In [ ]:

```
############### RDS Library methods #########################
#    RDS DIGITAL TELESCOPE SOFTWARE LIBRARY SECTION
#    THIS SECTION CONTAINS LIBRARY METHOD WRIITEN
#    FOR USE BY THE SCS STUDENTS AS PART OF THE RDS PROJECT
############################################################

#
#    073122 CN v1 Initial release



from IPython.display import HTML
from IPython.display import display



import sys
import math
import time
import picamera
import PIL
from fractions import Fraction
from collections import OrderedDict
from PIL import Image, ImageDraw, ImageFile, ImageFont, ImageOps
from glob import glob
import os, sys
import time
from datetime import datetime
from IPython.core.display import Image, display

# scan a column to determine top and bottom of area of lightness
def get_spectrum_y_bound(pix, x, middle_y, pic_height, spectrum_threshold, spectrum_threshold_duration, adj_top, adj_bot):
    c = 0
    spectrum_top = middle_y
    for y in range(middle_y, 0, -1):
        r, g, b = pix[x, y]
        brightness = r + g + b
        if brightness < spectrum_threshold:
            c = c + 1
            if c > spectrum_threshold_duration:
                break
        else:
            spectrum_top = y
            c = 0

    c = 0
    spectrum_bottom = middle_y
    for y in range(middle_y, pic_height, 1):
        r, g, b = pix[x, y]
        brightness = r + g + b
        if brightness < spectrum_threshold:
            c = c + 1
            if c > spectrum_threshold_duration:
                break
        else:
                spectrum_bottom = y
                c = 0
    print("spectrum_top is %d spectrum bottom is %d" % (spectrum_top, spectrum_bottom))
    spec_adj_top = spectrum_top + adj_top
    spec_adj_bot = spectrum_bottom + adj_bot
    print("adj spectrum_top is %d adj spectrum bottom is %d" % (spec_adj_top, spec_adj_bot))

    #narrow height by cutting off bottom of box
    return spec_adj_top, spec_adj_bot


# find aperture on right hand side of image along middle line
def find_aperture(pic_pixels, pic_width, pic_height, adj_top, adj_bot):
    middle_x = int(pic_width / 2)
#    middle_y = int(pic_height / 2)
    middle_y = int(pic_height * 3/5)
    aperture_brightest = 0
    aperture_x = 0
    for x in range(middle_x, pic_width, 1):
        r, g, b = pic_pixels[x, middle_y]
        brightness = r + g + b
        if brightness > aperture_brightest:
            aperture_brightest = brightness
            aperture_x = x
    print("aperture_x b4 avg is:",aperture_x)

    aperture_threshold = aperture_brightest * 0.9
    aperture_x1 = aperture_x
    for x in range(aperture_x, middle_x, -1):
        r, g, b = pic_pixels[x, middle_y]
        brightness = r + g + b
        if brightness < aperture_threshold:
            aperture_x1 = x
            break
    print("aperture_x1 is:",aperture_x1)

    aperture_x2 = aperture_x
    for x in range(aperture_x, pic_width, 1):
        r, g, b = pic_pixels[x, middle_y]
        brightness = r + g + b
        if brightness < aperture_threshold:
            aperture_x2 = x
            break
    print("aperture_x2 is:",aperture_x2)

    aperture_x = (aperture_x1 + aperture_x2) / 2
    print("avg aperture_x is:",aperture_x)

    spectrum_threshold_duration = 64
    aperture_y_bounds = get_spectrum_y_bound(pic_pixels, aperture_x, middle_y, pic_height, aperture_threshold, spectrum_threshold_duration, adj_top, adj_bot)
