# Instrument Project

Grant Saggars

February 9, 2024

## 1 Introduction

Over the last five weeks, I have worked to design an optical spectrometer. I had intended to apply my device to calculate the temperature of a blackbody using Planck's law. The quality of my materials proved to be a considerable constraint, and ,any problems arose during the design process, particularly in the last few weeks; as I had countless rage-inducing complications with fabricating the housing and mounts for my parts. This led to even more problems getting a stable and precise alignment, among other things. Despite the difficulty, the device is fully capable of visible spectrum measurements, and I had significant success in fitting Planck's Law to my data.

## 2 Methodology

### 2.1 Physical Design

My spectrometer is as simple as possible in design: light enters through a thin silt, is diffracted, and then the resulting spectral lines are captured by camera. My initial designs involved sophisticated mounts and precise CAD designs. These unfortunately became unrealistic due to technical difficulties with the 3D printer available to me. The design process concluded with a long night, a cardboard box, and a lot of tape. Despite the low quality of the materials, the device works rather well. The lower quality of housing materials has resulted in some internal instability, as the system can occasionally fall out of calibration due to rough handling or heat.

### 2.2 Software

Development of the software was significantly smoother, thankfully. The OpenCV library provided a very solid foundation for my code, which allowed for speed and flexibility in my own feature set, which in its final form includes a live video feed from the spectrometer, a corresponding spectral density plot, and calibration visuals. I am able to dump the measured information with a keybind at any time, which allows me to "export" a frame for analysis.

Had more time been available, I would have liked to have made the visuals prettier, and allowed for interactive calibration with the mouse. It has more than enough functionality for what I need, though.

### 2.3 Analysis

Because I wanted to analyze blackbodies with my device, process the data collected. I do this externally, partially because it is easier to work with a single "frame" of data, but also because it is useful to permanently save my data. For this, I implemented the ability to dump a snapshot or "frame" of what the spectrometer sees, which allows me to transfer my data to a CSV file.

Planck's law describes a spectral density distribution (what is measured by my spectrometer) in terms of a blackbody's temperature and wavelength (or other related properties such as frequency or wave number). Because I can calibrate for wavelength, it is relatively trivial to fit this distribution to my data. I have experience using the IMinuit library developed by CERN's ROOT team to fit distributions to datasets using a Chi-Squared cost function.

### 2.4 Data Collection

I analyzed two blackbodies with my spectrometer: a lightbulb of known color temperature, and the Sun.

Analysis of the lightbulb was very straightforward, as I only need shine the light from the lightbulb into my spectrometer.

The real test of my spectrometer has always been to measure the temperature of the sun with my spectrometer. The surface of the sun is known to be around 15 million degrees, so planck's law would indicate that the peak of such a distribution should be far into the ultraviolet spectrum. However, due to

atmospheric absorption and scattering, the observed wavelength on Earth has a peak in the visible spectrum, which makes sense because if we were not protected from ultraviolet light by the atmosphere we would be rather dead.

A quick google search revealed that the perceived blackbody temperature of the sun from earth is about 5770k as a consequence of these factors. It may be possible to produce estimates for the missing wavelength bands as a consequence of the atmospheric scattering using Rayleigh and Mie scattering distributions; however, I do not have enough time to make such estimates.

Measurements of the sun were taken at midday on an (unfortunately) cloudy day. This should not have had a very large impact on my measured spectral distribution, since due to scattering the sun appears yellowish, when it is in reality white. I did not have access to the necessary filters to combat this issue, although the increased amount of blue from the cloudy-gray sky would have produced a color closer to the sun's actual color by pure coincidence.

# 3 Calibration & Testing

The calibration process is straightforward in principle: shine light with known wavelengths at my spectrometer, and adjust the x-axis accordingly. In practice, known wavelengths were measured using the emission spectra of the hydrogen/helium tubes in the lab. In software, the location of the calibration bands in the image can be recorded and used to project from pixel space to unit space by sampling along the line between bands. After calibration, the device can be pointed at any light source to analyze its spectra.

The y-axis is left relative, as it would be difficult to get a precise calibration (especially with a device that needs to be frequently recalibrated); however, I do scale my values to be on the correct order of magnitude. I used a 2700k lightbulb to produce a reference curve which I scaled my values toward, although this is not necessarily needed with some trial and error and good judgment.

