

8.1 センサ実験3：カラーセンサの基礎

8.1.1 はじめに

【演習目的】

カラーセンサを使用し、色の表現方法やセンシング技術を理解しながら演習を進める。
最終的には、カラーセンサについて学んだ知識を統合的に応用する。

特に、以下の項目の修得を授業目標とする。

- (1) カラーセンシング技術を理解し、応用できる。
- (2) カラーセンサのキャリブレーションを理解し、応用できる。
- (3) I²Cによる通信方法を理解し、状況に合わせて通信プログラムの設定を変更できる。
- (4) 色空間の変換方法を理解し、色情報を Processing（ソフトウェア）で可視化できる。

8.1.2 色と表色系

【色の表現方法】

(1) 色が見えるしくみ：人の目に見える光の範囲の波長を「可視光線」と呼ぶ（図 8.1.1）。光自体に色はついておらず、人の目に光が届くことで色を感じることができる。つまり、目の視細胞が光の強弱と波長の違いを刺激として感じることで、脳は色の識別を行うことができる。従って、動物と人で見えている色は、目の視細胞の違いから当然異なる。人の目の視細胞は感光性があり、明るい場所で赤（R）・緑（G）・青（B）の刺激に反応する錐体細胞と、暗い場所で明暗を知覚する杆体細胞の 2 種類に分類できる。例えば、リンゴが赤く見える様に、実際の物に色がついて見えるのは、物体に投射された光が反射され、目に届くためである。逆に、物体に吸収された光の色は目に届くことはないため、見ることができない（図 8.1.2）。

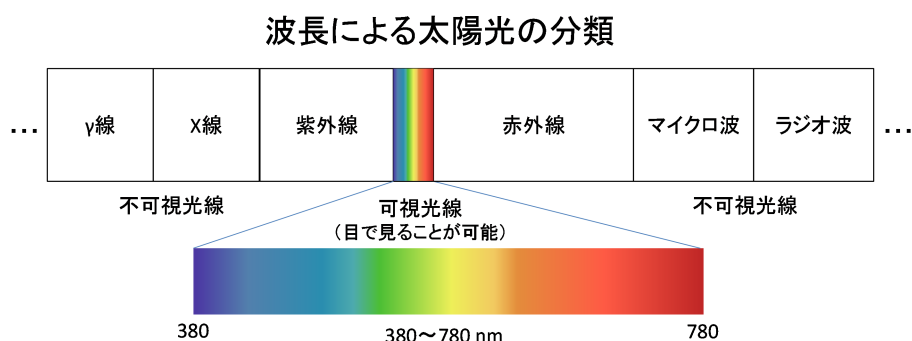
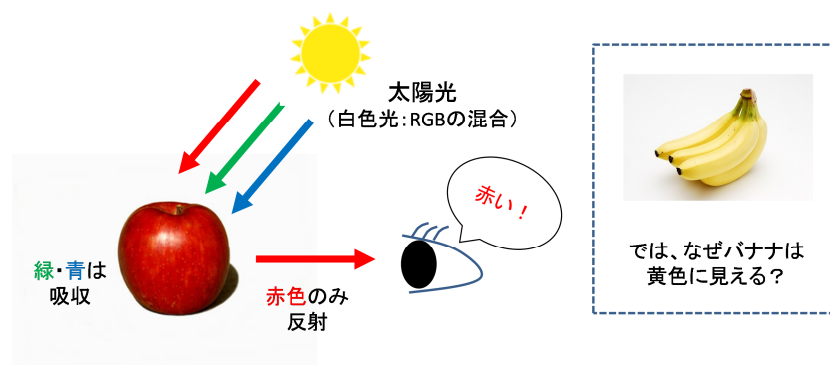


図 8.1.1 可視光線の波長領域



(2) 光の 3 原色：RGB は、光の 3 原色と呼ばれる。赤 (R) と緑 (G) の光を重ねるとイエロー、赤 (R) と青 (B) ではマゼンタ、青 (B) と緑 (G) ではシアンになり、3 色を全て重ねると白色光になる (図 8.1.3 左)。この様に、光の色を加えて表すことを「加法混色」という。RGB の光の強度を調整することで、様々な色を表現できる。例えば、テレビやパソコンの画面も RGB の光を組み合わせることで作られている。

(3) 色の 3 原色：色の 3 原色は、シアン (C)・マゼンタ (M)・イエロー (Y) から構成される (図 8.1.3 右)。写真やカラー印刷での様々な色は、この CMY の 3 原色をいろいろな割合で混合することで表現できる。また、CMY を 100% の濃度で混色すると黒になる。つまり、色を加えていくと暗くなる (光を減らす) ので、このような色の表し方を「減法混色」という。但し、カラー印刷では、CMY の混色による黒は鮮やかにならないため、黒を加えた CMYK (K は Black) の 4 色が基本色となる。

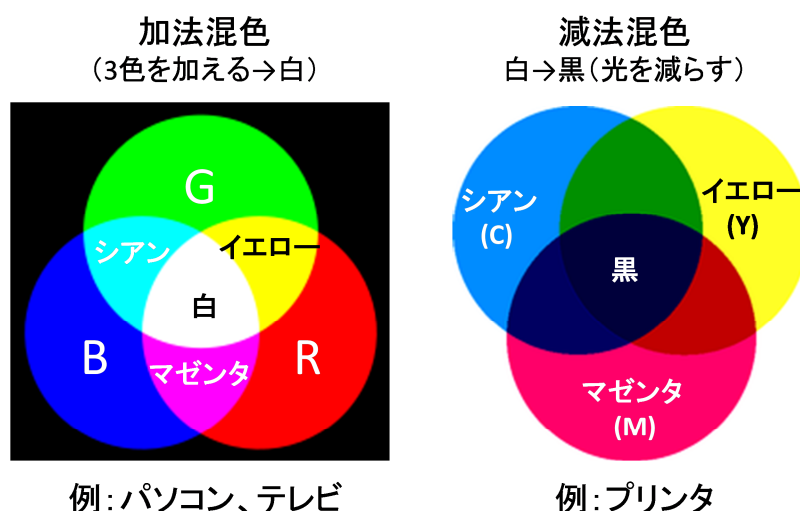


図 8.1.3 色光の混合による色の表現方法 (加法混色と減法混色)

