Arduino-Based UV Measurements

David R. Brooks, *Institute for Earth Science Research and Education*, © 2021

David Brooks' book, *Exploring Your Environment with Arduino Microcontrollers* (https://instesre.org/ArduinoBook/ArduinoBook.htm), includes a chapter on UV measurements, using the analog GUVA-S12SD sensor module and the now-discontinued I2C VEML6075 module. This document provides a more in-depth look at the GUVA-S12D and two other analog UV sensors.

Despite loss of the VEML6075, there are still several Arduino-compatible UV sensors available, such as the VEML6070 (which has a single UVA sensor and an I2C interface), costing less than $10. Some may argue that the convenience of an I2C interface, even though it requires downloading a software library from the sensor module vendor, is preferable because the user doesn't have to worry about interpreting and processing sensor output. But the straightforward nature of an analog device can be an advantage, assuming that the voltage output is properly interpreted – see https://instesre.org/ArduinoBook/ProcessingAnalogInputs.pdf.

Two of these sensors produce output for UV power, in units of mW/cm^2. But what many users may want to know is the ultraviolet index (UVI). At least one device described as a "UVI sensor" the SI1145, only *infers* a UVI from measurements of visible and IR light, without *any* direct measurement of UV radiation at all!

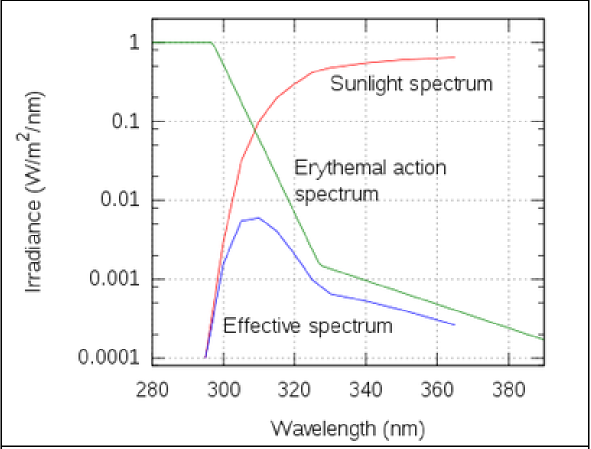


Figure 1. The McKinlay-Diffey erythemal action spectrum.

The ultraviolet index (UVI) is based on models of how Caucasian skin reacts to exposure to UV radiation from the sun. The most widely used model is the McKinlay-Diffey erythemal action spectrum, shown in Figure 1. The action spectrum represents the wavelength-dependent UV power from sunlight reaching Earth's surface under clear skies weighted by the response of average Caucasian skin (the "effective strength" at each wavelength). Integration of those effective strengths across wavelengths from 290 to 400 nm yields the total effect of UV radiation on Caucasian skin. That value is then divided by an arbitrary constant value to yield the UVI. (See https://www.epa.gov/sunsafety/calculating-uv-index-0.)

The daily UVI index forecast is calculated for clear sky conditions at solar noon – the time when the sun is highest in the sky – looking straight up from a horizontal surface. That forecast is then adjusted (downward) based on models and observations for site elevation, total ozone, aerosols, and cloud amounts and conditions. Hourly UVI forecasts can then be made based on the changing solar elevation angle during the day and changing sky conditions. "Official" UVI values are always reported as integers with values around 10 or less on clear summer days at temperate latitudes and low to moderate elevations. (See https://www.epa.gov/sites/default/files/documents/uviguide.pdf.)

You can find current noontime UVI forecasts here:
https://forecast.weather.gov/product.php?site=CRH&product=UVI&issuedby=CAC
The exposure risks associated with UVI values are characterized by the World Health Organization:

UVI	Exposure Level
0-2	low
3-5	moderate
6-7	high
8-10	very high
11 and greater	extreme

UV radiation is generally separated into three regions:

UVA: 315-400 nm (About 90% reaches Earth's surface from the sun.)
Sometimes UVA is subdivided into UVA-I (340-400 nm) and UVA-II (315-340 nm). Overexposure to UVA radiation is the leading cause of cataracts.

UVB: 280-315 nm (About 10% reaches Earth's surface from the sun.)
High and repeated exposure to UVB can produce sunburn and prolonged exposure can cause permanent eye damage and skin cancer.

UVC: 100-280 nm (None reaches Earth's surface from the sun.)
Exposure can cause severe damage to the human eye.

The analog-output GUVA-S12SD UV sensor

The GUVA-S12SD UV is a single-detector analog-output device from Korea-headquartered GenUV. See http://www.geni-uv.com/download/products/GUVA-S12SD.pdf.

Some pertinent details from the datasheet are shown in Figures 2 through 4. The device shown in Figure 2 is adafruit.com's version of a module using this sensor. The sensor itself is housed in the small white rectangle. For a size reference, the three connection points on the left (power, ground, and output) are on 0.1" centers.

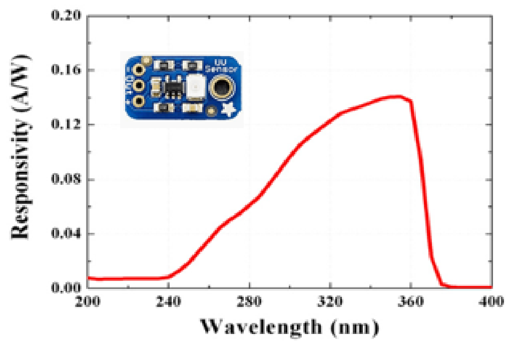
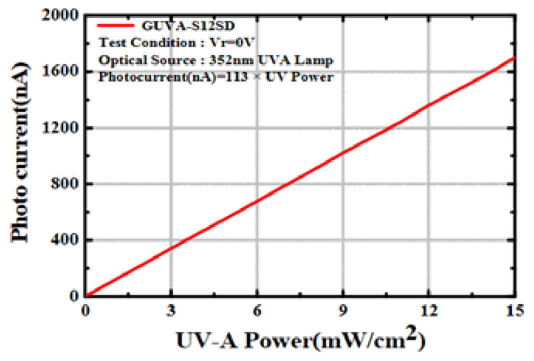


Figure 2. Wavelength response of GUVA-S12SD sensor.



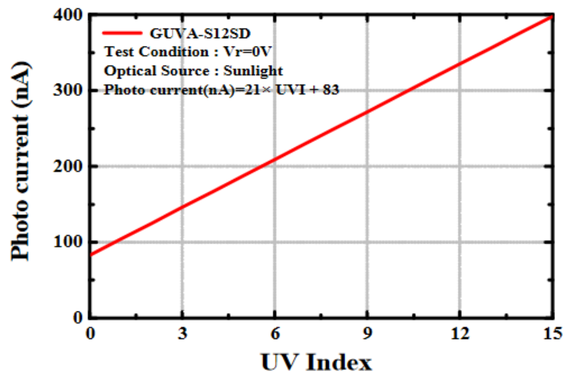


Figure 3. Photocurrent to UV power for GUVA-S12SD sensor.

Figure 4. Photocurrent to UVI for GUVA-S12SD sensor.

Like other similar sensors, the GUVA-S12SD produces a small photocurrent, as shown in Figure 3, when exposed to UV radiation. When the sensor is incorporated into an Arduino-compatible module, a transimpedance amplifier is used to convert the photocurrent to a usable output voltage. (See, for example, https://instesre.org/construction/TransimpedanceAmplifier/TransimpedanceAmplifier.htm.)

The conversion for the Adafruit version is given as:

(1) V_{out} (V) = 4.3•I (μA)

So, according to Figure 3, an output voltage is 4.3 V corresponds to a photocurrent of 1 μA, or 9 mW/cm^2.

The 4.3 value is not part of the GUVA-S12SD specification because the transimpedance amplifier is not part of that device. Rather, the value of 4.3 needs to be based on a calibration process involving an accurate voltmeter and low-current ammeter, and is also based on the assumption that the amplifier gain is the same for every module; giving this value with only two digits might indicate that 4.3 is only an "approximate" gain that might vary a little from sample to sample.

The actual output from the GUVA-S12SD should never be as high as 4.3 V under any natural sunlight condition. According to Figure 4, a photocurrent of 0.400 μA, produces an output voltage of 1.72 V, corresponding to an ultraviolet index of 15. Any UVI value of 11 or more is characterized as "extreme" for human exposure and is very unlikely to be exceeded anywhere except perhaps at very high elevation sites. (More about the UVI later in this document!) According to Figure 3,

(2) P (mW/cm^2) = photocurrent (nA)/113

The 113 value in equation (2) doesn't define a conversion from photocurrent to power that can be applied to other sensors; it refers only to the UV radiation measured by this particular sensor. Other sensors will have different spectral responses and even different samples of the *same* sensor may show some variability in their output.

Testing the GUVA-S12SD

Sketch 1 shows code for testing this sensor with an Arduino. The only significant difference between this code and code more commonly seen for interpreting analog voltage inputs is the use of an external reference voltage for the analog input pins to produce the most accurate A-to-D-to-A conversion regardless of how the board is powered; see https://instesre.org/ArduinoBook/ProcessingAnalogInputs.pdf.
Assuming a 5 V Arduino like an UNO or Nano, use the 3.3 V power pin for the GUVA-S12SD, Connect 3.3 V from this pin also to the AREF pin, the ground pin to GND, and the output pin to A0.

In this code, `N=10` values are averaged at intervals of `DT=50` ms. The delay time value is specified as a long integer, although this isn't necessary for delays less than about 30,000 ms.