    aperture_y = (aperture_y_bounds[0] + aperture_y_bounds[1]) / 2
    aperture_height = (aperture_y_bounds[1] - aperture_y_bounds[0]) * 1.0
    return {'x': aperture_x, 'y': aperture_y, 'h': aperture_height, 'b':
aperture_brightest}


# draw aperture onto image
def draw_aperture(aperture, draw):
    fill_color = "#000"
    draw.line((aperture['x'], aperture['y'] - aperture['h'] / 2, apertur
e['x'], aperture['y'] + aperture['h'] / 2),
              fill=fill_color)


# draw scan line
def draw_scan_line(aperture, draw, spectrum_angle):
    fill_color = "#888"
    xd = aperture['x']
    h = aperture['h'] / 2
    y0 = math.tan(spectrum_angle) * xd + aperture['y']
    draw.line((0, y0 - h, aperture['x'], aperture['y'] - h), fill=fill_c
olor)
    draw.line((0, y0 + h, aperture['x'], aperture['y'] + h), fill=fill_c
olor)


# return an RGB visual representation of wavelength for chart
# Based on: http://www.efg2.com/Lab/ScienceAndEngineering/Spectra.htm
# The foregoing is based on: http://www.midnightkite.com/color.html
# thresholds = [ 380, 440, 500, 520, 565, 590, 625 ]
#                 vio  blu  cyn  gre  yel  org  red
def wavelength_to_color(lambda2):
    factor = 0.0
    color = [0, 0, 0]
    thresholds = [380, 450, 495, 570, 590, 620, 750]

    for i in range(0, len(thresholds) - 1, 1):
        t1 = thresholds[i]
        t2 = thresholds[i + 1]
        if lambda2 < t1 or lambda2 >= t2:
            continue
        if i % 2 != 0:
            tmp = t1
            t1 = t2
            t2 = tmp
        if i < 5:
            color[i % 3] = (lambda2 - t2) / (t1 - t2)
        color[2 - int(i / 2)] = 1.0
        factor = 1.0
        break

    # Let the intensity fall off near the vision limits
    if 380 <= lambda2 < 420:
        factor = 0.2 + 0.8 * (lambda2 - 380) / (420 - 380)
    elif 700 <= lambda2 < 1000:
        factor = 0.2 + 0.8 * (750 - lambda2) / (1000 - 700)

    return int(255 * color[0] * factor), int(255 * color[1] * factor), i
```

```
nt(255 * color[2] * factor)

def wavelength_to_rgb(wavelength, gamma=0.8):

    '''This converts a given wavelength of light to an
    approximate RGB color value. The wavelength must be given
    in nanometers in the range from 380 nm through 750 nm
    (789 THz through 400 THz).

    Based on code by Dan Bruton
    http://www.physics.sfasu.edu/astro/color/spectra.html
    https://sciencestruck.com/color-spectrum-chart
    '''


    wavelength = float(wavelength)
    if wavelength >= 380. and wavelength <= 450.:
        R = -1.*(wavelength - 450.) / (450. - 380.)
        G = 0.0
        B = 1.0

    elif wavelength >= 450. and wavelength <= 495.:
        R = 0.0
        G = (wavelength - 450.) / (495. - 450.)
        B = 1.0

    elif wavelength >= 495. and wavelength <= 570.:
        R = 0.0
        G = 1.0
        B = -1.*(wavelength - 570.) / (570. - 495.)

    elif wavelength >= 570 and wavelength <= 590:
        R = (wavelength - 570.) / (590. - 570.)
        G = 1.0
        B = 0.0

    elif wavelength >= 590. and wavelength <= 620.:
        R = 1.0
        G = -1.*(wavelength - 620.) / (620. - 590.)
        B = 0.0

    elif wavelength >= 620. and wavelength <= 750.:
        R = 1.0
        G = 0.0
        B = 0.0

    else:
        R = 0.0
        G = 0.0
        B = 0.0

    if wavelength > 700.:
        attenuation=0.3 + 0.7 * (750.-wavelength) / (750.-700.)
    elif wavelength < 420:
        attenuation=0.3 + 0.7 * (wavelength-380.) / (420.-380.)