An issue I discovered in this process was that my camera filters infrared light, which greatly affects the overall spectral density distribution. I discuss the consequences of this in more detail in section 4.1. I was able to deal with this by clipping my data after the red peak, although this has drawbacks. Put simply, I am limited most by a combination of the measurable radiant intensities and visible wavelengths.

Figure 1: Calibration

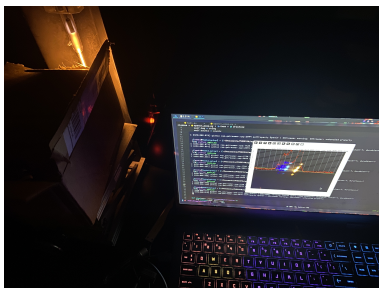
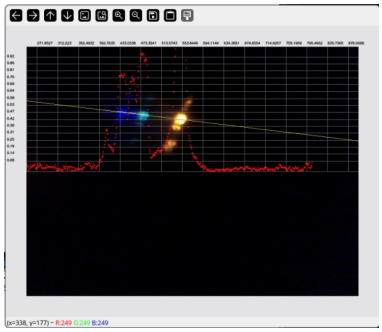


Figure 2: ui (during calibration)



# 4 Results & Discussion

My spectrometer has some major limitations for the purpose I designed it for, mostly which arise from the quality of my components (particularly the camera).

The device has a theoretical wavelength resolution of 1.827 nm, although this is probably an underestimate, since there is visible "smearing" of the bands as a result of my diffraction grating.

While the spectral intensities are relative, and therefore unitless, there are only 255 values possible for the intensities. This could be allevaited by remapping my image, a process which is common in film; however, this was not done because the camera itself has some limitation as to what intensity of light can be measured, which I do not know and lack the ability to determine in this time frame.

For these two reasons, I have chosen to exclude uncertainty from my calculations since there are so many unmeasurable yet significant factors which contribute to the overall instrumental uncertainty. Statistical uncertainty is astronomically smaller than my instrumental, and there are also hundreds of data points for each measurement. Taking multiple measurements would not greatly reduce the uncertainty because of the insignificance of noise relative to the other fundamental issues with the device.

### 4.1 2700k Lightbulb

| Actual Temperature (k) | Predicted Temperature (k) | Percent Difference |
|---|---|---|
| 2700 | 2850 | 5.4% |

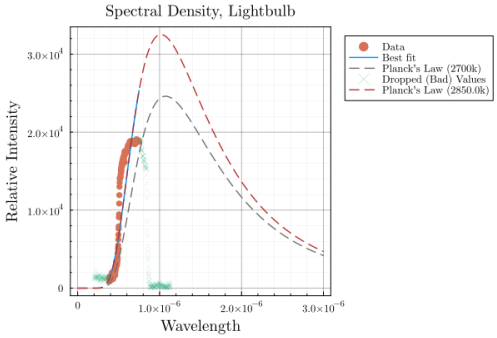
I first tested my spectrometer against an LED lightbulb which claimed to have a color temperature of 2700k. Technically, this is not an ideal thing to test against, since unlike an actual blackbody, little infrared light is emitted; although my sensor cannot measure those frequencies to begin with, so this simply serves to compare results between something at a human scale. Two things were made apparent with this test:

(1) The limited visible intensities of my camera (not color space range) are very significant, since if the light is shown directly through the slit, it clips off completely and the distribution is lost (this is not shown in my plot, however it was evident while gathering data).

(2) The lack of an infrared spectrum causes the distribution to be cut off very early, making it difficult to get an accurate fit as the peak is not included.

I believe that if these were not issues, there would be a more gradual decrease near the peak intensity. This is because from the unclipped distribution, it is clear that the peak should lie in the infrared spectrum, but because this is missing from my data, the distribution falls off from the prediction (Planck's Law) around 750nm. A reasonable approximation can still be made, however, as the temperature predicted by my fit is 2835k.

Figure 3: 2700k Lightbulb

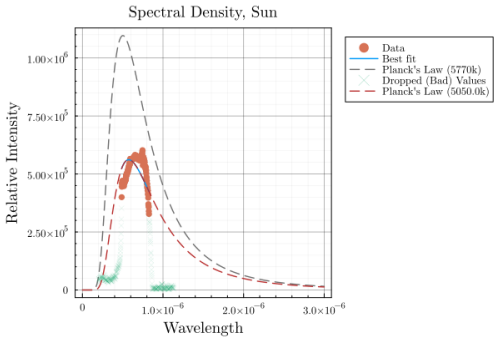



### 4.2 Sun

| Actual Temperature (k) | Predicted Temperature (k) | Percent Difference |
|---|---|---|
| 5770 | 5047.21 | 13.367% |

As discussed in section 2.4, the observed temperature of the sun due to Earth's atmosphere is approximately 5770 degrees k. My estimate was 5050 degrees k, which can in large part be attributed to the unfortunate weather conditions during data collection. Thankfully, my peak lies closer to the visible spectrum, so if I could collect more data, I believe I could get a more accurate measurement.