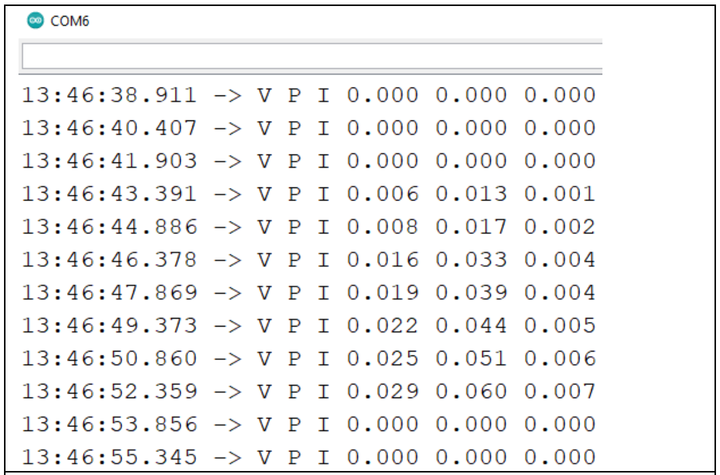
Sketch 1. Testing the GUVA-S12SD sensor.

```
/* GUVA_S12SD.ino D. Brooks August2021
  Test the GUVA-S12SD UV sensor using the 3.3V power pin
  as an external reference voltage for A-D-A conversions.
  Connect the 3.3 V power pin to the AREF pin.
  See http://www.geni-uv.com/download/products/GUVA-S12SD.pdf
```

```
*/
const float aRef=3.3; // Check this value on your board!
const int UVpin=A0,N=10,DT=50;
float I,P,V,UVI;
int i,sum; // be careful not to exceed allowed size for int - N=10 is OK.
long int DelayTime=1000L;
void setup() {
  Serial.begin(9600);
  analogReference(EXTERNAL); // Absolutely REQUIRED here!
  pinMode(UVpin,INPUT);
}
void loop() {
  sum=0.;
  for (i=1; i<=N; i++) {
    sum+=analogRead(UVpin);delay(DT);
  }
  V=(sum/(float)N)*aRef/1023.;
  I=V/4.3; // I in uA, see Adafruit.com documentation.
  P=I*1000./113.;
  //UVI=(I*1000.-83.)/21.;  // I in nA, see GUVA-S12SD datasheet
  Serial.print("V P I ");
  Serial.print(V,3); Serial.print(' ');
  Serial.print(P,3); Serial.print(' ');
  Serial.println(I,3);
  delay(DelayTime);
}
```

Figure 5 shows some output from Sketch 1 collected indoors away from a double-paned window. Under these conditions, there isn't any appreciable "natural" UV. For part of the time the sensor was held very close to an LED desk lamp, which apparently produced a very small amount of UV.

Figure 6 shows a breadboard layout using an Arduino Nano and a small IZOKEE OLED to display data, powered by a 9 V battery with a clip to connect it to the GND and VIN pins. This is a good system for intermittent use even with an inexpensive carbon battery as shown.



```
13:46:38.911 -> V P I 0.000 0.000 0.000
13:46:40.407 -> V P I 0.000 0.000 0.000
13:46:41.903 -> V P I 0.000 0.000 0.000
13:46:43.391 -> V P I 0.006 0.013 0.001
13:46:44.886 -> V P I 0.008 0.017 0.002
13:46:46.378 -> V P I 0.016 0.033 0.004
13:46:47.869 -> V P I 0.019 0.039 0.004
13:46:49.373 -> V P I 0.022 0.044 0.005
13:46:50.860 -> V P I 0.025 0.051 0.006
13:46:52.359 -> V P I 0.029 0.060 0.007
13:46:53.856 -> V P I 0.000 0.000 0.000
13:46:55.345 -> V P I 0.000 0.000 0.000
```

Figure 5. Output from Sketch 1.

The Nano is supported by a standard Arduino IDE installation and can be selected from Tools→Board in the IDE window. For a project like this, the Nano is completely code-compatible with an UNO. See https://www.makerguides.com/arduino-nano/#arduino-nano-pinout for a pinout diagram.

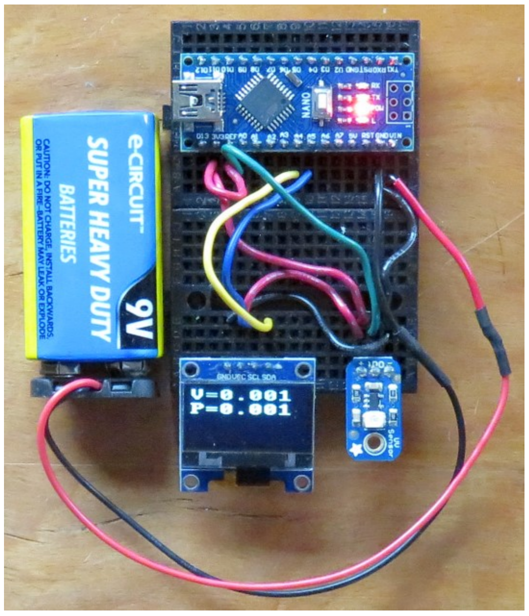


Figure 6. GUVA-S12SD with Arduino Nano and OLED.

Clones of the original Arduino Nano are widely available online for only a few dollars. They may be older versions of the "official" Arduino Nano. If so, code may not upload using the default ATmega328 processor option. Instead, you must click on the Tools menu and select the ATmega328 (Old Bootloader) option; as far as I know, there's no way to tell ahead of time if this is necessary just by looking at the board, but you will quickly find out!

The green wire sends the GUVA-S12SD analog output to pin A0. The yellow/blue wires for the OLED's I2C SDA/SCL connections go to pins A4 and A5. (There's not another pair of dedicated SDA/SCL header pins on the Nano like there is on an UNO.) The short red jumper wire connects the 3.3 V power pin to the AREF pin.

Sketch 2 shows the code for this setup, which is similar to Sketch 1 with OLED code added. This code uses a software library, U8x8lib , which is a character-only subset of the U8g2 library that includes some graphics capabilities not needed for this project. You can install this library from the Arduino IDE window by selecting Sketch→Include Library→Manage Libraries and entering U8g2 or U8x8lib in the search box. The Wire library for I2C devices is included in an Arduino IDE installation. The `U8X8_SSD1306`... statement may need to be changed for other OLED modules that *look* like but aren't *exactly* the same as this one; you can find examples online or in the software library folder for using these 1306 OLED displays that provide a large selection of possible choices for instantiating (accessing) your device.

Sketch 2. Using the GUVA-S12SD with an OLED display.

```
/* GUVA_S12SD_OLED.ino D. Brooks August2021
  Tests the GUVA-S12SD UV sensor using the 3.3V power pin
  as an external reference voltage for A-D-A conversions.
  Connect the 3.3 V power pin to the AREF pin.
  This code adds an IZOKEE OLED for data display.
  See http://www.geni-uv.com/download/products/GUVA-S12SD.pdf
*/
const float aRef=3.30; // Check this value on your Arduino!
const int UVpin=A0,N=10,DT=50;
long int DelayTime=1000L;
float I,P,Vsum;
int i;
#include <Wire.h>
#include "U8x8lib.h"
U8X8_SSD1306_128X64_NONAME_HW_I2C u8x8(/* reset=*/ U8X8_PIN_NONE);
void setup() {
  Serial.begin(9600);
  u8x8.begin();
  //u8x8.setFont(u8x8_font_chroma48medium8_r);
  //This is a better (bigger) font.
  u8x8.setFont(u8x8_font_px437wyse700a_2x2_r);
  analogReference(EXTERNAL); // Absolutely REQUIRED here!
  pinMode(UVpin,INPUT);
}
void loop() {
  Vsum=0.;
  for (i=1; i<=N; i++) {
    Vsum+=analogRead(UVpin);delay(DT);
  }
  Vsum=(Vsum/N)*aRef/1023.;
  I=Vsum/4.3; // I in uA, see Adafruit.com documentation.
  P=I*1000./113.;  // I in nA, see GUVA-S12SD datasheet.
  Serial.print("V I P ");
  Serial.print(Vsum,3); Serial.print(' ');
  Serial.print(I,3); Serial.print(' ');
  Serial.println(P,3);
  u8x8.setCursor(0,0);
  u8x8.print("V="); u8x8.println(Vsum,3);
  u8x8.print("P="); u8x8.println(P,3);
  delay(DelayTime);
}
```

Logging data from the GUVA-S12SD sensor

Sketch 2 showed how to implement a UV system that would be suitable for intermittent outdoor use with a battery. This section shows how to record UV data with a date/time stamp continuously over a longer period of time. You could still use this system to record data intermittently because with UNO programming when you re-open a data file that already exists, new data is added to the end of the existing file. (See https://instesre.org/papers/ProgrammingGuide.pdf for more about data logging with Arduinos.)

Logging data with an Arduino UNOs and compatibles requires a data logging module (called a "shield") like the one shown in the left-hand image in Figure 7. The shield includes an SD card slot and a real time clock with CR1220 coin cell backup that will store the date and time literally for years when the board isn't under power; sometimes the coin cell is included with a purchase and sometimes not because of restrictions on shipping lithium-based batteries.

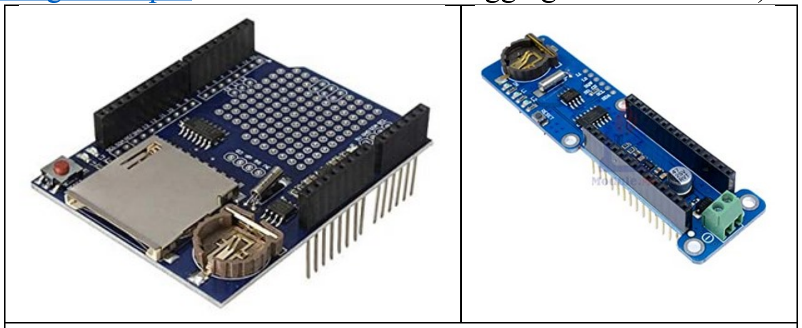


Figure 7. Data logging shield for Arduino UNOs (left) and Nanos (right).

An alternative used for this project is a more recently developed data logging shield for Nanos, shown on the right in Figure 7. A microSD card slot is out of sight on the underside of the board. The 2-wire screw terminal is for a 9-12 VDC power supply, taking the place of the 2.1 mm power input jack on an UNO. The Nano fits in the headers, with its miniB USB connector on the same end as the screw terminals. There's no reason to choose one of these boards or shields over the other, as the code is exactly the same for either one. Nanos require slightly less power than UNOs. Both data logging modules use the same CR1220 coin cell and both have a DS1307 real time clock module; the only difference is the SD cards. Some online commentary on these relatively inexpensive offshore devices suggest that the quality control may not be very good. I have used several of them and haven't had any problems.

With any data logging shield that records a date/time stamp along with data, you must first set the clock based on your computer's clock. Sketch 3 gives code to do this.

