Number 11

INTELLIGENT OPTO SENSOR

# DESIGNER'S NOTEBOOK

# Color Classification with the TCS230

## Identifying and Sorting Colors by Hue

*Contributed by Jack Berlien*
*3/4/04*

**Introduction**
A common requirement in the field of color sensing is that of color identification, or sorting of objects by color. Typically this type of application is simpler than a general-purpose color measurement application, since all we are interested in is identifying which of a predefined list of categories a color belongs. The TCS230 is an RGB color sensor capable of making high-resolution color measurements using the three values obtained from its red, green, and blue sensors. For this application we will use the TCS230 to perform what *color classification*, or the matching of an unknown color to one of a set of known colors. This technique can be used in applications such as LED sorting and testing, industrial sorting and identification, process control in labeling and printing machines, etc.

In this two-part application note we will examine two variations on the theme of color classification. In the first example, Part 1 of the application note, we will classify an unknown color, in this case an LED, into one of several pre-defined hues. In the second example, Part 2, we will determine the best match of an unknown color to one of several previously sampled colors.

**Classification by Hue**
When we think of a color, we typically think in terms of one of the general color names such as red, yellow, green or blue – roughly speaking, *hue* (to be defined later). A common task in color sensing is to identify an unknown color as falling into one of these general categories. This is relatively easy for a person with normal color vision to do, given that we have a vocabulary of color names that conjure reasonably consistent perceptions among different individuals. In fact, eleven basic color names have been identified: white, gray, black, red, yellow, green, blue, orange, purple, pink, and brown. Most or all colors can be described in terms of variations and combinations of these colors.

In order to quantify a color, however, we need to use one of the several color specification systems that have been developed for this purpose. There are many systems in use today, each having its own advantages in different applications and industries. For our purpose, a convenient system to use is hue, saturation, luminance (or value).

**An Overview of Color Specification**
Due to the fact that human color vision is accomplished in part by three different types of cone cells in the retina, it follows that three values are necessary and sufficient to define any color (for more information on color vision and perception, please refer to "Basics of Light and Color" available on the TAOS website). These three values can be thought of as coordinates of a point in three-dimensional space, giving rise to the concept of *color space*. Hue, saturation, luminance (HSL) is one such color coordinate system, or color space, and is convenient for use in this application for a couple of reasons. First, the TCS230 outputs color information in terms of red, green, and blue (RGB) values, and it is a fairly simple task to convert RGB values to HSL values. Second, once we have HSL values, it is a simple matter to examine the hue (H) value to determine the hue of the unknown color.

**HSL Color Coordinate System**
As previously mentioned, the three photo-sensing elements in the TCS230 provide such a set of three signals in the form of R, G, and B. The three elements have spectral responses weighted in the red, green, and blue portions of the spectrum, respectively. While useful in performing color comparisons, monitoring color consistency, or performing color matching using a lookup table, these values are not very useful from a human-readable standpoint, as hue is not readily apparent upon inspection of a set of arbitrary RGB values.

These RGB values can be converted to HSL, which is more intuitive for interpretation by a user. Hue is associated with the dominant wavelength of a color, and is described by standard color names such as red, yellow, green, cyan, blue, and magenta. Saturation describes the degree of colorfulness; a color becomes less saturated as it becomes more gray or white. Luminance describes the brightness of a