【表色系】

言葉だけでは共通の色を正しく表現したり、人に上手く伝えることができない。しかし、目に見えている色を数値化してデータ表示できれば、体系的に色を分類しながら、色の情報を詳細に伝えられる。従って、全ての人に共通な色の分類方法として、様々な表色系が定義されている。例えば、①混色系の表色には、オストワルト表色系、RGB 表色系、XYZ 表色系 (国際照明委員会：CIE による規格化) 等がある。また、人の感覚に近い色の分類方法である②顕色系の表色には、マンセル表色系、NCS (ナチュラルカラーシステム) 表色系等がある。

(1) XYZ 表色系：色を客観的かつ定量的に表示する方法として、3 色 (RGB) 表色系は有効である。この表色系では、色光の 3 色を定めており、その混色量の数値により種々の色を表現できる。しかし、現実の色光 R (赤)・G (緑)・B (青) を原色として扱うと、人の色に対する感度を表す等色関数 (図 8.1.4 左) に負の値が生じて扱いにくくなる。従って、等色関数の負の値をなくす様に、数学的に変換したものとして XYZ 表色系がある (図 8.1.4 中央)。XYZ 表色系の 3 原色 (XYZ) を原刺激といい、原刺激の量を 3 刺激値と呼ぶ。全ての色は、この 3 刺激値により数値で表現できる。

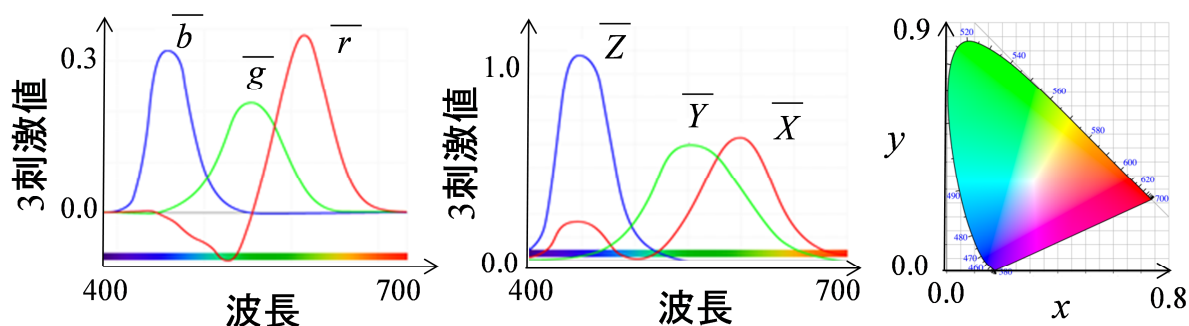


図 8.1.4 RGB 表色系（左）と XYZ 表色系（中央）の等色関数，xy 色度図（右）

(2) xy 色度図：XYZ 表色系の 3 刺激値 X, Y, Z の混色比である x, y, z は，下記の式で表すことができる．

$$x = \frac{X}{X+Y+Z}, \quad y = \frac{Y}{X+Y+Z}, \quad z = \frac{Z}{X+Y+Z}$$

色は混色比で表されるため， $x+y+z=1$ となる． x, y の値が分かれば z の値も分かるため， x と y だけを用いて色度を示す座標を表示したものを「xy 色度図」と呼ぶ．この xy 色度図（図 8.1.4 右）の座表面上に示されている色は色相と彩度のみを示しており，明るさに対応するものはない．xy 色度図の曲線部分（釣鐘状の形）はスペクトル軌跡と呼ばれ，赤から青紫にかけての単色光が順に並んでいる．図の下の方の直線部分は，両端の赤と青紫を混色してできる紫から赤紫の色が並んでおり，純紫軌跡と呼ばれる．実在色は，この座標の範囲内で表すことができる．しかし，RGB 値に基づくカラーセンサやモニターでは，この座標系の全ての色（実在する全色）を表せない．色度座標は混色比のため，人が直接見る色を表現するものではないが，混色比からどのような色かをイメージしやすくなるという利点がある．本授業では，カラーセンサを用いて，実際に表現できる色座標の範囲を演習する．

RGB 空間から XYZ 空間への変換式は，RGB 成分を (R, G, B) ，この点の XYZ 空間での座標を (X, Y, Z) ，変換行列を A (3×3 行列) とすると，次式で表される．
$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = A \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

【色再現の原理（カラーマネジメント）】

実際に目で見た景色の色が印刷物やモニターを通して見ると，異なった色として見えたりした経験はないだろうか．また，映像や写真の出力機器に依存して，同じ色でも異なって見えてしまうことがある．従って，画像をデジタル情報化して紙印刷する場合，できるだけ人が見ているままの色と形を再現することが望まれる．しかし，色の情報を取り込むデジタルカメラやスキャナー・色の情報を映し出すモニター・印刷用プリンターでは，色の再現方式と用いられるインク等の色材は異なってしまう．このような色の表現方法の違いを調整するため，複数の使用するデバイス間でカラーマネジメント（色の再現方法の統一）を行う必要がある．

