Building Circadian Effective Spectra: A C Language Toolkit

Readme

Jonathon David White
Version: November 19, 2024

0. Table of Contents

1. General Information

The program (whitespd.c) builds and calculates optical parameters and optimizes SPDs for LED strip lighting to produce energy-efficient white lighting. Additional routines allow a 24-hr dynamic lighting system to be simulated.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see https://www.gnu.org/licenses/gpl-3.0.en.html

If you choose to make use of this program or subroutines in your work, we would appreciate it if you would cite our paper "Building Circadian Effective Spectra: A C Language Toolkit", Leukos QQQ (2024), by Md Azaharuddin Ansari and Jonathon David White (doi: 10.1080/15502724.2024.2423721) in which we discuss the logic and the usage of this program in designing and building dynamic white lighting systems.

You may download the latest version of the code from https://software.whiteslight.com/

2. Preparation (MS-Windows)

2.1 Download the Software

a. Download and unzip whitespd() in the C directory
b. If not already on your computer, download and unzip MinGW in the same directory. This is used to compile the C language code.
c. If not already on your computer, download and unzip gnuplot() in the same directory. This is used to graph the files generated by whitespd().
d. Check that your directory structure looks like:

- `C:/whitespd`
- `C:/MinGW`
- `C:/gnuplot`

2.2 Set Up the Environment

a. Run "`0setup.bat`" in the directory whitespd. This will create the required directory structure, copy other files into whitespd() and provide a command line prompt.
b. Run "`1setpath.bat`" from the command prompt. This will add gnuplot and MinGW port of gcc to path directory (it will look for gnuplot and MinGW in the C directory or the same base directory as whitespd()

3. Running Demonstration to Create Graphs in Paper (MS-Windows)

Run "`2demo.bat`" to run the program and create all the plots shown in the paper "Building Circadian Effective Spectra: A C Language Toolkit". It will:

a. Prepare required input files (cf Fig. 1 in the paper[1])
   1. Copy the lighting configuration from the demo directory (`lightconfig.csv`)
   2. Copy Optimization Goals from the demo directory (`goal.csv`)
   3. Copy spectrum weighting files from the demo directory (`w_*.csv`). This provides the program information about which LED strips to include when building the SPD.
   4. Create SPD (radiant flux/nm) for the Red LED strip (SPD files are already in directory in)
      i. Call `procOcean()` to convert the ocean optics SPD file for Red and Warm White strips (counts/nm) into 2 column files
      ii. Call `calibrate()` to convert the measured Red LED file into radiant flux/nm, assuming that the Warm White strip spectrum also has a NIST calibrated spectrum (radiant flux/nm)
      iii. Call gnuplot() to plot the files for user verification

b. Call `whitespd()` in read mode to calculate the optical parameters and then plot the input spectra (Fig. 2 in the paper)

c. Call `whitespd()` in read mode followed to build spectra suitable for morning, afternoon, evening, and night, and calculate the optical parameters and plot (Fig. 3 and Table 2 in the paper)

d. Call `dayinterp()` to interpolate between the SPD to develop a 24-hour lighting schedule.

e. Call `whitespd()` in read mode followed by gnuplot() to calculate and plot optical parameters as a function of time (Fig. 4,5, and 6 in the paper)

f. Call `day2esp()` to convert the schedule into a hard-coded schedule to be used to control lighting in the room shown in Fig. 7 of the paper)

4. Input Files

4.1 Weighting of LEDs

a. The first line is required and specifies

– number of unique LEDs to use in building SPDs

– path and file names (no extension) of all input SPDs

– e.g.: `2,in/ww19,in/cw19,`

b. subsequent lines (are used only when whitespd() is reading weights, ignored of optimization)

– output file name (no extension)

1 In this file, references to figures and “paper”, refers to "Building Circadian Effective Spectra: A C Language Toolkit", Leukos QQQ (2024), by Md Azaharuddin Ansari and Jonathon David White (doi: 10.1080/15502724.2024.2423721)

– fraction of full power for each input LED strip

– e.g.: fname,0.90,0.5

4.2 Lighting configuration (lightconfig.csv)

This file scales the predicted output illuminance. The first 2 lines are measured in the room where the system is deployed with any LED strip. The third line is the simulation output for the same configuration when the measured lx is set to 1 in the program. Here is an example of this input file:

```
0.626,in situ ratio of vertical/horizontal illuminance
991,measured horizontal illuminance at 1.2 m height [lx]
289113,simulated horizontal illuminance [lx]
a, measured by Ansari
measured on 231023 in YZU-r70723 (Taiwan)
```

4.3 Optimization Goals (goal.csv)

This file contains the optimization goals and the criteria employed to reject SPDs as being unsuitable. The first column specifies

– the maximum acceptable value for the parameter (10),

– the minimum acceptable value for the parameter (-10),

– the parameter is to be maximized (+1)

– the parameter is to be minimized (-1)