    else:
        attenuation=1.0
    #R *= attenuation
    #G *= attenuation
    #B *= attenuation

    #R = R**gamma
    #G = G**gamma
    #B = B**gamma

    R *= 255
    G *= 255
    B *= 255

    return (int(R), int(G), int(B))

def take_spectrum(name, shutter):
    try:
        os.remove("capture_lockfile")
    except OSError:
            pass
    camera = picamera.PiCamera()
    try:
        print("allowing camera to warmup")
        camera.vflip = True
        camera.hflip = True
        camera.framerate = Fraction(1, 2)
        camera.shutter_speed = shutter
        camera.iso = 100
        camera.exposure_mode = 'off'
        camera.awb_mode = 'off'
        camera.awb_gains = (1, 1)
        time.sleep(3)
        print("capturing image")
        camera.capture(name, resize=(1296, 972))
    finally:
        camera.close()
        print("closing camera")
        os.mknod("capture_lockfile")
    return name

def determine_shutter(name, guess_shutter, ntrials):
    print("trying with increasing shutter speeds")
    shutter_speeds = []

    for trial in range(1,ntrials+1):
        shutter_speeds.append(guess_shutter * trial)

    for trial in range(1,ntrials+1):
        try:
            os.remove("capture_lockfile")
        except OSError:
                pass
        camera = picamera.PiCamera()
        try:
            print("allowing camera to warmup")
            camera.vflip = True
            camera.hflip = True
            camera.framerate = Fraction(1, 2)
            camera.shutter_speed = shutter_speeds[trial-1]
            camera.iso = 100
            camera.exposure_mode = 'off'
            camera.awb_mode = 'off'
            camera.awb_gains = (1, 1)
            time.sleep(3)
            print("capturing image at shutter speed %d" % shutter_speeds[trial-1])
            camera.capture(name + "_" + str(trial) + ".jpg", resize=(1296, 972))
        finally:
            camera.close()
            print("closing camera")
            os.mknod("capture_lockfile")

    return shutter_speeds

def getSize(filename):
    if os.path.isfile(filename):
        st = os.stat(filename)
        return st.st_size
    else:
        return -1

def wait_capture(file_path):
    time_to_wait = 10
    time_counter = 0
    while not os.path.exists(file_path):
        time.sleep(1)
        time_counter += 1
        if time_counter > time_to_wait:
            break

def draw_graph(draw, pic_pixels, aperture, spectrum_angle, wavelength_factor):
    aperture_height = aperture['h'] / 2
    step = 1
    last_graph_y = 0
    max_result = 0
    results = OrderedDict()
    for x in range(0, int(aperture['x'] * 7 / 8), step):
        wavelength = (aperture['x'] - x) * wavelength_factor
        if 1000 < wavelength or wavelength < 380:
            continue

        # general efficiency curve of 1000/mm grating
        eff = (800 - (wavelength - 250)) / 800
        if eff < 0.3:
            eff = 0.3

        # notch near yellow maybe caused by camera sensitivity
        mid = 571
        width = 14
        if (mid - width) < wavelength < (mid + width):
            d = (width - abs(wavelength - mid)) / width
            eff = eff * (1 - d * 0.12)
        # up notch near 590
        #mid = 588
        #width = 10
        #if (mid - width) < wavelength < (mid + width):
        #    d = (width - abs(wavelength - mid)) / width
        #    eff = eff * (1 + d * 0.1)

        y0 = math.tan(spectrum_angle) * (aperture['x'] - x) + aperture['y']
        amplitude = 0
        ac = 0.0
        for y in range(int(y0 - aperture_height), int(y0 + aperture_height), 1):
            r, g, b = pic_pixels[x, y]
            q = r + b + g * 2
            if y < (y0 - aperture_height + 2) or y > (y0 + aperture_height - 3):
                q = q * 0.5
            amplitude = amplitude + q
            ac = ac + 1.0

        amplitude = amplitude / ac / eff
        # amplitude=1/eff
        results[str(wavelength)] = amplitude
        if amplitude > max_result:
            max_result = amplitude
        graph_y = amplitude / 50 * aperture_height
        draw.line((x - step, y0 + aperture_height - last_graph_y, x, y0 + aperture_height - graph_y), fill="#fff")
        last_graph_y = graph_y