8.1.3 カラーセンサと I²C 通信

【カラーセンシング技術】

RGB の各センサにより色の強度を測定できる．演習で使用するカラーセンサ（SEN60256P）では，緻密な色の測定や判定・識別ができる様に，Red・Green・Blue・

Clear の光検出を行う。Clear 値は、フィルタなしで光をそのまま検知した場合を示す。また、SYNC（同期）入力により、カラーセンシングを外部イベントと同調させることができる。例えば、LED 固定照明では、SYNC 入力により色の読み取りを PWM 信号や LED の照明と同調させることで、正確な色測定が可能になる。

RGB 色空間では、RGB の各強度が 100% のとき、混色の結果は白色になる。しかし、白色をどのような色あいの白で表現するか（白色点：ホワイトポイント）を規定する必要がある。この白色点は、物質を高温に熱したときに放出される光の色に対応づけた色温度で表すことができる。

【カラーセンサの仕様】

演習で使用するカラーセンサ（SEN60256P）は、TCS3414CS という IC チップを搭載しており、 I^2C 通信により色情報の入出力を行うことができる。TCS3414CS には、 8×2 のフォトダイオードアレイがあり、16 ビットの AD 変換により周囲の光の色度や物体の色を検出できる。計 16 個のフォトダイオードのうち 4 個ずつが、Red・Green・Blue フィルタに対応しており、残りの 4 個がフィルタなし（Clear）となる。同期式入力ピンを使用することで、外部光源から同期した変換制御を行うこともできる。また、閾値設定による割込み機能が使用できる。この IC を駆動する電圧は、5 V（3.3～6 V の範囲）である。TCS3414CS のピン配置や接続方法等の詳細については、製造元のデータシートを参照せよ。

<演習 8.1.1> カラーセンサのセットアップ

カラーセンサによる色計測を行うため、図 8.1.5 に示す様に、カラーセンサと Arduino を接続する。カラーセンサ（SEN60256P）のピン配列は、左から順番に、GND・5V・SDA（A4 ピン）・SCL（A5 ピン）となっている。

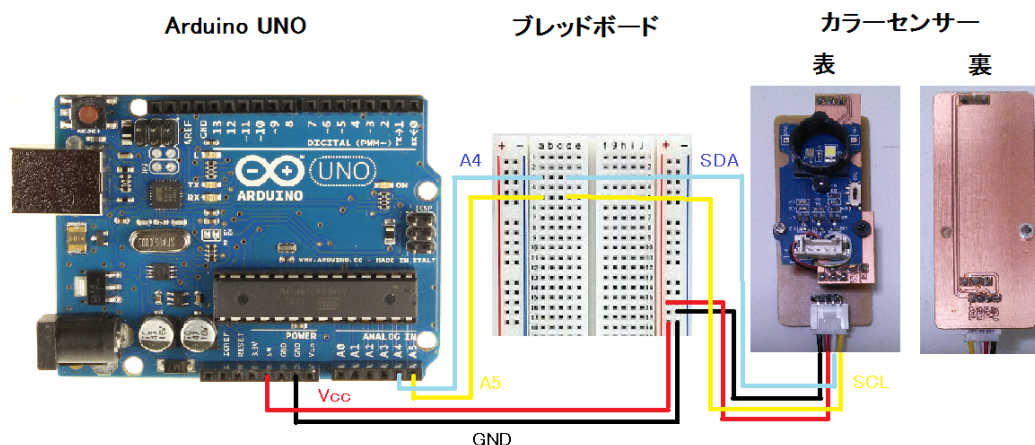


図 8.1.5 カラーセンサと Arduino の接続

【 I^2C 通信】

I^2C （Inter-Integrated Circuit）通信は、フィリップス社が開発した周辺デバイスとシリアル通信を実現するための方式（図 8.1.6）である。 I^2C 通信では、マスターとスレーブを分類し、マスター側が主な制御を行う。また、1 つのマスターで、複数のスレーブデバイスと通信を行うことができる。 I^2C 通信では、抵抗でプルアップされた双方向の 2 本の信号線のみを使用する。この 2 本の信号線は、シリアルクロック（SCL）とシリアルデータ（SDA）に分けられる。SCL により一定周期でのデータ取得が可能となり、SDA で実際のデータの

送受信が行われる。演習で使用する Arduino UNO では、SDA は A4 ピン，SCL は A5 ピン にそれぞれ対応している（図 8.1.5 参照）。

I²C 通信では、スレーブのアドレス指定方法として、7 ビットアドレスや 8 ビットアドレスがある。ただし、演習で使用する Arduino の I²C 通信用の Wire 関数は、7 ビットアドレスのみに対応している。また、マスターとスレーブの間で、常に、ACK（NACK）がやり取りされることで、データ保証がなされている。