– the parameter should not be considered in the optimization routine (0)

The second column provides the minimum or maximum value of the parameter. The third column provides the parameter's name, which the program ignores. The following is a goal file used for maximizing m-EDI efficiency [lx/W] under the constraints that Duv<|0.006|, CCT<6500K and CRI-Ra>80 and that m-EDI >250 lx. These conditions are suitable for midday.

```
10,0.006,Duv,
10,6500,CCT,
-10,80,CRI-Ra,
0,0,CRI-Rx,
0,0,a-opic s-cone ELR
0,0,a-opic m-cone ELR
0,0,a-opic l-cone ELR
0,0,a-opic rhodopic ELR
0,0,a-opic melanopic ELR
0,0,photopic illuminance [lx],
-10,250,m-EDI [lx],
0,0,irradiation horizontal plane,
0,0,cost upfront,
```

```
0,0,photopic lx/W,
1,0,m-EDI lx/W,
```

The program requires that all parameters be specified in this file, even if they are to be ignored in the optimization procedure. For early evening lighting, one would want to maximize the photopic illuminance efficiency [lx/W] while maintaining a minimum level of photopic illuminance.

4.4 LED strip information (2 files)

Two files are required for each type of LED. The first file (`*.csv`) contains the SPD of the LED (nm, uW/nm) from 380nm to at least 780nm. If the correct units are not used (e.g., counts/nm rather than uW/nm), the simulation will give incorrect results. The format of the file is as follows where the first column is the wavelength [nm] and the second column the radiant flux [uW/nm]

```
380,1.27e-001
381,1.24e-001
382,3.03e-001
```

The second file (`*_d.csv`) provides the cost/m, W/m and strip information formatted as:

```
2.35, US$/m
18, W/m
bl19cw_60, supplier strip type and LED/m
```

NIST-calibrated spectra of SPD of many different strips (that we have measured) are included in the subdirectory `in`

4.5 Optical Constants

These files are included with the program and downloaded from the CIE website. It includes the files necessary for calculating the α-opic parameters, CRI, and CIE1931.

5. Program Flow

5.1 Preparing Input SPD files (Skip if using included SPDs or have in uW/nm).

a. Take experimental spectra of LEDs [380 780] nm
   1. Optimize the spectrometer time constant (tc) to use full count range
   2. Take all spectra under similar conditions

b. Convert the spectrometer output to 2 columns, comma-separated data, and scale by the time constant to get counts/s. (In 2demo.bat, `procOcean()` does this for files measured using an Ocean Optics spectrometer) These files should be stored with the extension `.txt`

c. If available, measure one spectrum using a NIST-calibrated integrating sphere (if available). Save the SPD (uW/nm) with the extension .csv
   1. Calibrate all the measured SPDs using the spectra obtained using the NIST calibrated sphere

where nameTST is the SPD to be calibrated (no extension), and fnameSTD is the name of the NIST calibrated spectra pair. For example:

```
calibrate fnameTST standard fnameSTD
```

d. If a NIST calibrated spectrumm is unavailable, convert the SPDs from counts/s to uW/nm. For example.

```
calibrate fnameTST counts2energy
```

5.2 Main Program whiteSPD()

a. Overview

The control program in this software in whitespd() The format of the program call is:

```
whiteSPD weight.csv mode flgSaveSPD
```

where the first parameter is a file that contains the names and weights of the LED strips (cf §3.1), the second parameter is the mode [read | brute | perm], and the final parameter flgSaveSPD is 0 (don't save the SPD) or 1 (write the SPD to a file).

In read mode, the weights of the LED strips are read from the file weight, parameters calculated, and the SPD plotted (if flgSaveSPD=1). In the optimization modes, only the first line of the weight file is used to specify the SPDs to be used as input.

In brute force mode, the weightings of each SPD are generated from 0.0 to 1.0 with an increment input by the user. This is exhaustive but slow. A step size of 0.05 can provide results for systems with up to 5 LEDs in a reasonable time.

Permutation mode is like brute force, except it only builds unique SPDs. For example, assume I have a two-LED system with Warm White and Cool White strips. The SPD of 1% Warm White and 1% Cool White is identical, minus a scale factor, with 100% Warm White and 100% Cool White. In this case, the program only builds the SPD with the highest radiant flux and evaluates this. With the current speed of a notebook computer, this is required when the increment is 0.01 (1%), and the system has 5 or more distinct input SPDs.

b. Program Structure

`main()` exists only to call the function `whitespd()`.

`whitespd()` first initializes the variables by calling `readOpticalConstants()` to read the optical constants, `readLightConfig()` to read the room lighting configuration and readGoal() to read the simulation goals. It then reads the weight file to determine the SPDs to include in building the spectra and calls `readSPD()` to read the csv files describing each strip (SPD and power consumption).

