x5 or 1000x dilution), then repeat this step four more times with larger final volumes. Below is a chart summarizing the process. Each solution is made by diluting 1 mL of the 200x stock solution

|  |  |  |
| --- | --- | --- |
| final volume (mL) | dilution | 1/dilution |
| 5 | 1000 | 0.00100 |
| 10 | 2000 | 0.00050 |
| 15 | 3000 | 0.00033 |
| 20 | 4000 | 0.00025 |
| 25 | 5000 | 0.00020 |

I used a slightly different dilution scheme than the one described here; however, the results are the same. We also need the reading from a cuvette with just water to account for light absorbed in the absence of dye; this is called the background. Ideally, when the detector output is plotted verses the inverse of the dilution factor, we should observe a linear relationship.

**The Software**

Now that the Wolfram Language is available to all owners of a Raspberry Pi, we have at our disposal a comprehensive system for data acquisition, analysis and visualization. Here I'll describe one way to use Mathematica as the front end of the spectrophotometer.

We need several functions to (a) discharge the capacitor, (b) read the status of the pin tied to our detector and (c) time how long it takes for the capacitor to charge. For this project, I wanted to do all the programming in Mathematica, which at the moment is not very fast when it comes to reading and writing to the GPIO, which is done through the functions DeviceRead and DeviceWrite. Here's the function **short[]** used to discharge the capacitor

short[] := Module[{},

DeviceConfigure["GPIO", 24 -> "Output"];

DeviceWrite["GPIO", {24->0}];

Pause[2];

DeviceConfigure["GPIO", 24-> "Input"];

]

**Module[]** is used to define local variables, although we have none in this function and Module is being used to keep the code tidy. The **short[]** function first configures GPIO pin 24 to serve as an output pin and a 0 is written to the pin. After a 2-second delay, the pin is reconfigured as input. This function serves to discharge the capacitor and prepare the detector for making a measurement.

In the next function, **readpin[]**, we could use Mathematica's **DeviceRead[]**; however, DeviceRead is a little too slow for the capacitor I'm using. I circumvent this problem by reading the state of our detector pin a different way:

readpin[] := Module[{pin, out},

pin = OpenRead["/sys/class/gpio/gpio24/value"];

out = Read[pin];

Close[pin];

out

]

Here, there are two local variables, **pin** and **out**; the former is used to open a file stream to the kernel treepath for the GPIO pins and the value is stored the latter. In the Wolfram Language, output is suppressed by the ";" at the end of a line, so the last statement in **readpin[]** makes the function return the value of **out**.

The last function wraps together turning the source on and off, discharging the capacitor and timing how long it takes to charge:

measure[] := Module[{np = 1200, data = {}, tth},

DeviceWrite["GPIO", {25 -> 1}];

short[];

data = AbsoluteTiming[Table[readpin[], {np}]];

tth = Quiet@If[NumberQ[#], #, np]/np \* First@data &[

Position[Last@data, 1, 1, 1][[1,1]]];

DeviceWrite["GPIO", {25 -> 0}];

tth

]

The three local variables in **measure[]** are: the number of points (np) or times the GPIO pin will be read; data, which is a place to store the pin values; and tth or the time needed for the GPIO be to go high. The **DeviceWrite[]** function turns on the LED source and then the detector is initialized with **short[]**. Next, the data are collected along with the time needed for that operation. Perhaps the most confusing line of code is next, which searches the data for the first instance of a one and determines how long it took for the GPIO pin to go high. Occasionally, no 1 will be found, indicating that insufficient light is reaching the detector, and a warning would be thrown if **Quiet** wasn't applied to the command. If you are trying this setup on your own, the value of np will probably need to be adjusted based on how well you block out stray light and the size of your capacitor.

Mathematica has a bunch of tools to create lists of data, average multiple datapoints, plot the data and perform a least squares fit. I don't have enough space in this article to go through the details; however the on-line documentation (reference.wolfram.com/language) has plenty of examples to work through and use.

**The analysis**

By measuring each sample five times and plotting the average output versus the inverse of the dilution factor, I obtained the results shown below. The linear relationship means I can be fairly confident in the results of the unknown measurement. The equation for a line is y = m x + b; so by rearranging I can obtain an equation for finding the "dilution factor equivalent" for the soft drink.

[Plot of data with linear fit]

These results indicate that the amount of Red 40 in Watermelon Punch is equivalent to taking one drop of the McCormick dye solution and diluting it by a factor of 3000. In order to get a more usable value, I analyzed the commercial dye in my lab and found it to contain approximately 11 g of dye per liter of solution. A little bit of math and we find that this is about 3.7 miligrams of dye per liter of soft drink. Using a commercial spectrometer, I obtained 4.6 mg/L, so the home-made spectrophotometer agrees reasonably well.