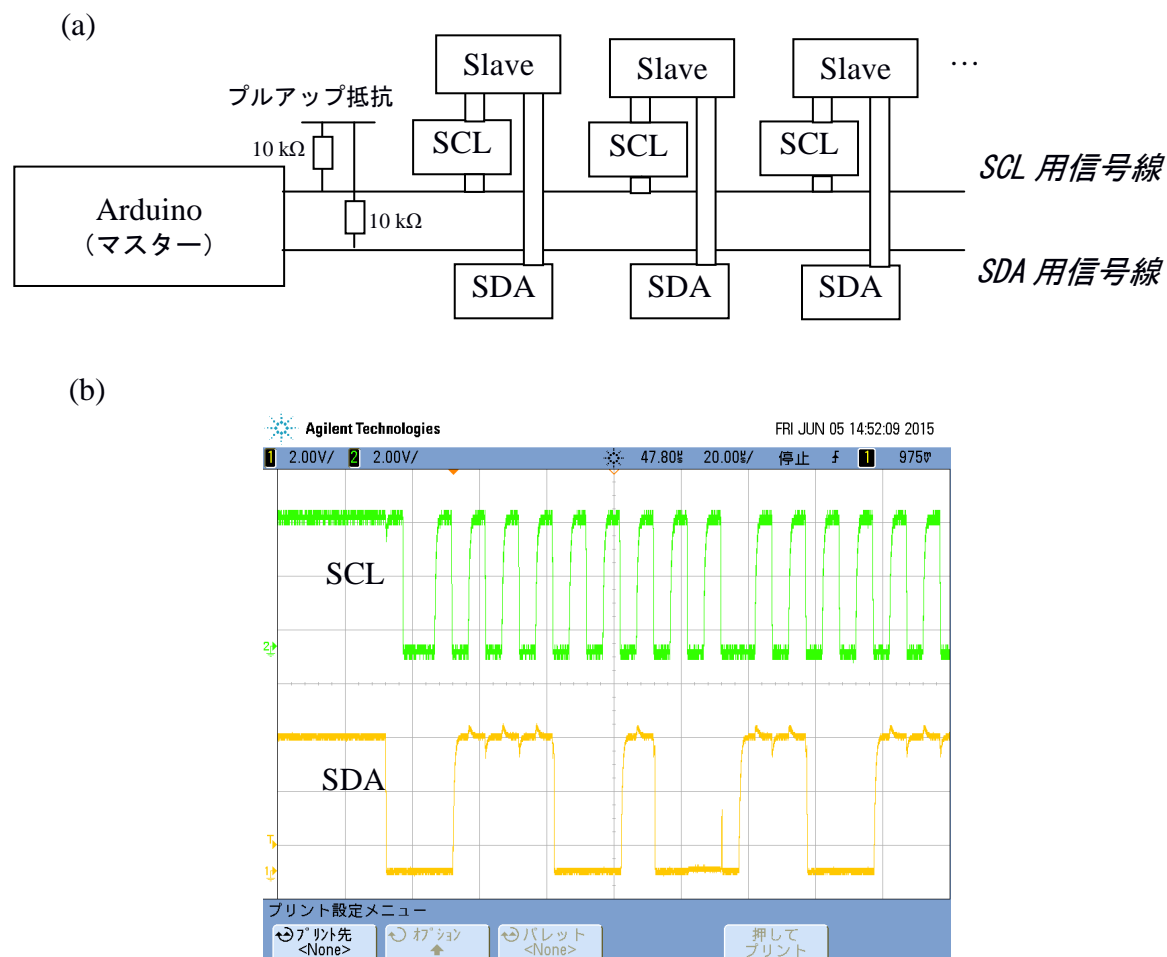


図 8.1.6 (a) I²C 通信での信号線（SCL と SDA）の配線例
(b) SCL（上段）と SDA（下段）の計測波形

I²C 通信を利用してカラーセンサの情報を取り込む演習を行う。具体的には、Arduino をマスターに設定し、カラーセンサをスレーブに設定することで I²C 通信を行う。以下に、I²C 通信の手順を簡単にまとめる。使用する関数の引数等は、巻末の付録を参照せよ。

【I ² C 通信の手順】
(1) I ² C 通信の開始
//マスター設定の初期化 Wire.begin()
(2) Arduino（マスター）からの書き込み要求

//スレーブにスタートコンディションの発行とアドレス送信 Wire.beginTransmission(アドレス)
(3) カラーセンサ（スレーブ）とのデータの送受信
//データ送信 Wire.write(送信データ)
//データ受信 Wire.read(受信データ)
(4) I ² C 通信の終了
//ストップコンディションの発行 Wire.endTransmission()

<演習 8.1.2> I²C 通信によるセンサ情報（RGB 値）の読み込み

I²C 通信では、Arduino（マスター）でシリアル通信や I²C バスに接続する初期設定を行った後、指定した一定間隔で、カラーセンサ（スレーブ）からの情報を読み込むことができる。ここで、カラーセンサから色情報が送信されてくる順番は、Green（下位 8 bit・上位 8 bit）、Red（下位 8 bit・上位 8 bit）、Blue（下位 8 bit・上位 8 bit）、Clear の順に固定されていることに注意する。

最初に、Arduino で I²C 通信を実行するための Wire 関数の使い方について理解する。下記のサンプルスケッチを実行し、シリアルモニタにより、RGB 値が取込めることを確認せよ。その際、種々の色（測定対象が光源でない場合、カラーセンサの LED ライトを ON にすることに注意）で、RGB 値がどの様に変化しているかを検討せよ。

・ S8.1.1 Arduino と I²C デバイスの送受信用サンプルスケッチ（Arduino program）

```
#include <Wire.h>
#define COLOR_SENSOR_ADDR 0x39
#define REG_BLOCK_READ 0xCF
unsigned int readingdata[20];
unsigned int i, green, red, blue;
//double X, Y, Z, x, y, z;                                //演習 8.1.6 で使用

void setup(){
  Serial.begin(9600);                                       //シリアル通信の初期化
  Wire.begin();                                             //I2C バスに接続
  Wire.beginTransmission(COLOR_SENSOR_ADDR);              //カラーセンサの AD 変換開始
  Wire.write(0x80);                                         //REG_CTL
  Wire.write(0x03);                                         //CTL_DAT_INITIATE
  Wire.endTransmission();
}

void loop(){
  readRGB();
  //calculateCoordinate();                                  //演習 8.1.6 で使用
  delay(500);
}

void readRGB(){
  Wire.beginTransmission(COLOR_SENSOR_ADDR);
  Wire.write(REG_BLOCK_READ);
  Wire.endTransmission();                                  //送信完了
  Wire.beginTransmission(COLOR_SENSOR_ADDR);              //送受信開始
  Wire.requestFrom(COLOR_SENSOR_ADDR, 8);                 //カラーセンサにデータ要求
  delay(500);
  if(Wire.available() >= 8){                                //8byte 以上受信したとき
```

```

for(i =0; i < 8; i++){
    readingdata[i]=Wire.read();           //データの受信
}
}
green = readingdata[1]*256 + readingdata[0]; //受信データの配列と色情報の順番固定
red   = readingdata[3]*256 + readingdata[2]; //2byte の RGB 値にデータ復元
blue  = readingdata[5]*256 + readingdata[4];
//Serial.println("RGB values = ");
Serial.print(red, DEC); Serial.print(","); //DEC: ascii code で 10 進表記
Serial.print(green, DEC); Serial.print(",");
Serial.println(blue, DEC);
}