Sketch 3. Setting the real time clock on a data logging shield.

```
/* RTC_2.ino, D. Brooks, August 2018
  This code has been modified to work with both old
  and new Adfruit datalogging shields, plus a separate DS3231 module.
  The clock chip on the old and new Adafruit shields are different.
*/
#include <Wire.h>
#include <RTClib.h>
RTC_DS1307 RTC; // clock for most Arduino data logging shields
//RTC_PCF8523 RTC;
//RTC_DS3231 RTC;
void setup() {
  Serial.begin(9600);
  Wire.begin();
  RTC.begin();
  if (RTC.begin()) Serial.println("clock running...");
  else Serial.println("clock not running...");
  RTC.adjust(DateTime(__DATE__, __TIME__));
  DateTime now=RTC.now();
  Serial.println("Setting date/time...");
  Serial.print(now.year());   Serial.print('/');
  Serial.print(now.month());  Serial.print('/');
  Serial.print(now.day());    Serial.print(' ');
  Serial.print(now.hour());   Serial.print(':');
  Serial.print(now.minute()); Serial.print(':');
  Serial.println(now.second());
}
void loop() {
}
```

It's important to understand that if you upload and run this sketch, and then later power your Arduino from an external supply, the RTC-2 sketch will still be in memory and will try to initialize the date and time again. This will give a junk result because there's no "computer time" available. So, initializing the RTC module and then installing a sketch to record data is a two-step process:

1. Upload and run RTC-2 through the USB cable from your computer.
2. Keep the computer USB cable in place for power and upload the new data logging sketch.

Because the RTC-2 sketch is still your Arduino's memory, it will update the date and time again when you apply power, but then the RTC-2 code will be replaced with your new sketch.

The DS1307 clock modules on these data logging shields aren't perfect. Over time – a few weeks or months – all of these modules, regardless of their source, will gain or lose a few minutes. The rate at which time is gained or lost depends on the ambient temperature because the oscillators on the modules aren't temperature compensated and their oscillating frequency changes with temperature. Some online commentary suggests that the oscillators in inexpensive DS1307 modules aren't very accurate, which may make the time drift worse. Whether time drift is a problem depends on your application. Of course, you can always reinitialize the date/time by rerunning RTC-2 and loading your data collection sketch again, following the process given above.

Sketch 4 gives code for logging GUVA-S12SD data. It's a version of Sketch 1 with data logger code added. There's roughly a three-minute interval between logging activity. If you're using this intermittently, perhaps hand-held outdoors, you can watch the on-board LED that turns on when the system is writing to the SD card and off when it's finished. This should prevent you from removing power from the board during a write operation. You could replace references to the on-board LED at digital pin 13 with an off-board LED on some other digital pin if you wanted a more visible "writing now" signal – for example, if you mounted this system in a case that would block view of the on-board LED. Or you can just comment out the lines that enable this option. And, of course, you can increase or decrease the `DelayTime` value.

Sketch 4. Logging data from a GUVA-S12SD.

```
/* GUVA_S12SD_log.ino D. Brooks August2021
  See GUVA_S12SD.ino.
  Logs averaged data with date/time stamp.
*/
const float aRef=3.3; // Check this value on your board!
float I,P,sum;
const int UVpin=A0,N=10,DT=50;
long int DelayTime=179500L; // ~3 minute logging interval
#include <Wire.h>
#include <SD.h>
#include <RTClib.h>
RTC_DS1307 rtc;  // for most data  logging shields
const int SDpin=10; // Don't change this pin!
int i;
File logfile;
char filename[]="GUVA_LOG.CSV";
int year,month,day,hour,minute,second;
void setup() {
  Serial.begin(9600);
  pinMode(13,OUTPUT); // on-board LED to signal logging
  analogReference(EXTERNAL); // Absolutely REQUIRED here!
  pinMode(UVpin,INPUT);
  rtc.begin();
  Wire.begin();
  if (!SD.begin(SDpin)) {Serial.println("Card failed.");
    delay(50);exit(0);
  }
  Serial.println("card initialized.");
  logfile = SD.open(filename, FILE_WRITE);
  if (!logfile) {
    Serial.println("Couldn't create file."); delay(50); exit(0);
  }
  Serial.print("Logging to file "); Serial.println(filename);
}
void loop() {
  sum=0;
  for (i=1; i<=N; i++) { // average over N readings.
    sum+=analogRead(UVpin);delay(DT);
  }
  sum=(sum/N)*aRef/1024.;
  I=sum/4.3; // I in uA, see Adafruit.com documentation.
  P=I*1000./113.; // mW/cm^2, I in nA, see GUVA-S12SD datasheet.
  // Serial port prints just for testing. They can be removed.
  Serial.print("I P ");
  Serial.print(sum,3); Serial.print(' ');
  Serial.print(I,3); Serial.print(' ');
  Serial.println(P,3);
  DateTime now=rtc.now();
  year=now.year(); month=now.month(); day=now.day();
  hour=now.hour(); minute=now.minute(); second=now.second();
  digitalWrite(13,HIGH); // turn on LED
  logfile.print(year); logfile.print('/'); logfile.print(month);
  logfile.print('/');
  logfile.print(day); logfile.print(','); logfile.print(hour);
  logfile.print(':');
  logfile.print(minute); logfile.print(':');
  logfile.print(second);logfile.print(',');
  // Write day expressed as decimal fraction.
  logfile.print(day+hour/24.+minute/1440.+second/86400.,6);
  logfile.print(',');
  logfile.print(I,3); logfile.print(',');
  logfile.println(P,3);
  logfile.flush();
  digitalWrite(13,LOW); // turn off LED
  delay(DelayTime);
}
```

Figure 8 shows the hardware layout used with this code. A short red jumper wire at the lower left-hand corner of the Nano connects the 3.3 V power pin to the AREF pin just to its right. The green wire goes to the A0 analog input pin, which is just to the right of the AREF pin. In the right-hand image, the microSD card can be seen extending past the top edge of the shield. You could power this system with 6 C or D cells in series through the VIN and GND pins, which are the two rightmost pins on the lower row of Nano pins as shown in this image, or through the screw terminals (hidden under the USB cable). For short periods of time, 6 AA cells should also work.

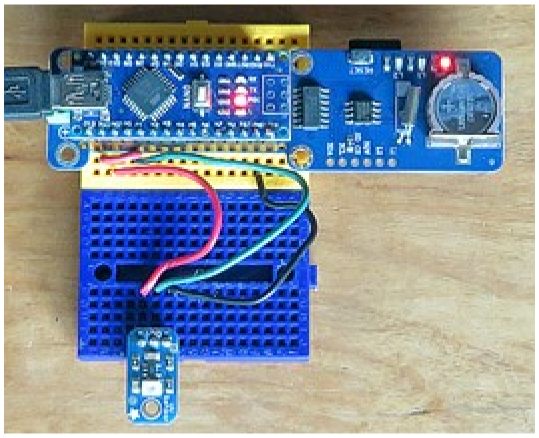


Figure 8. Data logging with GUVA-S12SD.

Figure 9 shows data from Sketch 4 taken on a day with cumulus clouds moving across a bright blue sky. This site is at about 40°N latitude. Insolation data recorded from a Kipp & Zonen SP-Lite pyranometer is also shown. What's significant about this graph is not the relative values of insolation and UV power, but that the *shape* of the UV curve agrees reasonably well with the insolation curve; the significance of this observation will be discussed later.

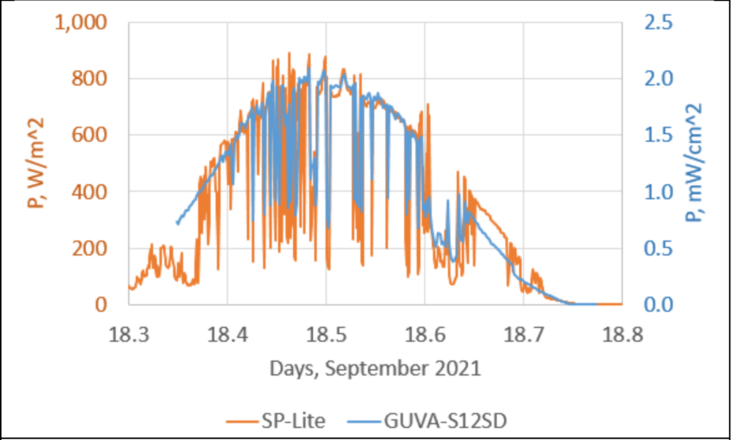


Figure 9. Some GUVA-S12SD UV power data and insolation data.

During the morning, trees around the data collection site, east and south of the instruments shade direct sunlight and have a large impact on the insolation values as the sun rises in the sky, but not on the UV values. Later, when the sun is above the trees, the spikes and large dips in the insolation data are due to cumulus clouds blocking and reflecting direct sunlight; those clouds have a relatively smaller effect in blocking UV radiation. The smaller signal later in the day, starting around 18.65 (about 3:30 EST) may be due to the fact that the GUVA-S12SD is sitting on a small table on the ground while the pyranometer is mounted on a trellis about two meters higher. The pyranometer is carefully leveled to be precisely horizontal, but the instruments on the table were not and this is especially evident when the sun is low in the sky.

Other single-detector UV devices

It's worth comparing the GUVA-S12SD to other UV devices. They will all give different results because different detectors respond differently across the UV wavelength spectrum.

ML8511

Like the GUVA-S12SD, the ML8511 provides an analog output, using a transimpedance amplifier to convert a small photocurrent to a usable output voltage. But its performance is much different from the GUVA-S12SD. Here's the datasheet:
https://www.digikey.com/htmldatasheets/production/1342229/0/0/1/ml8511-datasheet.html.
Unlike the GUVA-S12SD, this datasheet doesn't include a calculation for UVI. Rather, perhaps in recognition of the limitations of a single-detector sensor for that job, it provides just a conversion from output voltage to UV power.

Figure 10 shows the sensor module with the detector facing up in the upper right-hand corner. The connections are marked only on the bottom of the board so the Arduino connections are shown here. For a size reference, the connection points are on standard 0.1" centers. Figure 10 also shows the conversion from output voltage to UV power and the spectral response characteristics for this detector. Note that this detector has some spectral response in the visible spectrum – wavelengths greater than about 400 nm. That response seems small, but it's significant because solar input increases rapidly as wavelengths rise into the visible spectrum.