    draw_ticks_and_frequencies(draw, aperture, spectrum_angle, wavelength_factor)
    return results, max_result


def draw_ticks_and_frequencies(draw, aperture, spectrum_angle, wavelength_factor):
    aperture_height = aperture['h'] / 2
    for wl in range(400, 1001, 50):
        x = aperture['x'] - (wl / wavelength_factor)
        y0 = math.tan(spectrum_angle) * (aperture['x'] - x) + aperture['y']
        draw.line((x, y0 + aperture_height + 5, x, y0 + aperture_height - 5), fill="#fff")
        font = ImageFont.truetype('/usr/share/fonts/truetype/lato/Lato-Regular.ttf', 12)
        draw.text((x, y0 + aperture_height + 15), str(wl), font=font, fill="#fff")


def inform_user_of_exposure(max_result):
    exposure = max_result / (255 + 255 + 255)
    print("ideal exposure between 0.15 and 0.30")
    print("exposure=", exposure)
    if exposure < 0.15:
        print("consider increasing shutter time\n")
    elif exposure > 0.3:
        print("consider reducing shutter time\n")


def save_image_with_overlay(im, name):
    output_filename = name
    ImageFile.MAXBLOCK = 2 ** 20
    im.save(output_filename, "JPEG", quality=80, optimize=True, progress
ive=True)


def normalize_results(results, max_result):
    for wavelength in results:
        results[wavelength] = results[wavelength] / max_result
    return results


def export_csv(name, normalized_results):
    csv_filename = name + ".csv"
    csv = open(csv_filename, 'w')
    csv.write("wavelength,amplitude,red,green,blue\n")
    for wavelength in normalized_results:
        R,G,B = wavelength_to_rgb(wavelength)
        csv.write(wavelength)
        csv.write(",")
        csv.write("{:0.3f}".format(normalized_results[wavelength]))
        csv.write(",")
        csv.write(str(R))
        csv.write(",")
        csv.write(str(G))
        csv.write(",")
        csv.write(str(B))
        csv.write("\n")
    csv.close()


def export_diagram(name, normalized_results):
    antialias = 4
    w = 600 * antialias
    h2 = 300 * antialias

    h = h2 - 20 * antialias
    sd = PIL.Image.new('RGB', (w, h2), (255, 255, 255))
    draw = PIL.ImageDraw.Draw(sd)

    w1 = 380.0
    w2 = 750.0
    f1 = 1.0 / w1
    f2 = 1.0 / w2
    for x in range(0, w, 1):
        # Iterate across frequencies, not wavelengths
        lambda2 = 1.0 / (f1 - (float(x) / float(w) * (f1 - f2)))
#         c = wavelength_to_color(lambda2)
        c = wavelength_to_rgb(lambda2)
        #print(x,c)
        draw.line((x, 0, x, h), fill=c)
    pl = [(w, 0), (w, h)]
    for wavelength in normalized_results:
        wl = float(wavelength)
        x = int((wl - w1) / (w2 - w1) * w)
        # print wavelength,x
        pl.append((int(x), int((1 - normalized_results[wavelength]) * h
)))
    pl.append((0, h))
    pl.append((0, 0))
    draw.polygon(pl, fill="#FFF")
    draw.polygon(pl)

    font = PIL.ImageFont.truetype('/usr/share/fonts/truetype/lato/Lato-R
egular.ttf', 12 * antialias)
    draw.line((0, h, w, h), fill="#000", width=antialias)

    # Drawing 3 pixel high ticks on the x axis every 10 nm
    for wl in range(380, 750+1, 10):
        x = int((float(wl) - w1) / (w2 - w1) * w)
        draw.line((x, h, x, h + 3 * antialias), fill="#000", width=antia
lias)

    # Drawing 5 pixel high ticks on the x axis every 50 nm and values be
low
    for wl in range(380, 750+1, 50):
        x = int((float(wl) - w1) / (w2 - w1) * w)
        draw.line((x, h, x, h + 5 * antialias), fill="#000", width=antia
lias)
        wls = str(wl)
        tx = draw.textsize(wls, font=font)
        draw.text((x - tx[0] / 2, h + 5 * antialias), wls, font=font, fi
ll="#000")

    # save chart
    sd = sd.resize((int(w / antialias), int(h / antialias)), PIL.Image.A
NTIALIAS)
    output_filename = name
    sd.save(output_filename, "PNG", quality=95, optimize=True, progressi
ve=True)

def export_inv_diagram(name, invname):
    # invert and save chart
    sd = PIL.Image.open(name)
    sd_mirror = PIL.ImageOps.mirror(sd)
    output_filename = invname
    sd_mirror.save(output_filename, "PNG", quality=95, optimize=True, pr
ogressive=True)

def display_image_in_actual_size(im_path):
    import matplotlib as mpl

    dpi = mpl.rcParams['figure.dpi']
    im_data = pyplt.imread(im_path)
    height, width, depth = im_data.shape