```

8.1.4 キャリブレーションと色空間の変換

カラーセンサの値を読み取る際には、センシングした値をそのまま表示するのではなく、物理的に意味のある値に校正（キャリブレーション）する必要がある。カラーセンサの場合、図 8.1.7 に示す様に、(A) 発光物体（例：フルカラーLED）そのものからキャリブレーションを実行する方法と、(B) LED 光源を直接対象物に照射し、その反射光からカラーセンサのキャリブレーションを行う方法がある。

※演習で使用するカラーセンサの LED ライトは、(A)のキャリブレーションの場合 OFF, (B)の場合 ON にすることに注意せよ。この LED ライトはカラーセンサに付属しているスイッチにより、ON・OFF を切り替えることができる。

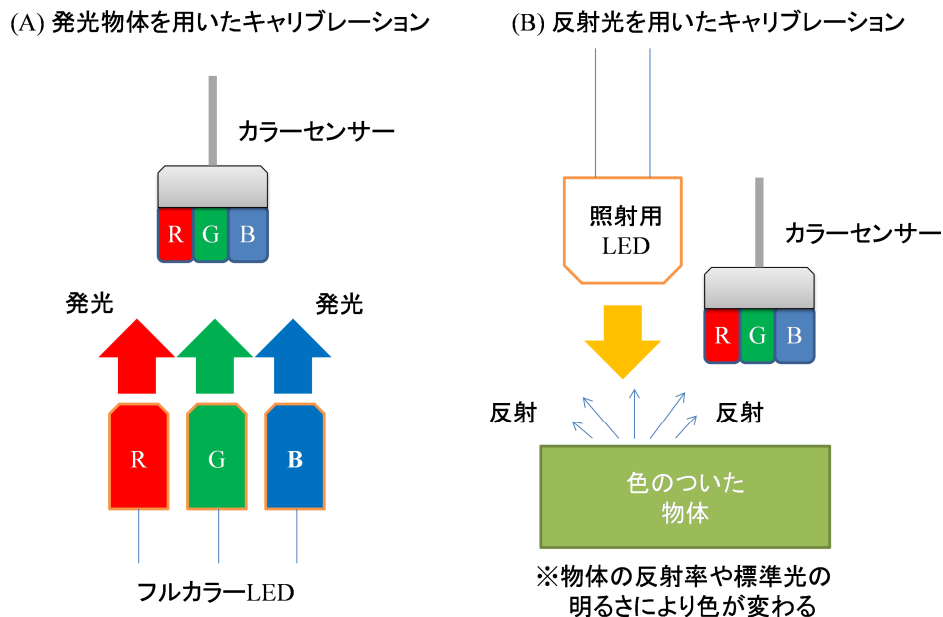


図 8.1.7 カラーセンサのキャリブレーションの方法

<演習 8.1.3> Processing による RGB 値の表示

演習 8.1.2 で取得したカラーセンサの値を Processing により可視化する。取得した RGB 値に応じて、図形の色や大きさを変化させる。カラーパターンは、manaba で配布した画像（color-pattern.jpg）をディスプレイに表示させて利用する。色をセンシングする場合、カラーセンサをディスプレイ画面に近づけすぎない様に注意する。また、測定対象が光源となるため、カラーセンサの LED ライトは OFF にする。センサの Red の値のみを可視化する

るサンプルスケッチを下記に掲載する。Green・Blue の値についても、同様に可視化できる様にプログラムを作成せよ。その後、可視化した図形の色から、キャリブレーションの有無による違いを検討せよ。この演習でのキャリブレーションには、RGB の各センサの最大値を利用する。