Figure 4: Sun - (Clipped)




## 5 Conclusion

My spectrometer demonstrated its ability to measure the spectral density of the visible bandwidth with reasonable accuracy. Actual spectrometers are calibrated in light intensity, however I did not have the means to do this, so it was left relative. This worked fine for my purposes; however, my distribution fits would have been much more accurate with correct units. My results lack uncertainty due to the nature of its complexity, and I was able to produce relatively good fits.

If I could repeat this, I would use a diffraction grating which is in better condition, and seek out other cameras as sensors, particularly ones for which I can remove the infrared filter.

# 6 Appendix

## 6.1 Julia Code

```
using IMinuit, Plots, CSV, DataFrames, LaTeXStrings, PlotThemes

theme(:dao)

h = 6.626e-34
kb = 1.381e-23
c = 3e8

yaxis(p) = @. ((502.5-577) / (289-40 - 363-40)) * (p-363-40) + 577
dist1(x, T) = @. (8*π*h*c)/(x^5) * (exp( (h*c)/(x*kb*T) ) - 1)^(-1)
dist(x, T, b) = @. (8*π*h*c)/(x^5) * (exp( (h*c)/(x*kb*T) ) - 1)^(-1) + 0*b
dist(x, p) = dist(x, p...)

df = DataFrame(CSV.File("dump-sky.csv", header=false))

y = Array(df[1, 160:350]) ./ 40e-5
x = Array(df[2, 160:350]) .* 10^(-9)
yerr_data = 0 .* x .+ 1e-5

data = Data(x, y, yerr_data)

m_planck = @model_fit dist data [50000, 0]
migrad(m_planck)
hesse(m_planck)
p = @plt_best(dist, m_planck, data)

m_planck_values, m_planck_errors = convert.(Vector{Float64}, (m_planck.values, m_planck.errors))

print(m_planck_values)
x2 = 0:1e-9:3e-6
y2 = dist1.(x2, m_planck_values[1])

x3 = 0:1e-9:3e-6
y3 = dist1.(x2, 5770)

yDropped = Array(df[1, Not(160:350)]) ./ 40e-5
xDropped = Array(df[2, Not(160:350)]) .* 10^-9

plot!(x3,y3, label="Planck's Law (5770k)", linestyle=:dash)
plot!(p)
plot!(xDropped, yDropped, seriestype=:scatter, label="Dropped (Bad) Values", ms=3, marker=:x, markeralpha=1)
plot!(x2,y2, title="Spectral Density, Sun", label="Planck's Law ($(round(m_planck_values[1], sigdigits=3))k)",
linestyle=:dash)

xlabel!("Wavelength")
ylabel!("Relative Intensity")

savefig("SunSpectralDensity.png")
```