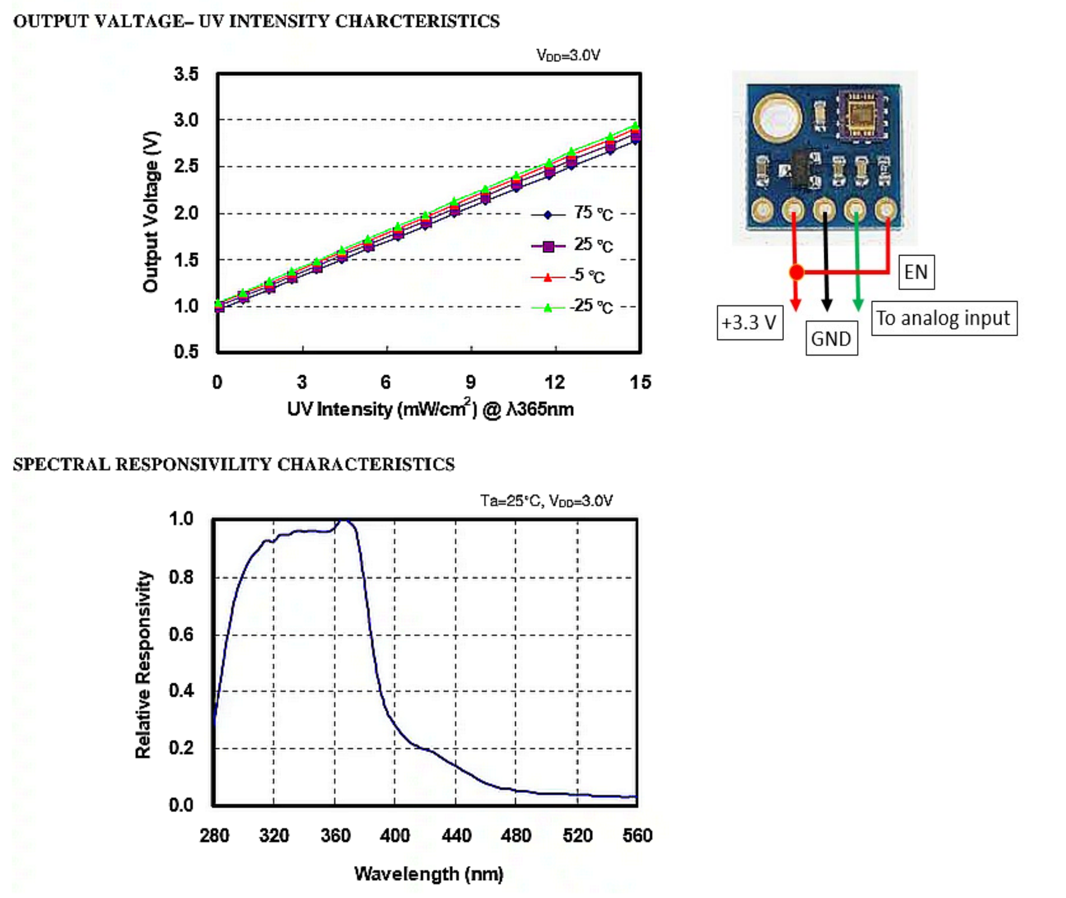


Figure 10. ML8511 UV intensity and spectral response characteristics. https://www.digikey.com/htmldatasheets/production/1342229/0/0/1/ml8511-datasheet.html
For more information about this sensor: https://cdn.sparkfun.com/assets/learn_tutorials/2/0/6/ML8511_UV.pdf

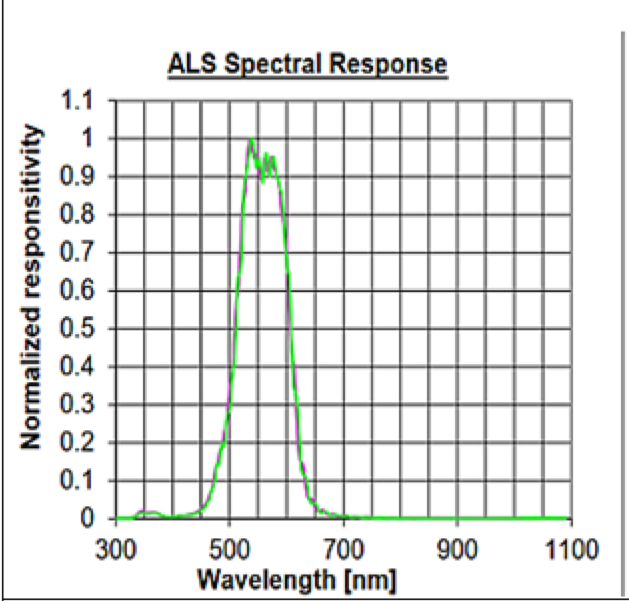
A table entry in the ML8511 datasheet says that the output voltage at 10 mW/cm^2 is 2.20±0.12 V, *i.e.*, 2.20±5% V. The output is only weakly temperature dependent. A connection diagram in the datasheet shows a 0.1 µF capacitor between the VDD (power) and GND pins. It's not clear whether this "noise filter" is required when the sensor mounted on a breakout board. (I didn't include a capacitor for this project and the output wasn't noticeably noisy.) Based on this table entry, and ignoring the small temperature dependence,

(3) $P\ (\text{mW/cm}^2) = (V - 0.98)/0.12$

where the 0.98 is the voltage observed under dark conditions for the module I used. (Not 1.0 as given in the reference cited in Figure 10, and perhaps a little different for different modules.)

LTR-390UV

The LTR-390UV has both visible light and UV sensors, as shown in Figure 11. The visible sensor peaks around 550 nm (green light). The UV sensor peaks around 320 nm, at the lower end of the UVA spectrum. The LTR390 breakout board shown in Figure 11 is from Adafruit. For a size reference, the six connection points are on 0.1" centers; four connections are required – power, ground and, because this is an I2C rather than analog-output device, SDA/SCL.



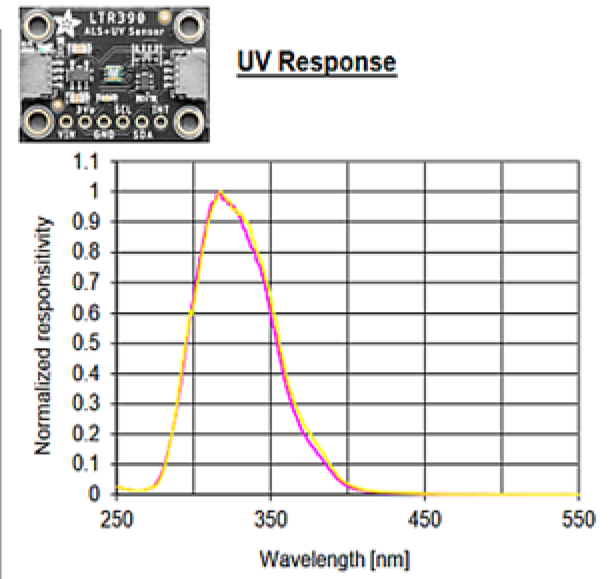
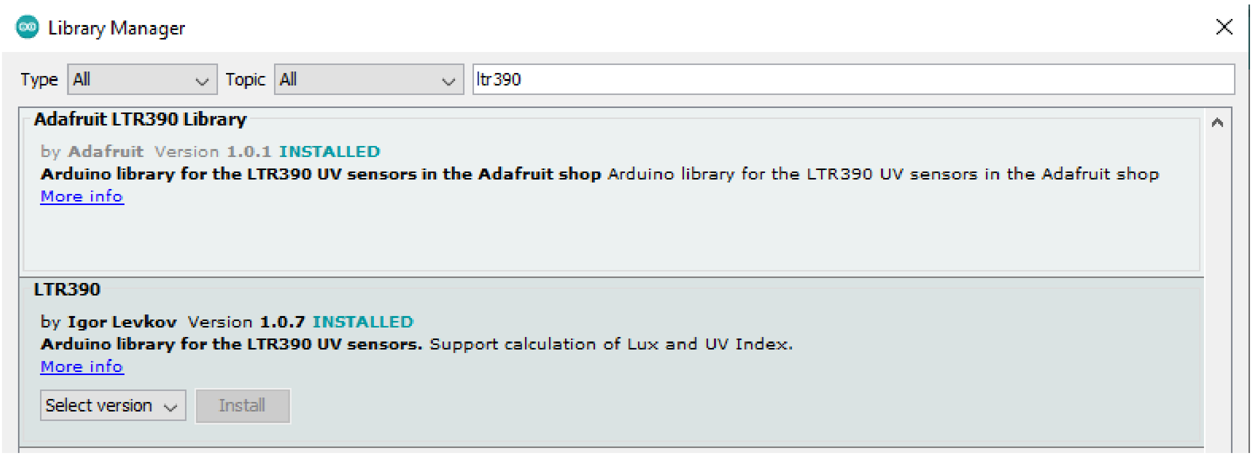


Figure 11. Spectral response characteristics of LTR390 visible/UV sensor from Figures 4.1 and 4.2 at https://optoelectronics.liteon.com/upload/download/DS86-2015-0004/LTR-390UV_Final_%20DS_V1%201.pdf

Adafruit provides a software library for its LTR-390UV board. I installed that library, but later switched to the library by Igor Levkov because it includes a function for getting the UVI index), which the example code in the Adafruit library does not. (In principle, outputs available through the Adafruit library could be used to calculate UVI from information in the data sheet cited in Figure 11.)



ltr390

Adafruit LTR390 Library
by **Adafruit** Version **1.0.1** **INSTALLED**
Arduino library for the LTR390 UV sensors in the Adafruit shop Arduino library for the LTR390 UV sensors in the Adafruit shop
More info

LTR390
by **Igor Levkov** Version **1.0.7** **INSTALLED**
Arduino library for the LTR390 UV sensors. Support calculation of Lux and UV Index.
More info

Logging data from the LTR-390UV

The only useful UV output from the Levkov LTR-390UV library is the UV index. Equations for getting UVI values are "hidden" in the software library files and aren't directly available from the sample code provided with the Levkov library. Those equations are presumably based on information in the LTR-390UV data sheet; a comparison between UVI values from the LTR-390UV and a reference instrument is shown in Figure 12. The differences are always less than 1 UV unit even under very bright summer sunlight conditions at low to moderate elevation conditions.