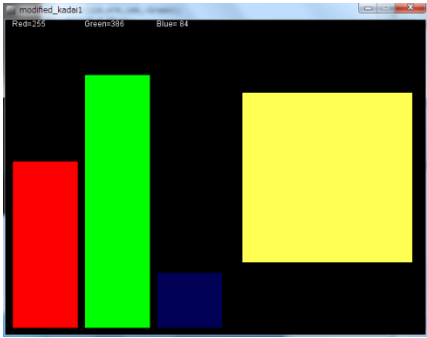
・ S8.1.2 図形描画用サンプルスケッチ (Processing program)

```

import processing.serial.*;
Serial myPort;
float Red, Green, Blue;
void setup() {
  size(640, 480);
  myPort = new Serial(this, "/dev/ttyACM0", 9600);
  myPort.bufferUntil('\n'); //改行までメッセージ受信
}
void draw() { //受信した値で描画
  background(0);
  fill(Red,0,0); rect(10,470,100,-Red);
  fill(0,Green,0); rect(120,470,100,-Green);
  fill(0,0,Blue); rect(230,470,100,-Blue);
  fill(Red, Green, Blue); rect(360,110,260,260);
  print("R="+Red+", "); print("G="+Green+", "); println("B="+Blue);
  fill(255);
  text("Red=",10,10); text(int(Red),40,10);
  text("Green=",120,10); text(int(Green),160,10);
  text("Blue=",230,10); text(int(Blue),265,10);
}
void serialEvent(Serial myPort) {
  String myString = myPort.readStringUntil('\n'); //シリアルバッファー読み込み
  if (myString != null){
    myString = trim(myString); //空白文字など消去
    float data[] = float(split(myString, ',')); //カンマ区切りで複数の情報を読み込む
    if (data.length > 1){
      // キャリブレーションなし
      Red = data[0];
      Green = ???; //ここに Green の処理を入れる
      Blue = ???; //ここに Blue の処理を入れる
      // キャリブレーションあり
      //Red = map(data[0], 0, 1000, 0, 255); //下線部をシリアルモニタの最大値に変更
      //Green = ???; //ここに Green の処理を入れる
      //Blue = ???; //ここに Blue の処理を入れる
      myPort.clear();
    }
  }
}

```

実行例 (黄色をセンシング)



<演習 8.1.4> フルカラーLED のセットアップ

基本色である RGB をフルカラーLED により再現する。最初にフルカラーLED の配線や抵抗の配置等のセットアップ (図 8.1.8 参照) を行う。フルカラーLED のピン配列は、図 8.1.8 の左から順に、Green (Arduino の 11 番ピン), Blue (10 番ピン), カソード, Red (9 番ピン) の順になっている。既に演習してきた様に、LED の電流が流れる向き (アノード・カソード) は、決まっていることに注意する。また、3 色 (RGB) それぞれの LED に同じ

電流を流すことを考えると，Red の LED は，Green, Blue の LED と比べて低い電圧で済む．これを補正するには，各 LED に対する抵抗値を調整するのが良い．

※抵抗値の計算例：抵抗 (Ω) = (電源電圧 - VF) / (電流 mA / 1000)

VF/IV (20 mA) Red : 2.0 V/2000 mcd, Green : 3.6 V/2500 mcd, Blue : 3.6 V/7000 mcd

このとき，電源電圧を 5 V とすると，各 LED で必要な抵抗は下記の様になる．

$$\left\{ \begin{array}{l} \text{Red} : (5 - 2.0) / 0.02 = 150 \Omega \\ \text{Green} : (5 - 3.6) / ((2000 / 2500) * 0.02) = 240 \Omega \\ \text{Blue} : (5 - 3.6) / ((2000 / 7000) * 0.02) = 91 \Omega \text{ (演習では, } \underline{100 \Omega} \text{ を使用)} \end{array} \right.$$

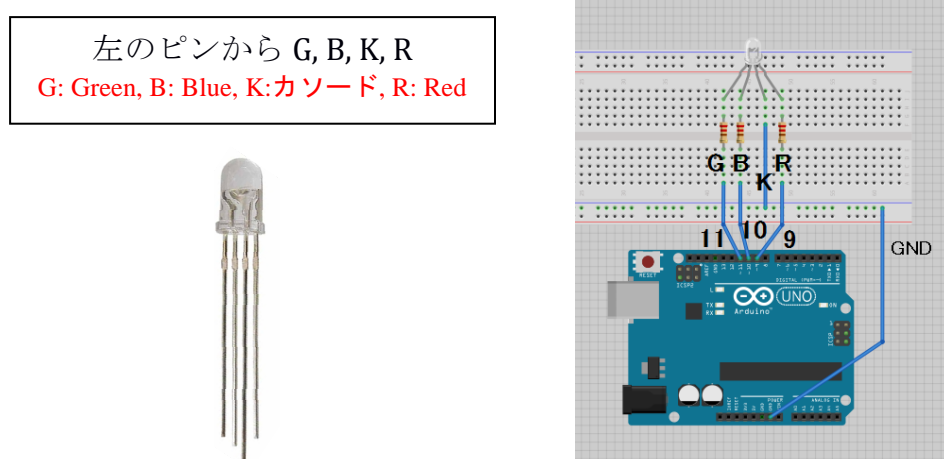


図 8.1.8 フルカラーLED のピン配列と配線図

<演習 8.1.5> フルカラーLED の点灯とカラーセンサ値の読出し

(1) カラーセンサにより，フルカラーLED の Red の値（レジスタ情報）だけを読出し，シリアルモニタで表示するプログラムを実行する．そのため，まずは，フルカラーLED のうち，Red のみを周期的に点滅させるサンプルスケッチを実行せよ．LED の点滅周期は，目視で確認できる様に，`delay()` 関数の引数を各自で調整すること．あまりに早い点滅速度の場合，カラーセンサによる認識が上手くいかないので注意する必要がある．

・ S8.1.3 フルカラーLED 点燈用のサンプルスケッチ (Arduino program)

```
void setup() {
  pinMode(9, OUTPUT); // RED 9 番ピン
  analogWrite(9, 0);   // PWM duty 比
}
void loop() {
  analogWrite(9, 255); // PWM duty 比
  delay(500);          // 点滅周期は各自で調整
  analogWrite(9, 0);
  delay(500);          // 点滅周期は各自で調整
}
```

(2) フルカラーLED の Red の点滅にカラーセンサを近づけることで、センサの Red の値（レジスタ情報）を読み出し、シリアルモニタに表示できる様にせよ。その際、LED の点滅と同期して、カラーセンサの値が変化していることを確認せよ。演習 8.1.2 のカラーセンサの読み出しをするサンプルスケッチに、演習 8.1.5(1)のフルカラーLED を点滅させるプログラムを追加すればよい。