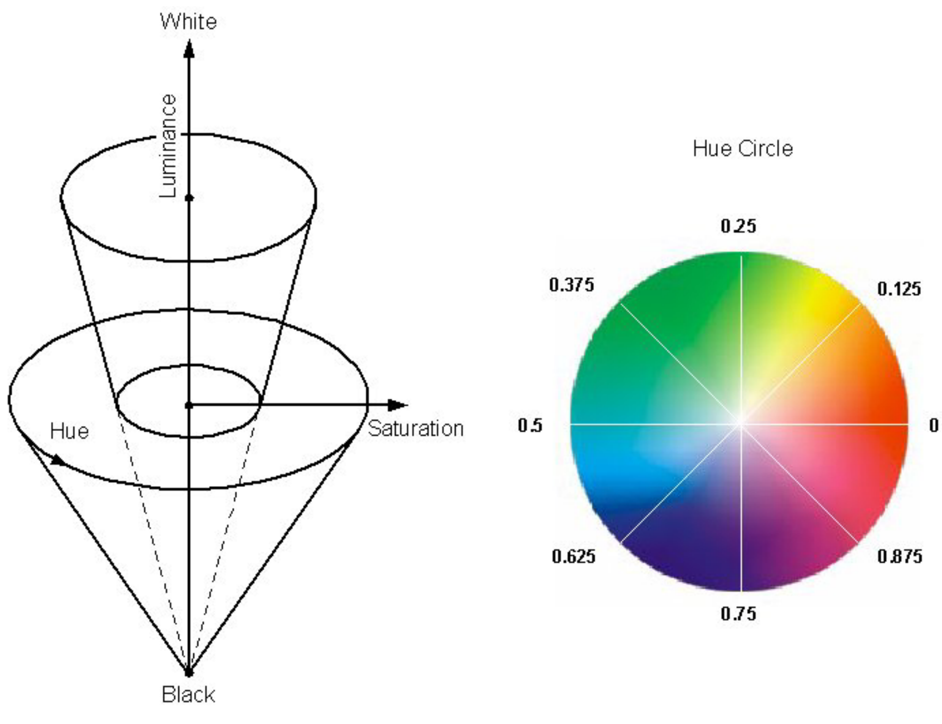


**Figure 1 - HSL Diagram with Hue Circle**

color; as luminance decreases, a color of a given hue becomes darker or more black. HSL can be represented as a conical diagram in which hue is represented by a value ranging from 0 to 1, corresponding to an angle from 0 to 360 around the perimeter of the cone. Note that actual hue values "wrap around" from 0.999 to 0.000. Saturation is represented as the diameter of the cone, and Luminance is the height of the cone (Figure 1). Also shown in Figure 1 is a color representation of the hue circle obtained by taking a slice through the cone in the HSL diagram. Note that the colors become less saturated toward the center of the circle. The hue circle is shown at maximum luminance.

An algorithm for converting RGB to HSL is implemented below in BASIC code (see Appendix A for C code version):

```
Public Sub ToHSL(Red as Single, Green as Single, Blue as Single)
    'Convert RGB (in range 0 to 1) to HSL (in range 0 to 1)
    Dim fmin As Single, fmax As Single
    fmax = Max(Max(Red, Green), Blue)
    fmin = Min(Min(Red, Green), Blue)

    Luminance = fmax

    If fmax > 0# Then
            Saturation = (fmax – fmin) / fmax
    Else
            Saturation = 0
    End If

    If Saturation = 0# Then
            Hue = 0#
    Else
            If Red = fmax Then
                    Hue = (Green – Blue) / (fmax – fmin)
            ElseIf Green = fmax Then
                    Hue = 2# + (Blue – Red) / (fmax – fmin)
            Else
                    Hue = 4# + (Red – Green) / (fmax – fmin)
            End If
            Hue = Hue / 6#

            If Hue < 0# Then Hue = Hue + 1#
    End If
End Sub

Private Function Min (x As Single, y As Single) As Single
    If x < y Then
            Min = x
    Else
            Min = y
    End If
End Function

Private Function Max (x As Single, y As Single) As Single
    If x > y Then
            Max = x
    Else
            Max = y
    End If
End Function
```

**Classification By Hue – LED Testing**
A wide variety of colors of LEDs are available today and are widely used on printed circuit (PC) boards and in systems as a means of conveying some type of information about the operation of the system to the user. In these and many other applications, it is critical that the proper color of LED be installed in its intended location. Complicating this requirement is the fact that most LEDs are not marked or symbolized, and many are molded in clear plastic, giving no visual indication of the emitted color. Thus it is relatively easy and common to have the incorrect LED installed in a given location. Therefore testing or verification of the LED color has become a requirement in many PC board final test operations. The TCS230 provides an economical and easy-to-implement means of checking both the intensity and color of an LED. Relative intensity of the LED can be measured directly using the clear channel of the TCS230. Color can be determined using the HSL algorithm presented above.