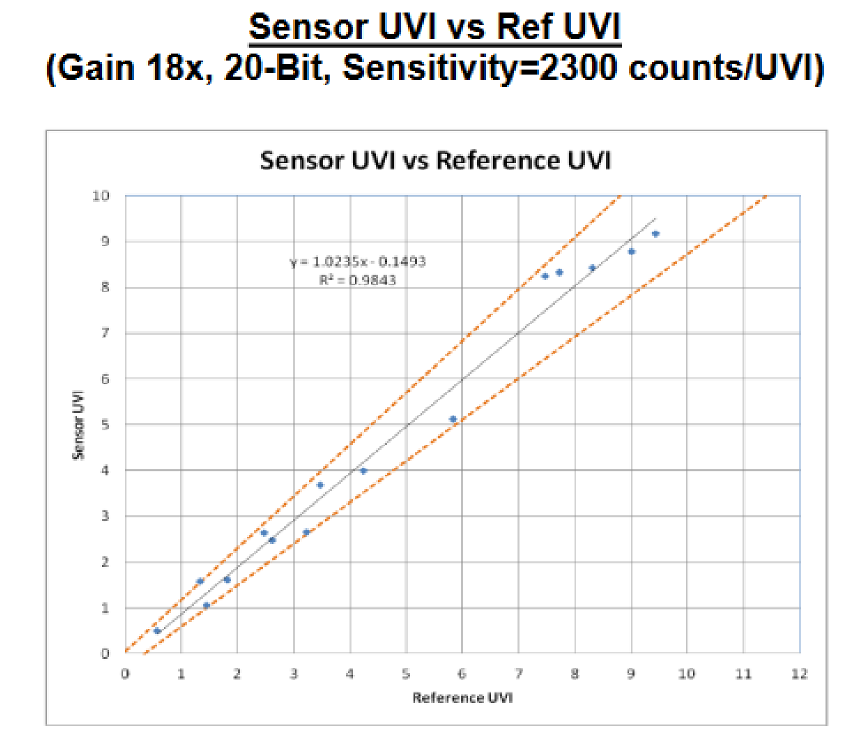


Figure 12. LTR-390UV compared to a reference instrument, from Figure 4.6 at https://optoelectronics.liteon.com/upload/download/DS86-2015-0004/LTR-390UV_Final_%20DS_V1%201.pdf.

Code for logging data from an LTR-390UV next to an ML8511 is shown in Sketch 5. This is an extension of the example sketch from the Levkov library, with additional code for logging on a Nano system shown in Figure 13, similar to the system shown above in Figure 8. It includes some code that isn't actually required for this application, but I didn't bother to remove that code.

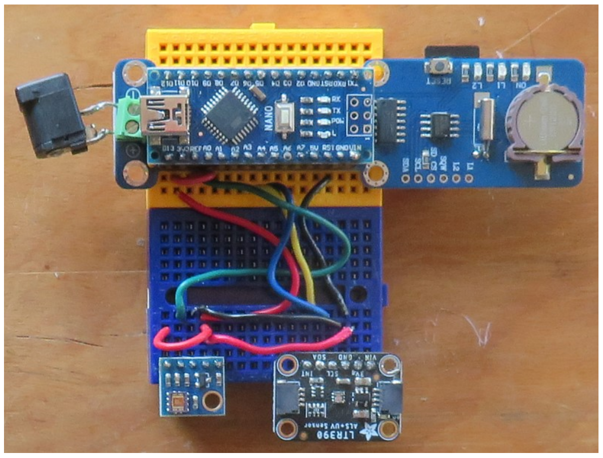


Figure 13. Arduino Nano system for logging data from ML8511 and LTR-390UV (lower right-hand corner).

Sketch 5. Logging data from LTR-390UV and GUVA-S12SD.

```
/* LTR390_UVI_ML8511_log.ino, D. Brooks, November 2021
  Tests LTR390 along with ML8511. Uses Igor Levkov LTR390 library
  instead of Adafruit library because the LTR390 library includes
  a function for UVI, which the Adafruit library does not.
*/
#include <Wire.h>
#include <LTR390.h>
#define I2C_ADDRESS 0x53
#include <SD.h>
#include <RTClib.h>
RTC_DS1307 rtc;
File logfile;
char filename[]="LTR_ML.CSV";
int year,month,day,hour,minute,second;
const int SDpin=10;
const long int DelayTime=60000L;
const float aRef=3.3; // for external reference.
const int UVpin=A0;
float V,LTR_UVI,ML_UVI;
/* There are several ways to create your LTR390 object:
 * LTR390 ltr390 = LTR390()                -> uses Wire / I2C Address = 0x53
 * LTR390 ltr390 = LTR390(OTHER_ADDR)      -> uses Wire / I2C_ADDRESS
 * LTR390 ltr390 = LTR390(&wire2)  -> uses TwoWire object wire2 I2C_ADDRESS
 * LTR390 ltr390 = LTR390(&wire2, I2C_ADDRESS) -> all together
 * Successfully tested with two I2C busses on an ESP32
*/
LTR390 ltr390(I2C_ADDRESS);
void setup() {
  Serial.begin(9600); rtc.begin();
  Wire.begin();
  //rtc.adjust(DateTime(__DATE__,__TIME__)); // use only for new RTC not set.
  if (!SD.begin(SDpin)) {Serial.println("Card failed.");
    delay(50);exit(0);
  }
  Serial.println("card initialized.");
  logfile = SD.open(filename, FILE_WRITE);
  if (!logfile) {
    Serial.println("Couldn't create file."); delay(50); exit(0);
  }
  Serial.print("Logging to file "); Serial.println(filename);
  // Optionally, write header line here, with logfile.flush
  logfile.println("date,time,day_frac,LTR_UVI,ML8511_UVI");
  logfile.flush();
  analogReference(EXTERNAL); // absolutely required here!
  Wire.begin();
  if(!ltr390.init()){
    Serial.println("LTR390 not connected!");
  }
  //ltr390.setMode(LTR390_MODE_ALS);
  ltr390.setMode(LTR390_MODE_UVS);
  ltr390.setGain(LTR390_GAIN_3);
  Serial.print("Gain : ");
  switch (ltr390.getGain()) {
    case LTR390_GAIN_1: Serial.println(1); break;
    case LTR390_GAIN_3: Serial.println(3); break;
    case LTR390_GAIN_6: Serial.println(6); break;
    case LTR390_GAIN_9: Serial.println(9); break;
    case LTR390_GAIN_18: Serial.println(18); break;
  }
  ltr390.setResolution(LTR390_RESOLUTION_18BIT);
  Serial.print("Resolution : ");
  switch (ltr390.getResolution()) {
    case LTR390_RESOLUTION_13BIT: Serial.println(13); break;
    case LTR390_RESOLUTION_16BIT: Serial.println(16); break;
    case LTR390_RESOLUTION_17BIT: Serial.println(17); break;
    case LTR390_RESOLUTION_18BIT: Serial.println(18); break;
    case LTR390_RESOLUTION_19BIT: Serial.println(19); break;
    case LTR390_RESOLUTION_20BIT: Serial.println(20); break;
  }
  //ltr390.setThresholds(100, 1000);
  //ltr390.configInterrupt(true, LTR390_MODE_UVS);
}
void loop() {
  if (ltr390.newDataAvailable()) {
      if (ltr390.getMode() == LTR390_MODE_ALS) {
         Serial.print("Ambient Light Lux: ");
         Serial.println(ltr390.getLux());
         ltr390.setGain(LTR390_GAIN_18);        //Recommended for UVI - x18
         ltr390.setResolution(LTR390_RESOLUTION_20BIT);   //for UVI - 20-bit
         ltr390.setMode(LTR390_MODE_UVS);
      } else if (ltr390.getMode() == LTR390_MODE_UVS) {
         Serial.print("LTR UV Index: ");
         LTR_UVI=ltr390.getUVI();
         Serial.println(LTR_UVI);
         ltr390.setGain(LTR390_GAIN_3);        //Recommended for Lux - x3
         ltr390.setResolution(LTR390_RESOLUTION_18BIT);   //for Lux - 18-bit
         //ltr390.setMode(LTR390_MODE_ALS);

      }
  }
  V=(float)analogRead(UVpin);
  V=V*aRef/1023;
  ML_UVI=10.4863*V-10.4076;
  Serial.print("ML8511 UVI ");
  Serial.println(ML_UVI);
  Serial.println("----------------");
  delay(DelayTime);
  // Get date and time.
  DateTime now=rtc.now();
  year=now.year(); month=now.month(); day=now.day();
  hour=now.hour(); minute=now.minute(); second=now.second();
  logfile.print(year); logfile.print('/'); logfile.print(month);
  logfile.print('/');
  logfile.print(day); logfile.print(','); logfile.print(hour);
  logfile.print(':');
  logfile.print(minute); logfile.print(':');
  logfile.print(second);logfile.print(',');
  // Write day expressed as decimal fraction.
  logfile.print(day+hour/24.+minute/1440.+second/86400.,6);
  logfile.print(',');
  logfile.print(LTR_UVI,2); logfile.print(',');
  logfile.println(ML_UVI,2); logfile.flush();
```

}

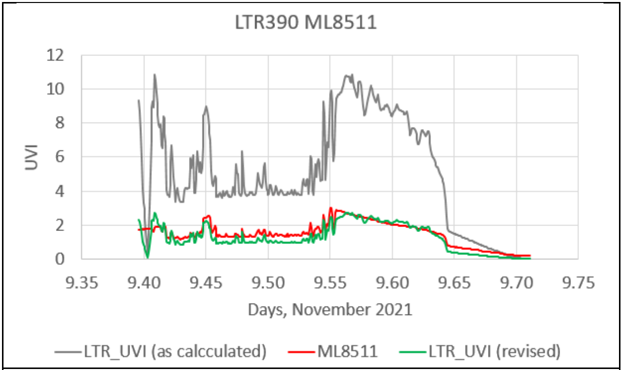


Figure 14. UVI data from LTR-390UV and ML8511.

Figure 14 shows some data from the system shown in Figure 13. The UVI values as calculated in Sketch 5 are clearly ridiculous for this time of year, but this appears to be due to my error. Sketch 5, with code from the Levkov Library, assigns 18-bit A-to-D resolution, but Figure 12 indicates that the conversion from raw UV data to UVI should be based on 20-bit resolution; dividing by a factor of 4 (the green line) puts the LTR-390UV data in good agreement with UVI values obtained from a ML8511. A later section in this document describes how to extract UVI values from the ML8511; these values are reasonable for this location and time of year, but they have not been compared to a reliable external reference. (November is certainly not the best time of year to be investigating UVI data in the northern hemisphere!)

Low-power data collection systems

For logging data continuously for long periods of time away from plug-in power sources a battery or solar/battery supply system is the usual solution. But for data that needs to be recorded only intermittently – see Sketch 4, above, which records data about every three minutes – keeping the system under power all the time is very inefficient.

There are many online discussions about code to put Arduino systems into various low-power modes. But a better solution is simply to turn off the entire system when it doesn't have to do anything. This can be done by using an inexpensive TPL5110 low-power on/off timer module; I use the version from adafruit.com. (See Chapter 21 in *Exploring Your Environment with Arduino Microcontrollers* for an example of using this timer.) This device uses a (very small!) on-board potentiometer to determine a power cycle time. When the cycle starts, a system operating voltage is passed through the module to the Arduino, which executes the sketch in its memory and then sends an "off" signal to the TPL5110. While the power is in the "off" part of the cycle, the TPL5110 draws only a few micro amps.

