8.2　センサ実験 4：カラーセンサの応用

【演習目的】

カラーセンサによる色の表現方法やセンシング技術の理解を深める．また，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，種々の測定環境で，効率よくカラーセンサのデータを測定する手法を修得する．

8.2.1　紙印刷された色の計測

先週は，フルカラーLED や液晶ディスプレイに表示されたカラーパターンで，色の計測を行った．後期のロボット実験では，紙印刷された種々の色をカラーセンサにより計測する必要がある．従って，この演習では，紙印刷したカラーパターン（図 8.2.1）を用いて，色の計測を行っていく．特に，ロボット実験で想定される測定環境を意識しながら，演習を行うことが望まれる．例えば，ロボットは常に停止しているのではなく，動きながら色を認識しなければならない．また，複数の色の境界域で，センサの値を読み込む場合も考えられる．そのため，①ロボットが動く速度と色の認識に要する時間の関係，②色の境界域（混色）でのセンシングの方法にも注意する必要がある．

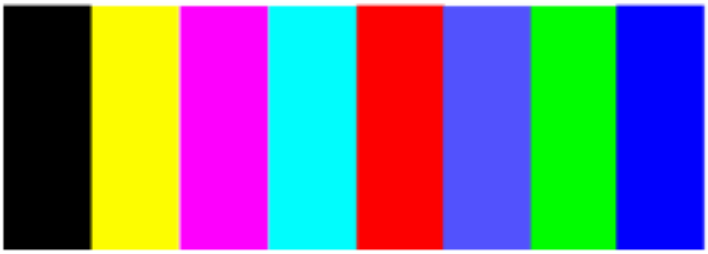

図 8.2.1 紙印刷されたカラーパターン
（左から黒・イエロー・マゼンタ・シアン・赤・（ブルーベリー）・緑・青・白）

<演習 8.2.1>　カラーセンサの配線とデータの表示（キャリブレーションなし）

先週の演習 8.1.1（図 8.1.5）に従い，カラーセンサの配線を行う．プログラムの実行前に，配線に間違いがないか必ず確認すること．次に，演習 8.1.2 で使用したサンプルスケッチ（S8.1.1）を実行し，シリアルモニタにより，RGB 値が読み込めること（キャリブレーションなし）を確認せよ．測定対象は紙の印刷物（光源ではない）のため，カラーセンサの LED ライトを ON にし，RGB 値がどの様な値で変化しているかを確認せよ．

【補足】Arduino では，シリアルモニタで表示する数値を簡単にグラフ化できるツール（シリアルプロッタ）が準備されている．Arduino IDE のメニューから[ツール]→[Serial Plotter]を選択すれば起動するので，試してみるとよい．ただし，Serial.print( )では描画されず，改行が入る Serial.println( )で，変数の値が描画されることに注意する．複数の変数を描画する場合は，カンマ（","）で各変数を区切り，最後の変数を Serial.println( )で出力するとよい．また，シリアルプロッタを利用した場合，数値情報そのものは読み取れなくなるので，簡易的な確認用の描画ツールとして利用すること．

次に，Arduino により取得したカラーセンサの RGB 値を，シリアル通信により Processing 側に送信し，可視化するプログラムを実行せよ．演習 8.1.3 で使用した描画プログラム（S8.1.2：キャリブレーションなしの場合）を実行すればよい．カラーパターンの色をセン

シングしながら，Processing で表示されている色と同じになるか（または，ばらつきを持つか）を確認せよ．

8.2.2　カラーセンサのキャリブレーション（紙印刷）

演習 8.2.1 で取得したデータは，キャリブレーションがないため，実際の色を上手く表現できていない．正しく色を表示するには，カラーセンサのキャリブレーションが必要となる．この演習では，紙印刷されたカラーパターン（白黒の部分）を用いて，ハードウェアでなく，ソフトウェア側でキャリブレーション（センサで取得できる色の最小・最大値の適切な設定による校正）を実施する．キャリブレーションは，実際の測定環境でカラーセンサを使用する前に行うことが重要である．