※カラーセンサの LED ライトは OFFにする。

※カラーセンサの値の取得には時間がかかるため、演習 8.1.5(1)で設定したフルカラーLED の点滅周期は各自で調整する（delay()関数で 1 秒程度は必要）。

<演習 8.1.6> RGB 値から xy 値への変換方法

カラーセンサ（TCS3414CS）により検出した RGB の値は、最初に式(1)の変換行列を用いて、CIE の定める XYZ 表色系の値に変換する必要がある。その後、式(2)から xy 色度図の xy 値を計算できる。

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} -0.142 & 1.549 & -0.956 \\ -0.324 & 1.578 & -0.731 \\ -0.682 & 0.770 & 0.563 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad (1)$$

$$\begin{cases} x = X / (X+Y+Z) \\ y = Y / (X+Y+Z) \end{cases} \quad (2)$$

カラーセンサから取得した RGB 値を XYZ 及び xy 値に変換する関数を実行し、シリアルモニタにより結果を確認せよ。演習 8.1.5 で作成したプログラムに下記の calculateCoordinate()関数のサンプルスケッチを追加して、シリアルモニタで表示できるようにすればよい。その際、「演習 8.1.6 で使用」と書かれたコメントアウト部分をプログラムに反映させる。ここで、RGB 値が最大するとき、xy 色度図上の白色点を示すこととなる。次に、各自でカラーセンサから取得する色を変化させ、実際の色がどれくらいの xy 値に対応しているかを検討せよ。

・ S8.1.4 RGB 値から xy 値に変換するサンプルスケッチ（Arduino program）

```
void calculateCoordinate(){
  X=(-0.142)*red+(1.549)*green+(-0.956)*blue;
  Y=(-0.324)*red+(1.578)*green+(-0.731)*blue;
  Z=(-0.682)*red+(0.770)*green+(0.563)*blue;
  x=X/(X+Y+Z); y=Y/(X+Y+Z);
  if( (X>0) && (Y>0) && (Z>0) ){
    //Serial.println("x, y = ");
    Serial.print(x, 2); Serial.print(","); Serial.println(y, 2);
  }
  else Serial.println("Error!");
}
```

【実施課題】

<課題 8.1.1> Processing による色情報の可視化

(1) 演習 8.1.3 で作成したプログラムを元に、Clear 値（明るさ）についても可視化し、その影響について検討（例：横軸に Clear 値、縦軸に RGB 値をとってグラフ化する等）せよ。その際、ディスプレイの明るさ（輝度）自体を変更してみるとよい。演習 8.1.2 で Clear 値を読み込む場合、「`clr = readingdata[7]*256 + readingdata[6];`」となる。

(2) カラーセンサの値を読み取る通信速度についても検討せよ。例えば、どの程度の遅延時間までリアルタイムに通信可能か試してみる。演習 8.1.2 や演習 8.1.5 のサンプルスケッチにある `delay()` 関数（青字で表記した部分）の引数を変更してみるとよい。

<課題 8.1.2> RGB 値の呼出しと混色の表現

演習 8.1.5 のサンプルスケッチをもとに、Green, Blue を加えた RGB（3 色）の点滅プログラムを完成させよ。その際、Red の値とともに、Green, Blue の値の変化もシリアルモニタにより確認すること。また、フルカラーLED によりシアン・マゼンタ・イエロー・白色などの混色を再現するため、サンプルスケッチの `analogWrite()` 関数の RGB 値を変更せよ。

<課題 8.1.3> xy 色度図への投影

取得したカラーセンサの RGB 値を XYZ 値に変換後、xy 色度図上に投影する。ここで、RGB の値を XYZ 値（xy 色度図）に変換することにより、カラーマネジメントが容易になり、他の色空間（例： $L^*a^*b^*$ 色空間等）への変換も行い易くなる。まずは、自動キャリブレーションを行う サンプルスケッチ（manaba で配布） を実行し、同様に xy 色度図の上に描画せよ。

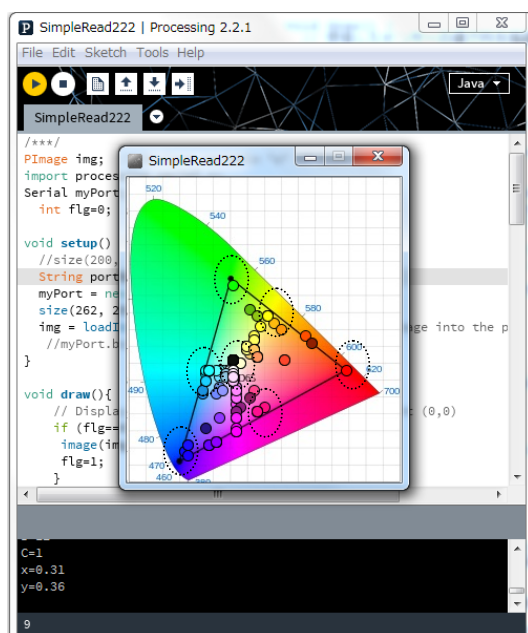


図 8.1.9 自動キャリブレーションと xy 色度図への投影

（基本色を示す○印の辺りにデータ値が描画されればキャリブレーション OK）

<課題 8.1.4> キャリブレーションの有無による結果の比較

キャリブレーションの有無による結果の比較を行う。キャリブレーションなし（演習 8.1.6）で、そのまま取り込んだカラーセンサの値を、Processing の xy 色度図の上に描画せ

よ。キャリブレーションの有無（課題 8.1.3 と本課題）による色調整の違いについて比較・検討せよ。また、xy 色度図で、実際に再現できない色があることを確認せよ。