## 6.2 Python Code

```
import cv2
import numpy as np
from classes.opencv_axes import Axes
from utils.helpers import extractValueChannel

cap = cv2.VideoCapture(2)
cap.set(cv2.CAP_PROP_EXPOSURE, 1)
cv2.namedWindow("UnnamedSpectrometer")

while True:
    ret, img = cap.read()
    if not ret:
        break

    borderSize = -40
    verticalOffset = -10
    # sampleStart = (borderSize + 80, borderSize + 252)
    # sampleEnd = (borderSize + 393, borderSize + 284)
    sampleStart = (borderSize, borderSize + 139 + verticalOffset)
    sampleEnd = (img.shape[1], borderSize + 220 + verticalOffset)

    #  masking and flattening the sample line into an array of intensity values
    vChannel = extractValueChannel(img)
    mask = np.zeros(img.shape[:2], dtype="uint8")
    cv2.line(mask, sampleStart, sampleEnd, 255, 1)
    masked = cv2.bitwise_and(vChannel, vChannel, mask=mask)
    slice = masked.flatten('F')  # flatten array (column major order)
    slice = slice[slice != 0]  # drop zeroes

    #  rendering flow
    #  post process a sample line onto the image
    img = cv2.bitwise_and(img, img, mask=cv2.bitwise_not(mask))
    cv2.line(img, np.add(sampleStart, (0,1)), np.add(sampleEnd, (0,1)), (28, 169, 147), 1)
    outImage = cv2.add(img, cv2.cvtColor(masked,cv2.COLOR_GRAY2RGB))

    #  plot values for intensity
    for x, value in enumerate(slice):
        cv2.circle(outImage, (x, 240 - value), 1, (0, 0, 255), -1)

    # throw the axes on there
    a = Axes(outImage)
    a.drawAxes()
    a.show()


    key = cv2.waitKey(1)
    if key == ord('q'):
        break
    elif key == ord('d'):
        for val in slice:
            print(f"{val},",end="")
        print()
        for wavelength in a.getAxis(len(slice)):
            print(f"{wavelength},",end="")

cap.release()
cv2.destroyAllWindows()
```

```python
import cv2
import numpy as np
from utils.helpers import projectArray2Line

class Axes:
    def __init__(self, cvImg, countx=16, county=16):
        # if settings need to be set do that here.
        self.img = cvImg
        self.countx = countx
        self.county = county

    def getAxis(self, size):
        height, width, _ = self.img.shape
        line_start_x_array = np.linspace(0, width, num=size)
        # line_start_y_array = np.linspace(0, height//2, num=count) # weird division done to put grid above sample
line
        axisValues = projectArray2Line( line_start_x_array, 338-40, 588, 194-40, 402 )
        return axisValues

    def drawGrid(self, color=(228, 226, 227), thickness=1, alpha=0.4):
        height, width, _ = self.img.shape
        line_start_x_array = np.linspace(20, width-20, num=self.countx)
        line_start_y_array = np.linspace(20, height//2 - 20, num=self.county) # weird division done to put grid
above sample line
        overlay = self.img.copy()

        cv2.rectangle(overlay, (0,0), (width, height // 2), color, thickness)

        for _, (x, y) in enumerate(zip(line_start_x_array, line_start_y_array)):
            # print(x)
            cv2.line(overlay, (int(x), 0), (int(x), height//2), color, thickness)  # vertical grid lines
            cv2.line(overlay, (0, int(y)), (width, int(y)), color, thickness)  # horizontal grid lines

        self.img = cv2.addWeighted(overlay, alpha, self.img, 1-alpha, 0)

    # I should generalize this sometime, but for now its application-specific
    def drawAxes(self, size=0.2, font_color=(59,56,57), font_thickness=1):
        height, width, _ = self.img.shape
        self.drawGrid()
        self.img = cv2.copyMakeBorder(self.img, 40, 40, 40, 40, cv2.BORDER_CONSTANT, value=(228, 226, 227))

        line_start_x_array = np.linspace(20, width-20, num=self.countx)
        line_start_y_array = np.linspace(20, height//2 - 20, num=self.county) # weird division done to put grid
above sample line
        # print(line_start_x_array)

        # wavelength (nm), peak 397 @ pixel 10, 410 @ pixel 20
        wavelength_labels = projectArray2Line( line_start_x_array, 338-40, 588, 194-40, 402 )
        intensity_labels = [round(2*val/height, 2) for val in line_start_y_array]
        intensity_labels = np.flip(intensity_labels)

        for i, (x, y) in enumerate(zip(line_start_x_array, line_start_y_array)):
           cv2.putText( self.img, str(round(wavelength_labels[i],4)), (int(x)+40, 35), cv2.FONT_HERSHEY_SIMPLEX,
size, font_color, font_thickness, cv2.LINE_AA)  # x-axis labels
           cv2.putText( self.img, str(round(intensity_labels[i],4)), (2, int(y)+40), cv2.FONT_HERSHEY_SIMPLEX, size,
font_color, font_thickness, cv2.LINE_AA)  # y-axis labels

    def show(self, name="UnnamedSpectrometer"):
        cv2.imshow(name, self.img)
```

```
import cv2
import numpy as np

def extractValueChannel(img):
    return cv2.cvtColor(img, cv2.COLOR_BGR2HSV)[:, :, 2]

def projectArray2Line(arr, x1, y1, x2, y2):
    m = (y2 - y1) / (x2 - x1)
    return [m*(x-x1)+y1 for x in arr]
```