For a system like a Nano or UNO you must provide 5 V to the TPL5110 through a 5 VDC regulated source or step-up or step-down regulator; this project uses a Pololu S7V7F5 step-up/step-down adjustable regulator (https://www.pololu.com/product/2118) with 6 D cells in series.

Another possibility is to use a solar/battery system with a solar power manager board like the DFRobot DFR0580 (https://www.dfrobot.com/product-1795.html) with a nominal 12 V solar panel (~18-22 V no-load) and a sealed lead-acid battery. These devices simultaneously charge the battery and run your system during daylight hours and automatically switch to battery power to run your system at night. The DFR0580 has three separate regulated 5 V outputs that can directly power a TPL5110 module.

A solar/battery system will run continuously as long as a fully charged battery will power the system throughout long winter nights and the solar panel is sized appropriately to fully charge the battery during short winter days even during extended periods of cloudy/rainy/snowy skies. This can be a challenge but, fortunately, the project discussed in this document requires very little power from a battery. In fact, because this TPL5110-enabled on/off system is off for almost all of the time – the code required to collect and log data takes only a few tens of milliseconds to execute – with an on/off cycle even as short as one minute you might not need a solar/battery system. Such a system should run for months with 6 C or D cells in series powering the timer board through a 5 V step-up/step-down regulator.

On the Adafruit version of the TPL5110, the potentiometer value is read between the GND and Delay pins on the module, when there's no power being applied to the module. You can find a table of cycle length times at https://cdn-learn.adafruit.com/downloads/pdf/adafruit-tpl5110-power-timer-breakout.pdf?timestamp=1629482617. It's not possible to set the potentiometer precisely (with a small jeweler's screwdriver), but I set mine at around 30 kΩ, for a cycle time of about 2 minutes. Some online sources suggest putting a resistor between the Done and GND pins to make sure that the cycling signal works properly; I used a 10 kΩ resistor. This system is shown in Figure 11.

With such a system, all the code to carry out the required tasks is in the setup() function – there should be no code to cycle continuously in the loop() function, as that would defeat the purpose of an on/off cycle. When that code has been executed (just once), the Arduino sends a "turn off" signal back to the TPL5110 and the cycle starts again. Except for defining and setting the turn off pin on the Arduino, no code is required to support the TPL5110; as far as your Arduino is concerned, all the power management work is done "off line."

Figure 15 shows a Nano/TPL5110 battery-powered system with a GUVA-S12SD and a ML8511. (You could also use an UNO and compatible data logging shield and you could also use an LTR-390UV in addition to or in place of either of these sensors) The battery input from 6 D cells goes through the Pololu S7V7F5 adjustable step-up/step-down regulator cited above, set to just under 5.00 V. There are no LEDs lit on the Nano in this image because the "on" cycle lasts for only about 1 second every couple of minutes! The TPL5110 is the device second from the bottom left. Pushing the small black button on that board will trigger an "on" cycle at any time, but the TPL5110 will automatically cycle indefinitely as long as 5 V is applied to it. The 10 kΩ resistor from the "done" pin on the left end of the board to ground may or may not be required – try it with and without and see!

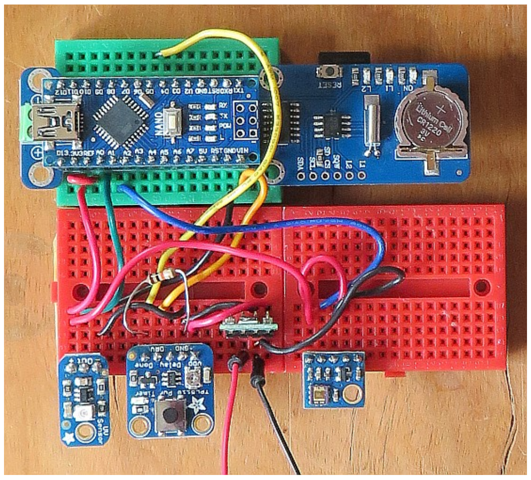


Figure 15. Battery-powered system using TPL5110 on/off timer.

One very important point about such systems is that you should never connect the Arduino board to USB or other external power source when the TPL5110 board is connected to the system, as this may destroy the TPL5110. Whenever you upload code, you must first disconnect the TPL5110 board.

Sketch 6 shows the code for this system. Only the output voltage and power are logged; the UVI calculations, for whatever they might be worth, are discussed later. The Serial.print() statements are just for testing, and can be commented out. With the code as is, removing the TPL5110 and powering the system through a USB cable will produce one set of output values written to the SD card and displayed in the serial port window.

Sketch 6. Logging GUVA-S12SD and ML8511 data with TPL5110 on/off timer.

```
/* GUVA_S12SD_ML8511_LowPower.ino, D. Brooks, August 2021
  Adds ML85111 for logging. See GUVA_S12SD_LowPower.
*/
const float aRef=3.3; // Check this value on your board!
const int GUVApin=A0,MLpin=A1,N=10,DT=10;
const int donePIN=5; // digital pin 5 for sending "off" to TPL5110.
#include <Wire.h>
#include <SD.h>
#include <RTClib.h>
RTC_DS1307 rtc;  // datalogger clock
const int SDpin=10;
File logfile;
char filename[]="GUVA_LOG.CSV";
int YR,MON,DAY,HR,MIN,SEC;
int ML_digitized;
float I_GUVA,V_GUVA,V_ML,P_GUVA,P_ML;
void setup() {
  Serial.begin(9600);
  pinMode(donePIN,OUTPUT); digitalWrite(donePIN,LOW);
  analogReference(EXTERNAL); // Absolutely REQUIRED here!
  pinMode(GUVApin,INPUT); pinMode(MLpin,INPUT);
  rtc.begin();
  Wire.begin();

  if (!SD.begin(SDpin)) { //Serial.println("Card failed.");
    delay(50);exit(0);
  }
   //Serial.println("card initialized.");
  logfile = SD.open(filename, FILE_WRITE);
  if (!logfile) {
    //Serial.println("Couldn't create file.");
    delay(50); exit(0);
  }
  // GUVA_S12SD
  V_GUVA=getV(N,DT,GUVApin);
  I_GUVA=V_GUVA/4.3; // I in uA, see Adafruit.com documentation.
  P_GUVA=I_GUVA*1000./113.;  // mW/cm^2
  // ML8511
  V_ML=getV(N,DT,MLpin);
  P_ML=(V_ML-1.)/0.12; // mW/cm^2
  DateTime now=rtc.now();
  YR=now.year(); MON=now.month(); DAY=now.day();
  HR=now.hour(); MIN=now.minute(); SEC=now.second();
  Serial.print(HR);Serial.print(':');
  Serial.print(MIN);Serial.print(':');
  Serial.print(SEC);Serial.print (' ');
  Serial.print(V_GUVA,3);Serial.print(' ');
  Serial.print(I_GUVA,3);Serial.print();
  Serial.print(P_GUVA,3); Serial.println(' ');
  Serial.print(V_ML,3); Serial.print(' ');
  Serial.println(P_ML,3);
  logfile.print(YR); logfile.print('/'); logfile.print(MON);
  logfile.print('/');
  logfile.print(DAY); logfile.print(','); logfile.print(HR);
  logfile.print(':');
```

```
    logfile.print(MIN); logfile.print(':');
    logfile.print(SEC);logfile.print(',');
    // Write day expressed as decimal fraction.
    logfile.print(DAY+HR/24.+MIN/1440.+SEC/86400.,6);
    logfile.print(','); logfile.print(V_GUVA,3);
    logfile.print(','); logfile.print(P_GUVA,3);
    logfile.print(','); logfile.print(V_ML,3);
    logfile.print(','); logfile.println(P_ML,3);
    logfile.flush(); // Write to SD card.
    delay(100); // Lets everything finish, maybe optional?
    digitalWrite(donePIN,HIGH); // Turn power off.
  }
  void loop() {
  }
  float getV(int N,int DT,int pin) {
    float sumV=0.;
    for (int i=1; i<=N; i++) {
      sumV+=(float)analogRead(pin);delay(DT);
    }
    return (sumV/N)*aRef/1023.;
  }
```

Figure 16 shows some data from the GUVA-S12SD and ML8511 sensor modules, collected on an early fall day under a clear bright blue sky in the morning with a cumulus cloud moving over the sun just after noon and more clouds moving in starting in early afternoon. As opposed to the deep dips in UV radiation seen in Figure 9, due to individual cumulus clouds moving across the sun, these skies represent a transition to a more lightly overcast condition.

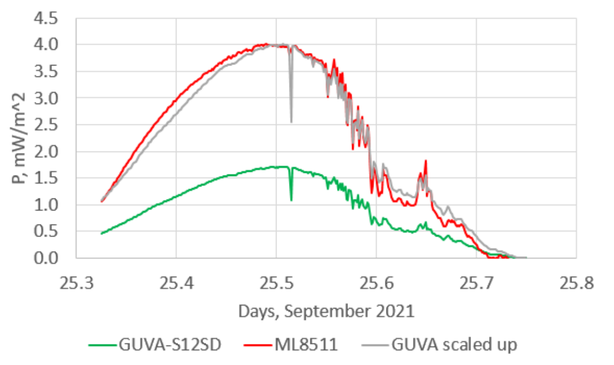


Figure 16. UV power data for GUVA-S12SD and ML9511 sensors.

The green and red lines on the graph show UV power from the two sensors. The values reported by these sensors will be very different because the spectral response of the ML8511 detector response extends into the visible part of the spectrum. Although that detector response isn't very large, the solar irradiance, including that portion of the irradiance which reaches Earth's surface, is rapidly increasing as the wavelength increases from UV to visible values. (See Figure 23, below.) Hence, the ML8511 should be expected to report a much higher UV power because part of the signal is due to irradiance from the visible spectrum.

The gray line in Figure 16 shows the GUVA-S12SD values scaled up to approximate the ML8511 values; these data show that the *shape* of the response of these two sensors to changing solar zenith angle is about the same. The differences between the ML8511 and scaled GUVA-S12SD graph is most likely due to the fact that no particular effort was made to make sure that the plane of these two sensors, just inserted into their breadboards, were carefully aligned to the horizontal.

Figure 17 shows the UV data from the GUVA-S12SD and ML8511 sensors shown in Figure 16 scaled (in arbitrary and meaningless "units") to overlay insolation data from a pyranometer,[1] in units of W/m^2. As previously shown in Figure 9, trees around the site cause large fluctuations in insolation in the early morning, but not in the UV data.