    # What size does the figure need to be in inches to fit the image?
    figsize = width / float(dpi), height / float(dpi)
    # Create a figure of the right size with one axes that takes up the full figure
    fig = pyplt.figure(figsize=figsize)
    ax = fig.add_axes([0, 0, 1, 1])

    # Hide spines, ticks, etc.
    ax.axis('off')

    # Display the image.
    ax.imshow(im_data, cmap='gray')

    pyplt.show()
```

In [ ]:

```
import pandas as pd
import numpy as np
import peakutils
from peakutils.plot import plot as pplot
from scipy.interpolate import make_interp_spline
from matplotlib import pyplot
%matplotlib inline

def display_bds_params(name,desc,shutter,slit_topadj,slit_botadj,spectrum_angle,wavelength_factor,samp_th,wlen_th):
    print("Title:\t\t", desc.upper())
    print("BDS parameters used for this run:")
    print("Spectrum Base Name is          \t", name)
    print("Camera Shutter is:             \t", shutter)
    print("Slit Top Adjustment is:        \t", slit_topadj)
    print("Slit Bottom Adjustment is:     \t", slit_botadj)
    print("Camera Spectrum Angle is:      \t", spectrum_angle)
    print("Camera Wavelength Factor is:   \t", wavelength_factor)
    print("Amplitude Threshold is:        \t", samp_th)
    print("Wavelength Threshold is:       \t", wlen_th)

def write_bds_params(fnametxt,name,desc,shutter,slit_topadj,slit_botadj,spectrum_angle,wavelength_factor,samp_th,wlen_th):
    fname = open(fnametxt, 'w')
    print("Title:\t\t", desc.upper(), file=fname)
    print("BDS parameters used for this run:", name, file=fname)
    print("Spectrum Base Name is          \t", name, file=fname)
    print("Camera Shutter is:             \t", shutter, file=fname)
    print("Slit Top Adjustment is:        \t", slit_topadj, file=fname)
    print("Slit Bottom Adjustment is:     \t", slit_botadj, file=fname)
    print("Camera Spectrum Angle is:      \t", spectrum_angle, file=fname)
    print("Camera Wavelength Factor is:   \t", wavelength_factor, file=fname)
    print("Amplitude Threshold is:        \t", samp_th, file=fname)
    print("Wavelength Threshold is:       \t", wlen_th, file=fname)

def get_plot_titles(element, pwl):
    t1="THE MEASURED PEAK WAVELENGTHS FOR {} IN NANO_METERS ARE: {}".format(element.upper(), pwl)
    if   element.lower() == 'argon':
        t2="THE NIST STANDARD STRONG LINE WAVELENGTHS FOR {} ARE: {}".format(element.upper(),'750 763 794 810')
    elif element.lower() == 'helium':
        t2="THE NIST STANDARD STRONG LINE WAVELENGTHS FOR {} ARE: {}".format(element.upper(),'587 668 706 728')
    elif element.lower() == 'hydrogen':
        t2="THE NIST STANDARD STRONG LINE WAVELENGTHS FOR {} ARE: {}".format(element.upper(),'486 656')
    elif element.lower() == 'krypton':
        t2="THE NIST STANDARD STRONG LINE WAVELENGTHS FOR {} ARE: {}".format(element.upper(),'811 828 850 877')
    elif element.lower() == 'mercury':