It then enters the main loop (cf. Fig. 1 in paper) and calls `getWeight()` is called to get the weights for the next SPD to be evaluated. `getWeight()` returns the weight for each SPDs based on the mode chosen on the command line. Based on these weights, `buildSPD()` is called to build the combined SPD. Next `calcSPDpar()` is called. This function is the core of the program. Its operation is detailed below:

1. `calcCIE1931()` is called to calculate the CIE coordinates
2. `calcCCT()` is called to calculate CCT
3. If CCT is outside the minimum or maximum limits specified in the file goal.csv, the SPD is rejected, and control is returned to `whitespd()`
4. calcDUV() and calcDUVtarget() are called to check the value of Duv tolerance.
5. If Duv tolerance exceeds the maximum limit, the SPD is rejected, and control is returned to `whitespd()`. If this is a parameter to maximize or minimize, points are given to the SPD's score.
6. `calcCRI()` is called calculate the CRI-Ra and Rx.
7. If CRI is lower than the minimum limit, the SPD is rejected, and control is returned to whitespd(). If this is a parameter to maximize, points are given to the SPD's score.
8. `calcAlphaOpicELR()` is called to calculate the ELR coefficients, illuminance and m-EDI illuminance in the horizontal direction
9. Photopic and m-EDI illuminance are scaled by the light configuration and the m-EDI illuminance is obtained in the vertical direction in accord with specifications
10. If any of the ELR coefficients or the photopic or m-EDI illuminance is outside of the limits, the SPD is rejected, and control is returned to `whitespd()`. If the goal is to optimize any of these parameters, points are given to the SPD's score.
11. After calculating the upfront cost [$] and the electricity costs [W], the efficiency of photopic and m-EDI are calculated.
12. If any of these are outside of the limits, the SPD is rejected, and control is returned to `whitespd` If the goal is to optimize any of these parameters, points are given to the SPD's score.
13. Control is returned to `whitespd()`.

Based on the mode, `whitespd()` either writes the SPD's optical parameters to a file (read) or checks if this spectrum best meets the optimization goals and if so saves it as the temporary best SPD. Control then loops back to process the next SPD's weights. Once all weightings have been processed, the optical parameters of the highest-scored SPD are written (brute or perm mode)

c. View graphs

Gnuplot() is used to view graphs. To view the SPDs, the control file is combine.plt. To view the calculated optical parameters as a function of the time of day, the input control file is spdpar_day.plt. Thus, we call either:

```
gnuplot combine.plt
gnuplot spdpar_day.plt
```

In the second case, the names of the files are the time of day in seconds.

5.3 Creating a 24-hr spectrum

Once the SPD has been optimized at critical times (e.g. morning, noon, early and late evening and night), the function `dayinterp()` can be used to interpolate between the points. It reads from the weight file (cf §3.1) the time of day in seconds (1st column) and the weightings of the input SPDs at that time. It then interpolates to build a 24-hour spectrum, which it writes in the file `dayinterp.csv`. The parameters for the 24-hour schedule can then be viewed by calling `writespd()` in read mode and then using `gnuplot()` to plot the changes in optical parameters throughout the day. For example,

```
dayinterp w_build.csv
whitespd dayinterp.csv read 0
gnuplot spdpar_day.plt
```

5.4 Cleaning up

After running the program, the batch file "`clean.bat`" can be used to delete all the files generated in the simulation

6. Function Tree

6.1 whiteSPD

main()
- └prnLog()
- └whiteSPD()
  - └readGoal()
    - └readGoalLine()
  - └readLightConfig()
  - └readOpticalConstants()
    - └readInterpSPD()
  - └writeSPDparHeader()
  - └readSPD()
  - └getWeight()
    - └readWeight()
    - └genWeightBruteForce()
    - └genPerm()
  - └buildSPD()
  - └writeSPD()
  - └calcSPDpar()
    - └calcCIE1931()
    - └calcCCT()
    - └calcUV()
    - └calcDUV()
    - └calcDUVtarget()
    - └calcCRI()
      - └calcCIE1931()
      - └calcUV()
      - └calcCD()
      - └calcRef_BlackBody()
      - └calcRef_StandardIlluminateD()
      - └calcUVkibar()
      - └calcUV_TCS()
      - └calcUVW()
    - └calcAlphaOpicELR()
    - └calcIrrad()
    - └calcCLA()
  - └writeSPDpar()
- └plotSPD()
  - └gnuplotSPDopen()
  - └gnuplotSPD()
  - └gnuplotSPDclose()

6.2 dayinterp

main()
- └dayinterp()
- └readWeight()
- └simuSchedule()
- └calcCS()

6.3 day2esp

main()
- └day2esp()
- └readWeight()
- └writeSch2Esp()

6.4 post

main()
- └readGoal()
- └goalMinMax()
- └readSPDpar()
- └flg_update()
- └writeSPDpar()

6.5 calibrate

main()
- └calibrate()
  - └readInterpSPD()
  - └readSPD()
- └writeSPD()
- └gnuplotCMD()

6.6 procOcean

main()