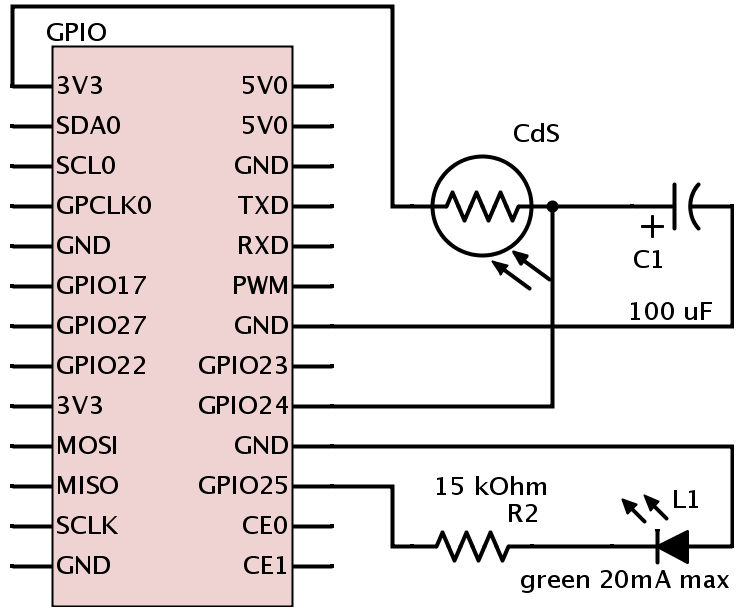
Presently, the recommended acceptable daily allowance (in the US) of Red 40 is 7 mg/kg body weight per day. It looks like I'd have to drink a large amount of this beverage before reaching my daily limit of Red 40.

**Where to go from here?**

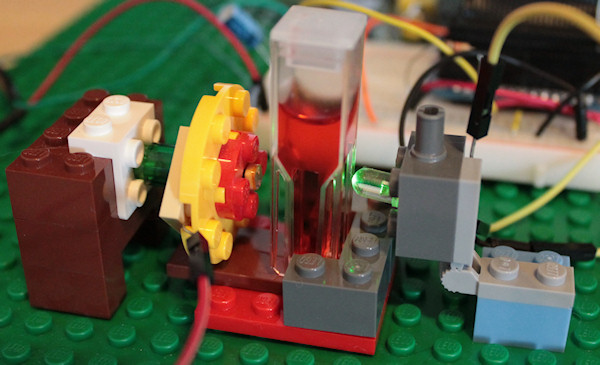
I only measured one of many different fruit drinks - which leaves many other samples to probe. Similar tests can be used for other food colorings which would require the use of different LEDs. There are also improvements to be made to the spectrometer itself: an ADC can replace the resistor/capacitor detector, an autosampler can be added; multiple LEDs can be incorporated into the design to allow for multi-dye analysis. The door to your Raspberry Pi driven home-chemistry lab is wide open and ready for discovery.

**Figures**

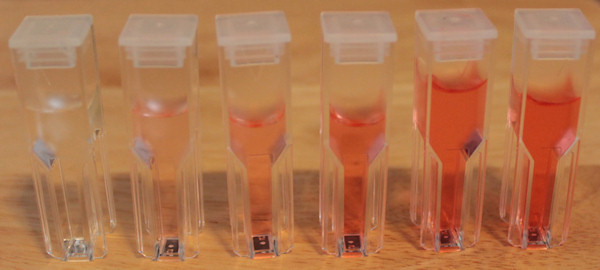
[Circuit diagram]

****

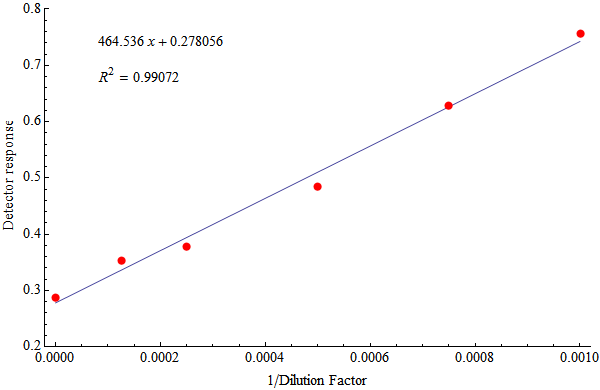
[Photo of setup] High res image attached to email.



[calibration solutions] High res image attached to email



[Plot of data with linear fit] This is a new figure which incorporates the background measurement.

****