        t2="THE NIST STANDARD STRONG LINE WAVELENGTHS FOR {} ARE: {}".format(element.upper(),'404 436 546')
    elif element.lower() == 'neon':
        t2="THE NIST STANDARD STRONG LINE WAVELENGTHS FOR {} ARE: {}".format(element.upper(),'585 607 615 626 640 837 865 878')
    elif element.lower() == 'nitrogen':
        t2="THE NIST STANDARD STRONG LINE WAVELENGTHS FOR {} ARE: {}".format(element.upper(),'745 868')
    elif element.lower() == 'terbium':
        t2="THE NIST STANDARD STRONG LINE WAVELENGTHS FOR {} ARE: {}".format(element.upper(),'432 535')
    elif element.lower() == 'fluorescent':
        t2="THE NIST STANDARD STRONG LINE WAVELENGTHS FOR {} ARE: {}".format(element.upper(),'404 436 535 546 611 631')
    elif element.lower() == 'cfls':
        t2="THE NIST STANDARD STRONG LINE WAVELENGTHS FOR {} ARE: {}".format(element.upper(),'HG 404 434 TB 542 586 EU 599 612 632')
    else:
        t2 = "THE NIST STANDARD STRONG LINE WAVELENGTHS DO NOT EXIST FOR THIS ELEMENT!"

    return t1, t2


def draw_spectral_line_peaks(element,name,fnamepng,desc,samp_th,wlen_th):
    sdf = pd.read_csv(name)
    fig = pyplot.figure(figsize=(10,6))
    x = sdf['wavelength']
    y = sdf['amplitude']
    #r = sdf['red']
    #g = sdf['green']
    #b = sdf['blue']
    peaks = peakutils.indexes(y, thres=samp_th, min_dist=wlen_th)
    peak_wavelength_list = ""
    for i in reversed(peaks):
        peak_wavelength_list += (str("%3.0f " % x[i]))
    t1, t2 = get_plot_titles(element, peak_wavelength_list)
    pyplot.title('SPECTRAL PEAK WAVELENGTHS FOR '+element.upper()+'\n'+t1+'\n'+t2)
    #pyplot.show()
    pplot(x, y, peaks)
    #fig.savefig(fnamepng,bbox_inches='tight')
    fig.savefig(fnamepng)

    return peak_wavelength_list, t1, t2


def draw_spectral_color_fill_chart(element,name,fnamepng,desc,samp_th,wlen_th,t1,t2):
    sdf = pd.read_csv(name)
    df2 = sdf[['wavelength', 'amplitude']].copy()
    fig = pyplot.figure(figsize=(10,6))

    #print(df2)

    r = sdf['red']
    g = sdf['green']
    b = sdf['blue']    # Plotting the Graph
    col_c=[]
    for i in range(0,len(r)):
        col_c.append('#%02X%02X%02X'%(r[i],g[i],b[i]))
    df3 = df2.assign(hexclr=col_c)

    #print(df3)
    x = df3['wavelength']
    y = df3['amplitude']
    c = df3['hexclr']
    pyplot.rcParams["figure.figsize"] = [10, 6]
    #pyplot.rcParams["figure.autolayout"] = True
    fig,ax=pyplot.subplots()

    #ax.plot(x, y, color=c, alpha=1)
    #ax.fill_between(x, y, color=c, alpha=.1)

    for i in range(len(x)):
        ax.scatter(x[i], y[i], color=c[i], alpha=1)
        #ax.fill_between(x[i], y[i], color=c[i], alpha=.1)

    #ax.plot(x, y, color='#%02X%02X%02X'%(r,g,b), alpha=1)
    #ax.fill_between(x, y, color='#%02X%02X%02X'%(r,g,b), alpha=.1)
    pyplot.title('SPECTRAL COLOR CHART FOR '+element.upper()+'\n'+t1+'\n'+t2)
    pyplot.xlabel("wavelength (color)")
    pyplot.ylabel("amplitude")
    pyplot.show()
    #fig.savefig(fnamepng,bbox_inches='tight')
    fig.savefig(fnamepng)
```