先週のキャリブレーションでは，カラーセンサから取得した 2 byte のデータ（Arduino 側）をシリアル通信でそのまま送り，Processing 側の map( )関数を用いて，1 byte（0～255）に変換した．しかし，ロボット実験では，カラーセンサから取得した色情報を，Arduino 側だけで判断し，動作を決定（例：停止・回避やゾーン認識等）しなければならない場合がある．従って，この演習以降は，カラーセンサのデータを Arduino で取得した直後に，map( )関数（巻末の付録参照）でキャリブレーションを実施する．Arduino の map( )関数も，カラーセンサから取得したデータ（2 byte）を，設定した RGB 値（最小・最大値）の範囲内で，1 byte（0～255）のデータに変換できる．

<演習 8.2.2>　手動キャリブレーション

Arduino 側のサンプルスケッチ（S8.2.1）のパラメータを手動で調整し，カラーセンサのキャリブレーションを行っていく．最初に，カラーパターン（図 8.2.1）の白黒部分をセンサで計測し，シリアルモニタに表示された最小値（黒色の場合）と最大値（白色の場合）を表 8.2.1（左）に記録せよ．測定の際は，カラーセンサの LED ライトを ON にしておく．サンプルスケッチには，Red 値のキャリブレーション処理のみを記述している．Red 値と同様に，Green 値と Blue 値についても，キャリブレーション処理が行える様にプログラムを変更せよ．

次に，取得した RGB 値の最小・最大を用いて，サンプルスケッチ（S8.2.1）の変数（r_min, r_max, g_min, g_max, b_min, b_max）を書き換えよ．この処理により，適切な範囲にキャリブレーションされた値が出力される様になる．実際に，カラーパターンの各色について RGB 値を計測し，表 8.2.2（左）に記録せよ．また，描画プログラム（S8.1.2：キャリブレーションなしに設定）を実行し，Processing 側で表示された色がセンシングしているカラーパターンの色と同じになるか確認せよ．大きく異なる場合は，再度，最小・最大値を取り直してみるとよい．

表 8.2.1 カラーセンサの RGB 値の最小・最大

	演習 8.2.2: 手動キャリブレーション			演習 8.2.3: 自動キャリブレーション		
センサ / パターン	Red 値	Green 値	Blue 値	Red 値	Green 値	Blue 値
黒	(r-min)	(g-min)	(b-min)	(r-min)	(g-min)	(b-min)
白	(r-max)	(g-max)	(b-max)	(r-max)	(g-max)	(b-max)

表 8.2.2 カラーパターンの各色の RGB 値

	演習 8.2.2: 手動キャリブレーション後			演習 8.2.3: 自動キャリブレーション後		
センサ / パターン	Red 値	Green 値	Blue 値	Red 値	Green 値	Blue 値
黒						
白						
赤						
緑						
青						
シアン						
マゼンタ						
イエロー						

・S8.2.1 手動キャリブレーション用サンプルスケッチ（Arduino program）

```
////////////////////////////////////////////////////////////////////////////////////////////
//初期値：RGB の最小・最大を決定
//手動キャリブレーション後、コメントアウト
unsigned int r_min = 30000, g_min = 30000, b_min = 30000;
unsigned int r_max = 0, g_max = 0, b_max = 0;
//map()関数のパラメータ
//手動キャリブレーション後、min,max を適切な値に変更
//①manual_calib()関数をコメントアウトし、②map()関数を使える様にする。
//unsigned int r_min = ???, g_min = ???, b_min = ???;
//unsigned int r_max = ???, g_max = ???, b_max = ???;
////////////////////////////////////////////////////////////////////////////////////////////
void setup() {
 （省略）
}
void loop() {
  readRGB();
  //Serial.println("RGB values = ");
  Serial.print(red); Serial.print(",");
  Serial.print(green); Serial.print(",");
  Serial.println(blue);
  manual_calib();  //①手動キャリブレーション後、コメントアウト
}
void manual_calib() {
  //Manual Calibration for red
  if (red > r_max) r_max = red;
```

```
  if (red < r_min) r_min = red;
  Serial.print("r_min = "); Serial.println(r_min);
  Serial.print("r_max = "); Serial.println(r_max);
  //Green 値・Blue 値についても同様の処理を記述
  delay(1000);
}
void readRGB(){
  （省略）
  green = readingdata[1] * 256 + readingdata[0];
  red   = readingdata[3] * 256 + readingdata[2];
  blue  = readingdata[5] * 256 + readingdata[4];
  //②map()関数によるキャリブレーション
  //red = map(red, r_min, r_max, 0, 255);
  //ここに Green 値・Blue 値の処理を記述
}
```