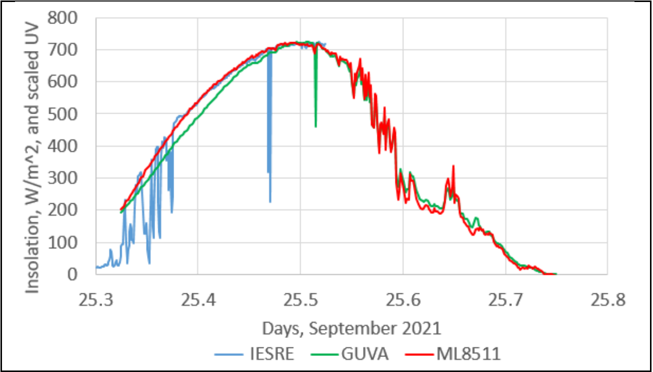


Figure 17. UV data "scaled" to match insolation from cosine-corrected pyranometer.

What's the significance of the shape of the UV graphs compared to the insolation data? In order to accurately represent incoming radiation from the sun and sky, a detector should have good "cosine response."

Consider an amount of direct solar radiation $I_{o,direct}$ received from the sun when it's overhead, that is, when the solar zenith angle Z is 0° (a circumstance that never occurs at non-tropical latitudes, but which doesn't matter for this discussion). At other zenith angles, that direct solar radiation measured on a horizontal surface is reduced by an amount proportional to the cosine of Z:

(4) $I_{direct} = I_{o,\,direct} \bullet \cos(Z)$

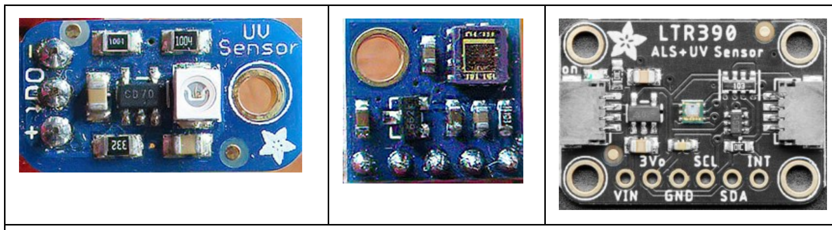


Figure 18. Close-up views of ML8511 (left), GUVA-S12SD (right), and LTR-390UV modules.

Pyranometers always have some kind of diffuser over their detector, often Teflon™, to improve their cosine response. The ML8511, GUVA-S12SD, and LTR-390UV sensors are mounted on their modules without any diffuser, as shown in Figure 18. The GUVA-S12SD detector is recessed in a round cutout in a 3×3 mm white housing. The ML8511 housing, at the upper right-hand corner of its board, is about 2×2 mm, and the LTR-390UV housing, in the center of its board, is even smaller.

Because none of these detectors have any kind of diffuser over them, it's reasonable to expect that they will have poor cosine response. Of the three sensors discussed in this document, only the LTR-390UV's data sheet publishes cosine response for its visible light detector, shown in Figure 19; perhaps the performance will be the same for the UV detector. This response is not great, but OK for such an inexpensive device and may be representative of the response for other surface-mounted sensors.[2]

[1] See https://instesre.org/Solar/Solar_home_page.htm for documentation about the IESRE-developed pyranometer used for these data. This instrument is available assembled and calibrated or as a kit.

[2] The cosine response for this device can be represented approximately by a "pinched cosine" function, $\cos(Z)^{\beta}$, where Z is the solar zenith angle. For these data β=1.5 is about right.

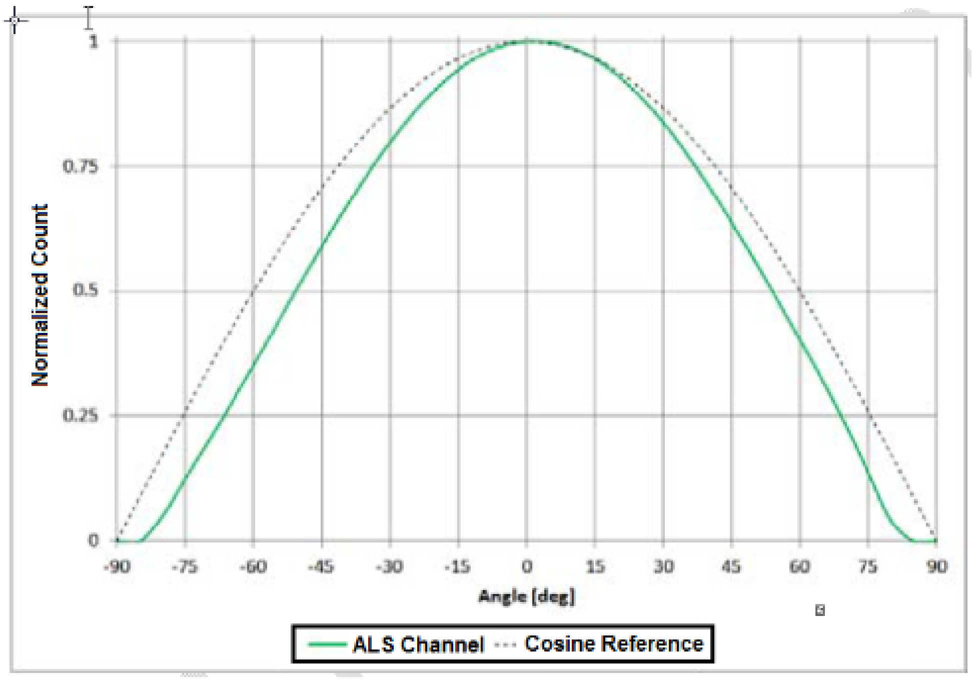


Figure 19. Cosine response for LTR-390UV visible light sensor, from Figure 4.3 at https://optoelectronics.liteon.com/upload/download/DS86-2015-0004/LTR-390UV_Final_%20DS_V1%201.pdf.

In fact, Figures 9 and 17 demonstrate that the cosine response for GUVA-S12SD and ML8511 sensors is also not bad at all compared with a cosine-corrected pyranometer. For measuring radiation from a hemispherical sky source, the housing for the GUVA-S12SD should be a better configuration than the ML8511 detector; however, Figure 17 indicates there isn't much difference and that the ML8511 actually matches the shape of the pyranometer data a little better.

UVI calculations for GUVA-S12SD and ML8511 sensors

Ideally, calculating the UVI from UV measurements requires at least separate UVA and UVB measurements. The no-longer available VEML6075 did this, but all of the devices discussed in this document have only a single UV sensor. Hence, attempts to derive a UVI from these devices requires careful examination. The LTR-390UV library used in this project includes a provision for getting a UVI, as shown in Figure 14, and the data sheet for this sensor is the only one of the three sensors discussed in this document that shows convincing comparisons against calibrated UVI meters, as shown in Figure 12.

There are online documents purporting to show how to convert analog outputs from the GUVA-S12SD and ML8511 to UVI values.

GUVA_S12SD

The GUVA-S12SD is basically just a UVA+B sensor that responds unevenly across the UV spectrum, as shown previously in Figure 2. So, there is ample reason to be concerned about how an accurate UVI can be derived from this measurement. Several online sources state that the accuracy of this sensor (see, for example, https://wiki.dfrobot.com/UV_sensor_SKU__SEN0162) is only ±1 UVI.

The GUVA-S12SD datasheet gives this equation for calculating UVI from photocurrent, according to Figure 4, above:

(5) $\text{UVI} = (I\ (\text{nA}) - 83)/21$

But it makes no sense to calculate the photocurrent according to equation (5) because then the UVI has negative values for small values of the photocurrent, as shown in Figure 20; for a photocurrent of 0 the calculated UVI will be -3.95, a clearly ridiculous result.

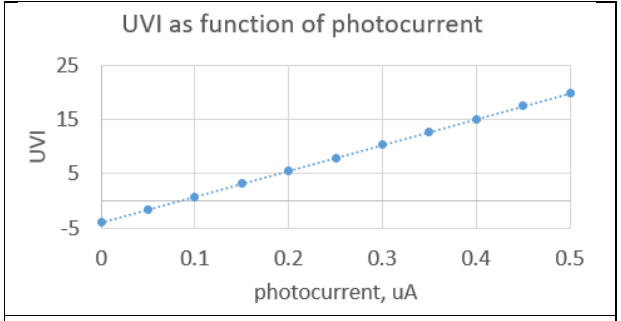


Figure 20. UVI calculated from equation (5).

In the documentation provided by adafruit.com for their GUVA-S12SD module, the suggested UVI calculation is $V_{out}/0.1$. Some results are shown in Figure 21. More reasonably, there are no negative UVI values, but it's not clear how such a calculation relates to Figure 4 or 20.

Dividing the voltage by 0.1, as suggested in Adafruit's documentation, does not yield a reasonable result, as the UVI value of 8 around solar noon is much too high even with a bright cloudless sky for this latitude (~40°N) and time of year.

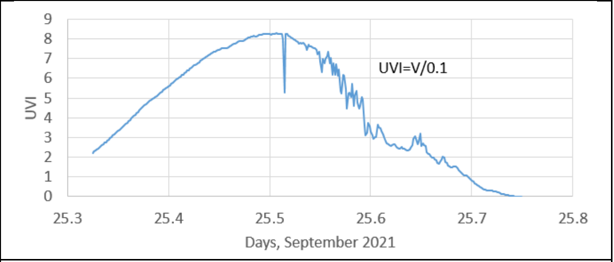


Figure 21. UVI values from GUVA-S12SD sensor module, based on multiplying photocurrent by dividing by 0.1.

The GUVA-S12SD photocurrent, as shown above in Figure 3, may be a reasonably accurate representation of the diurnal variability of UV radiation reaching Earth's surface under varying sun and sky conditions. However, because the GUVA-S12SD response isn't uniform across the UV spectrum and the conversion from photocurrent to power is based just on a laboratory calibration with a 352 nm UV lamp – at the peak wavelength response of the GUVA-S12SD detector – the photocurrent is really just a "proxy" for all the UV radiation reaching Earth's surface.

ML8511

As was true for the GUVA-S12SD, the photocurrent from the ML8511 is also just a proxy for UV radiation reaching Earth's surface; recall from Figure 10 that the ML8511 spectral response extends into the visible part of the solar spectrum. The datasheet for this sensor doesn't include a UVI calculation, but a document from ROHM Semiconductor, the manufacturer of the ML8511 module (UV Photodiode + Amp), *Ultraviolet Sensor IC with Voltage Output* discusses how to calculate UVI values from this device:
https://cdn.sparkfun.com/assets/learn_tutorials/2/0/6/ML8511_UV.pdf.
(This link is to a SparkFun website, but the ML8511 module is no longer available from SparkFun.)