For this example, several common and not-so-common LEDs were selected: red, amber, green, aqua, blue, ultraviolet, and purple. For measurement and data processing, the TCS230EVM (available from TAOS) was used. The EVM consists of a lens module with white LED light source, a controller board, and RS-232 interface to a PC-based host software application. The software application converts the RGB values from the TCS230 to HSL values and displays them on the screen in real-time. For this experiment, the lens was removed exposing the TCS230, and the light source was disabled.

The LED was positioned about one inch from the TCS230, shining directly on the sensor. Hue values were recorded for each LED, and are shown plotted in Figure 2. Note that in the HSL system, hue is independent of parameters such as LED position and (for the most part) current through the LED.

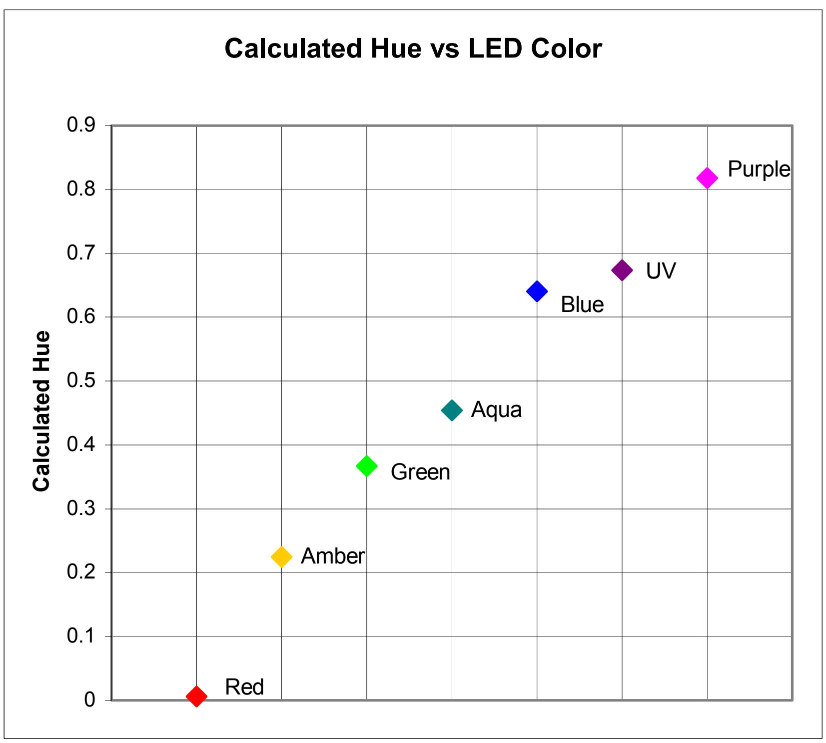


**Figure 2 - Calculated Hue for Various LEDs**

Table 1 illustrates the constancy of hue as the current through a red LED is varied. Therefore LED hue, or color, can be determined regardless of intensity at which the LED is operating. An LED, by nature, is a very saturated light source as verified by the saturation values obtained, which were above 0.95 for all the LEDs plotted in Figure 2 (except for purple – more on this later). This is because an LED has a narrow spectral power distribution, that is, the emitted light is concentrated around a narrow range of wavelengths. A color can be made less saturated by mixing in other wavelengths, or colors, of light. True white light, which is defined as an equal mixture of all wavelengths, has a saturation of zero.

A unique case in LED testing is the white LED. Many white LEDs today are manufactured by applying a phosphor over a blue LED chip. The phosphor absorbs the blue light from the LED and re-emits light over a broad range of longer wavelengths, creating a source that appears white. As suggested by the above explanation of saturation, the white LED has a low saturation value, 0.127, since it is a mixture of many wavelengths of light. The fact that the LED appears to have a slightly bluish tinge is confirmed by the hue value, 0.58, placing it in the blue region of the hue circle. In order to identify a white LED in testing, saturation would have to be calculated along with hue.

Another unusual case was the purple LED. It had a slightly lower saturation value than the standard LEDs, about 0.76. As it turns out, this type of LED is also produced by placing a phosphor over a blue LED, as with the white LED. This explains the lower saturation value, since the color is produced by a mixture of wavelengths.