<課題 8.1.5> センシングのタイミング

色をセンシングしてから表示するまでにかかる内部処理（readingdata 関数や Serial 関数等）の時間について、millis()関数（巻末の付録参照）を用いて計測せよ。また、計測した内部処理の時間を参考にしながら、カラーセンサによるセンシングのタイミング（サンプリング間隔）をどの様に設定すれば最も効率的かを検討せよ。

【発展課題 8.1】

色空間の変換（RGB 値⇒HSL 色空間）ができるサンプルスケッチ（S8.1.5）を実行し、RGB 値とともに、HSL 色空間に変換された値が表示されることを確認せよ。最初は、指定した色（RGB 値）の変換のみなので、通信はせず Processing だけ実行すればよい。RGB 値をいろいろと変更してみる。次に、このプログラムを図形描画用のサンプルスケッチ（S8.1.2）に組み込み、カラーセンサで取得した RGB 値（Arduino からシリアル送信）とともに、HSL 値に変換された値が表示される様にせよ。

・ S8.1.5 RGB 値から HSL 色空間への変換用サンプルスケッチ（Processing program）

```
void draw() {
  (省略)
  text("Red=",10,10); text(int(Red),40,10); //RGB
  text("Green=",120,10); text(int(Green),160,10);
  text("Blue=",230,10); text(int(Blue),265,10);
  //HSL 空間への変換
  PVector hsl = HSL(Red, Green, Blue);
  print("H=" + hsl.x + ", "); print("S=" + hsl.y + ", "); println("L=" + hsl.z);
  text("H=", 10, 20); text(int(hsl.x), 40, 20); //HSL
  text("S=", 120, 20); text(int(hsl.y), 160, 20);
  text("L=", 230, 20); text(int(hsl.z), 265, 20);
  noLoop(); //ループなしの場合
}
PVector HSL(float R, float G, float B) {
  float H=0, S=0, L=0;
  R = R / 255; G = G / 255; B = B / 255;
  float MAX = max(R, G, B); float MIN = min(R, G, B);
  L = (MAX + MIN) / 2;
  if (MAX == MIN) {
    S = 0; H = 0;
  }
  else {
    if (L <= 0.5) {S = (MAX - MIN) / (MAX + MIN);}
    else {S = (MAX - MIN) / (2 - MAX - MIN);}
    float Cr = (MAX - R) / (MAX - MIN);
    float Cg = (MAX - G) / (MAX - MIN);
    float Cb = (MAX - B) / (MAX - MIN);
    if (R == MAX) {H = Cb - Cg;}
    if (G == MAX) {H = 2 + Cr - Cb;}
    if (B == MAX) {H = 4 + Cg - Cr;}
    H = 60 * H;
    if (H < 0) {H = H + 360;}
    S = S*100; L = L*100;
  }
}
```

```
PVector p = new PVector(H, S, L);  
return p;  
}
```

【レポート 8.1】 ※切：7月19日（金）授業開始前まで.

- 全ての演習及び課題について、まとめと考察を行う.
 - PDF ファイルで提出のこと.
 - 必ず発展課題もトライすること.
 - サンプルプログラムをそのまま全てコピーしないこと.
- (自分で作成・変更したプログラム部分を中心に記述する. 各行の意味を補足する.)

【付録：主要関数】

• **Wire.requestFrom(address, count)**

他のデバイスにデータ要求を行う.

データは available() と receive() 関数を使って取得

【パラメータ】 address: データを要求するデバイスのアドレス(7 ビット)

quantity: 要求するデータのバイト数

【戻り値】 実際に受信したバイト数 (Arduino0012 以降).

• **Wire.beginTransmission(address)**

指定したアドレスの I²C スレーブに対して送信処理を開始

関数実行後, write() でデータをキューへ送り, endTransmission() で送信を実行

【パラメータ】 address: 送信対象のアドレス(7 ビット)

【戻り値】 なし

• **Wire.endTransmission()**

スレーブデバイスに対する送信を完了

【パラメータ】 なし

【戻り値】 送信結果 (byte)

0: 成功

1: 送ろうとしたデータが送信バッファのサイズを超えた場合

2: スレーブ・アドレスを送信し, NACK を受信した場合

3: データ・バイトを送信し, NACK を受信した場合

4: その他のエラー

• **Wire.write(value)**

マスターがスレーブに送信するデータをキューに入れるときに使用

beginTransmission() と endTransmission() の間で実行

【構文】 Wire.send(value)
 Wire.send(string)
 Wire.send(data, quantity)
【パラメータ】 value: 送信する 1 バイトのデータ (byte)
 string: 文字列 (char *)
 data: 配列 (byte *), quantity: 送信バイト数 (byte)
【戻り値】 送信バイト数 (byte)

• *millis()*

Arduino のプログラム開始時点から現在までの時間をミリ秒単位で返す。
約 50 日間でオーバーフローし、ゼロに戻る。

【パラメータ】 なし
【戻り値】 実行中のプログラムがスタートしてからの時間
【例】 プログラムがスタートしてからの時間を 1 秒ごとに出力

```
unsigned long time;  
void setup() {  
    Serial.begin(9600);  
}  
void loop() {  
    time = millis();  
    Serial.println(time);  
    delay(1000);  
}
```

【参考図書】

- 色彩工学の基礎, 池田光男著, 朝倉書店, 1999.
- 色彩工学 第 2 版, 大田登著, 東京電機大学出版局, 2001.
- よくわかる色彩の科学 (図解雑学), 永田泰弘, 三ツ塚由貴子著, ナツメ社, 2007.
- 色彩工学の基礎と応用, 嶋野法之, コロナ社, 2009.