(6) $\text{UVI} = 12.49 \bullet V_{out} - 12.49$ (small UVI)

The document goes on to state that data from "field testing" indicates that an additional offset, with a value of -1.8735, is needed when $V_{out} \geq 1.23$ and $\text{UVI} \geq 1$, giving

(7) $UVI = 12.49 \cdot V_{out} - 14.3635$ for $V_{out} \geq 1.23$

As noted above, the accuracy of calculating a UVI with any single UV detector is problematic. Also, the number if significant digits for numbers in equations (6) and (7) can in no way imply an accuracy at anywhere near this level. In any case, for diurnal monitoring it isn't clear how to apply equations (6) and (7) without having an abrupt discontinuity in the UVI when the output voltage exceeds 1.23 V.

Another document about how to calculate UVI from ML8511 data can be found at https://davidegironi.blogspot.com/2018/05/uv-sensor-ml8511-avr-atmega-library.html#.W20j_rh9hdh.

(8) $UVI_{T=25} = 10.4863 \cdot V_{out} - 10.4076$

at an assumed temperature of 25°C. (Recall from Figure 10 that the temperature dependence for this sensor is small.) Just like equations (6) and (7), (8) is obtained from fitting various parameters and the number of significant digits doesn't imply anywhere near this much accuracy for the result.

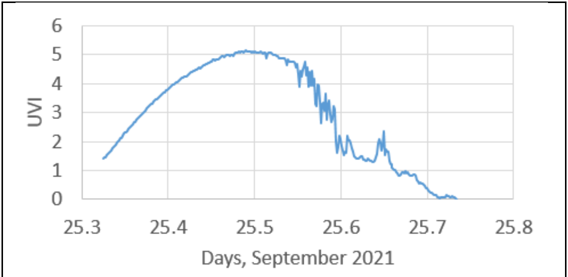


Figure 22. UVI from ML8511 sensor module.

Figure 22 shows some UVI data from the ML8511 sensor on the system shown in Figure 11, using equation (8). The maximum value of about 5 around solar noon seems much more reasonable for these sky conditions and time of year at a site latitude around 40°N than the GUVA-S12SD data in Figure 16. These results suggest that the denominator in the equation shown on Figure 16 should be larger. A value of about 0.16 instead of 0.10 would bring the GUVA-S12SD values more in line with the values in Figure 22, although there is way to determine whether these values are "right" without an independent source of UVI data.

Concluding remarks

The amount of solar radiation reaching Earth's surface, including UV radiation, is a complicated combination of direct solar radiation and indirect radiation from the rest of the entire sky hemisphere over a detector. As shown in Figure 23, the atmosphere absorbs a significant wavelength-dependent fraction of that radiation. Clouds and other atmospheric constituents also reflect some of the incoming radiation. The data in Figure 23 are for a clear (cloud-free) sky and a relative air mass of 1.5, which means that the solar zenith angle is about 48°. This graph shows why measuring radiation from the sun, passing through Earth's atmosphere, is a challenge!

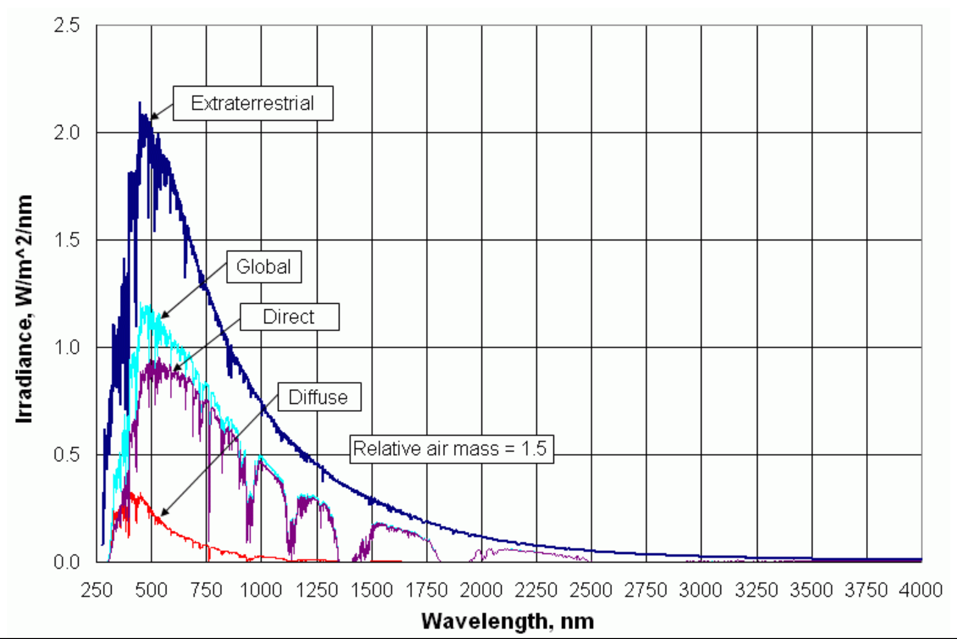


Figure 23. Solar radiation reaching Earth's surface. Caclulated with the SMARTS2 atmospheric radiative transfer model: https://www.nrel.gov/grid/solar-resource/assets/data/smarts295-users-manual-pc.pdf.

As described earlier in this document, UV sensors produce a small photocurrent based on their spectral response. For the LTR-390UV sensor, the process by which a photocurrent is converted to a UVI value is contained in the library and isn't visible at the Arduino sketch level.

For two of these sensors, the GUVA-S12SD and LTR-390UV, the photocurrent is then amplified to produce a usable analog voltage output. Working backwards from the measured output voltage the photocurrent can then be used to calculate power. That power value depends on the module's spectral response, which is why different sensor modules will produce different power values. For example, the ML8511 has some response into the visible spectrum but the GUVA-S12SD doesn't. For this reason, it's difficult to assign any particular significance to a power calculation other than as a relative value based on sky conditions, time of day, and season.

Although it may be possible to calculate UVI values from these inexpensive UV sensors based on vendor or other online sources, there's no reason to expect that any of these devices will produce uniformly accurate results. Using a single detector must assume that the variation of incoming solar power across the range of UV wavelengths used to calculate a UVI according to the erythemal action spectrum is the same under all sky conditions. This is simply an unwarranted assumption; two or more detectors at different wavelengths can reduce uncertainty about that wavelength-dependent distribution.

In fact, single-sensor modules such as the ones tested in this document can be expected to produce different UVI values based on suggested calculations, under *identical* sky conditions. The LTR-390UV has the best documentation comparing its UVI values against a UVI reference meter (Figure 12). For all these sensors, UVI varies linearly with photocurrent, so to the extent this is true, "calibration" for the GUVA-S12SD and ML8511 sensors could involve just changing one or two numerical values. In principle, this can be done if a reliable, independently calibrated UVI source is available. But any correction that's made to force agreement under one sky condition almost certainly will not be the same under all other sky conditions.

Commercial UVI meters with reliable calibrations cost several hundred dollars. For projects that require accurate UVI values, the cost of using such meters to calibrate inexpensive Arduino-compatible UV sensor modules, or even to replace microcontroller-based systems, may be justified. UVI calculations for GUVA-S12SD and ML8511 sensors based on online sources have been included in this document, but for casual measurements it's probably better to consider power data from these devices as useful only for monitoring relative UV values under a variety of time-of-year, time-of-day, sun, and sky conditions.

What's surprising about these inexpensive UV modules is that their sensor cosine response is better than expected – certainly good enough to track the diurnal variability in UV radiation. In principle it might be possible to place a diffuser over the sensor to further improve the cosine response, but based on the data presented in this document, doing this would not be worth the effort.

Finally, note that because these UV detectors are mounted "bare" on their module housings, it's important to avoid touching them because the oil from your fingers could alter their UV response.

In addition to the effects on human health, UV radiation is also important for other areas of environmental monitoring. Along riparian (stream) corridors, for example, UV radiation levels affect benthic (underwater) communities. (See, *e.g.*, https://scholar.google.com/scholar_url?url=https://era.library.ualberta.ca/items/056c2b23-c3a8-4cf3-b613-48bb5bef7f0f/download/909ca959-ef2b-48fe-a09f-461f35663c3a&hl=en&sa=X&ei=brhQYer8KoTjmQGq96rIBQ&scisig=AAGBfm2VC2bVwNENV_bB6lhrbY1-oHNHdQ&oi=scholarr)

Sources

Here are some sources for items used in this project. Some of these items are available from multiple online sources in addition to the ones given here. Prices are, of course, subject to change. (During the time this document was being written, prices on some of these components rose sharply due to global computer chip shortages.)

For data logging shields, there are many sources of SD and microSD cards. There's no reason to get cards with the largest storage capacity. Even inexpensive 1 or 2 GB SD cards, which are currently no longer easy to get (16 GB cards now appear to be a bottom-of-the line "standard" size), will hold a LOT of data for the projects described in this document.

Some of the devices cited in this document include SparkFun Qwiic or STEMMA QT support, for which cables for "plug and play" connections must be purchased separately. I personally still prefer soldered headers for using devices with breadboards.

Arduino Nano (compatibles), allelectronics.com, ARD-20, $9; amazon.com various sources <$9.

Arduino Nano data logging shield, amazon.com various sources, <$13 (CR1220 battery usually not included and microSD card not included).

Arduino UNO (compatibles), allelectronics.com ARD-21, $21; amazon.com, various sources, ~$14. (USB-A/B cable usually included.)

Arduino UNO data logging shield, amazon.com, various sources, < $10. (CR1220 battery usually not included and SD card not included).
GUVA-S12SD, adafruit,com ID 1918 and US distributors, $6.50.
IZOKEE 0.96" OLED, amazon.com, ~$14 for three.
LTR-390UV, adafruit.com, ID 4831, $5. (Use Levkov software library as described in this document.)
Mini breadboards, allelectronics.com, PB-470, $5 for five.
ML8511, amazon.com, various sources and prices, <$8.
Sealed lead-acid battery, 12 V 7.5 Ah, allelectronics.com GC-1270, $27.
(A smaller battery may also work for this project, depending on weather and season.)
Solar controller, dfrobot.com and US distributors DFR0580, $28.50.
Solar panel, 12 V/30 W, from various online sources, ~$40-$60.
(A smaller 12 V panel may also work for this project, depending on weather and season.)
TPL5110 on/off timer, Adafruit.com ID 3435 and US distributors, $5.
Voltage regulator, pololu.com, S7V7F5 adjustable step/up/step-down, $10;
D24V5F5 5 V step-down, $5.