**Table 1 - Hue and Luminance vs LED Current**

| LED Forward Current (mA) | Hue | Luminance |
|---|---|---|
| 0.5 | 0.003 | 0.422 |
| 1 | 0.003 | 0.523 |
| 1.5 | 0.005 | 0.625 |
| 2 | 0.003 | 0.711 |
| 2.5 | 0.005 | 0.785 |
| 3 | 0.003 | 0.855 |
| 3.5 | 0.003 | 0.93 |

There are a couple of ways to set up the system to test and identify LEDs. If the LEDs to be identified differ widely in color from each other, such as red, green, and yellow it may be sufficient to configure a set of fixed ranges to sort the hue values, as illustrated in the following BASIC code example:

```
Select Case Hue
Case Is >= 0.875, Is <= 0.1
    LED_Color = "Red"
Case 0.1 To 0.25
    LED_Color = "Yellow"
Case 0.25 To 0.5
    LED_Color = "Green"
Case Else
    LED_Color = "Other"
End Select
```

If there are several LEDs present that are similar in color, or if several different colors are present, it may be better to "teach" the system the correct values for each LED. For example, representative samples of each LED would be measured, and those hue values stored in non-volatile memory. Then each unknown LED would be sampled, and its value compared with the known values in memory and an error value calculated. Then the best match is the one with the lowest error value. A function is shown below that will calculate the best match using five stored values. The fact that a hue value of 0.99 is very close to a hue value of 0, that is hue values wrap around from 0.999 to 0, is handled by the first two IF statements.

```
Public Function LED_Color(Hue As Single) As String

        Dim vData(1 To 2, 1 To 5) As Variant
        'Set up the data for the example, by increasing hue value
        'Note – this array could have more or fewer elements, depending on required
        'hue resolution
        vData(1, 1) = 0.006
        vData(2, 1) = "Red"
        vData(1, 2) = 0.225
        vData(2, 2) = "Amber"
        vData(1, 3) = 0.367
        vData(2, 3) = "Green"
        vData(1, 4) = 0.454
        vData(2, 4) = "Aqua"
        vData(1, 5) = 0.64
        vData(2, 5) = "Blue"

        Dim MinError As Single, ErrorVal As Single, KnownHue As Single
        Dim i As Integer

        MinError = 1                  'Initialize error to a high value
        For i = 1 To 5                'Change stop value to match number of hue values
            KnownHue = vData(1, i)
            ErrorVal = Hue – KnownHue
            'Multiple comparison criteria are needed to account for the fact that hue values
            '"wrap around" from 0.999 to 0
            If ErrorVal < 0 Then ErrorVal = ErrorVal + 1
            If ErrorVal > 0.5 Then ErrorVal = Abs(ErrorVal - 1)
            If ErrorVal < MinError Then
                MinError = ErrorVal
                LED_Color = vData(2, i)
            End If
        Next

End Function
```

## Conclusion

The TCS230 can be used, along with some simple processing of RGB values, to reliably determine the color of LED and other colored sources by calculating the hue value, and then sorting according to hue. For just a few colors widely differing in hue, absolute hue sort ranges can be determined at design time and used reliably. For systems that must distinguish from several hues, or among similar hues, possible known values should be sampled (measured), and stored by the system. Then during test, the best match of an unknown sample can be determined among the known possible hues. The hue method only works well, however, for highly saturated color sources.

# Appendix A

## C Code for RGB to HSL Conversion

```
#define MAX(a, b)  (((a) > (b)) ? (a) : (b))
#define MIN(a, b)  (((a) < (b)) ? (a) : (b))

// convert RGB (in range 0 to 1) to HSL (in range 0 to 1)
void ToHSL(const float red, const float green, const float blue, float& hue, float&
saturation, float& luminance)
{
        float fmax, fmin;
        fmax = MAX(MAX(red, green), blue);
        fmin = MIN(MIN(red, green), blue);

        luminance = fmax;

        if (fmax > 0)
                saturation = (fmax - fmin) / fmax;
        else
                saturation = 0;

        if (saturation == 0)
                hue = 0;
        else
        {
                if (fmax == red)
                        hue = (green - blue) / (fmax - fmin);
                else if (fmax == green)
                        hue = 2 + (blue - red) / (fmax - fmin);
                else
                        hue = 4 + (red - green) / (fmax - fmin);

                hue = hue / 6;
                if (hue < 0) hue += 1;
        }
}
```