<演習 8.2.3>　自動キャリブレーション

手動キャリブレーションでは，カラーセンサの RGB 値（最小・最大）を毎回記録し，プログラムに反映させなければならない．そこで，カラーパターンの白黒部分にセンサを 5 秒間かざせば，RGB 値 の 最小・最大が自動で決定（自動キャリブレーション）できる様に改良する．最初に，サンプルスケッチ（S8.2.2: Red 値の場合）を実行し，カラーパターンの白黒部分から，カラーセンサの Red 値の最小・最大を自動で取得せよ．測定の際，カラーセンサの LED ライトを ON にする．自動決定された Red 値の最小・最大（シリアルモニタで確認）を，表 8.2.1（右）に記録せよ．また，Green 値と Blue 値の場合も，Red 値と同様の処理ができる様にプログラムを変更せよ．

次に，自動キャリブレーション後に計測したカラーパターンの各色について RGB 値を取得し，表 8.2.2（右）に記録せよ．さらに，Processing 側（S8.1.2：キャリブレーションなしに設定）で表示された色が，カラーパターンの色と同じになるかどうか確認せよ．センシングしている色と大きく異なる場合，蛍光灯等の外乱がなるべく入らない様にしながら，再度，自動キャリブレーションを行ってみるとよい．

※Processing（または，シリアルモニタ）とのシリアル通信開始時に，Arduino のサンプルスケッチはリセットされるため注意が必要である．この場合，setup( )関数が再実行（変数も初期化）されることになり，自動キャリブレーションも再度行う必要がある．

・S8.2.2 自動キャリブレーション用サンプルスケッチ（Arduino program）

```
#include <Wire.h>
#include "parameters.h"
#define  cal_time  5000
unsigned int i, red, green, blue;
int flg = 0;
////////////////////////////////////////////////////////////
//map()関数のパラメータ：キャリブレーション後、適切な値に変更
//初期値：RGB の最小・最大を決定
unsigned int r_min = 30000, g_min = 30000, b_min = 30000;
unsigned int r_max = 0, g_max = 0, b_max = 0;
////////////////////////////////////////////////////////////
void setup() {
```

```
  （省略）
  ////////////////////
  Serial.println("Start Auto Calibration for 5 s");
  auto_calib();     //loop に入る前の最初の 5 秒間で自動キャリブレーション
  Serial.println("End Auto Calibration");
  ////////////////////
}
void loop() {
  （省略）
}
void auto_calib() {
  unsigned long timeInit;
  timeInit = millis();
  while (1) {
     readRGB();
     if (red > r_max) r_max = red;
     if (red < r_min) r_min = red;
     Serial.print("r_min, r_max = ");
     Serial.print(r_min); Serial.print(", "); Serial.println(r_max);
     //ここに Green 値・Blue 値の処理を記述
     delay(100);
     if (millis() - timeInit > cal_time){
          flg = 1; break;
     }
  }
}
void readRGB(){
  （省略）
  green = readingdata[1] * 256 + readingdata[0];
  red   = readingdata[3] * 256 + readingdata[2];
  blue  = readingdata[5] * 256 + readingdata[4];
  if (flg == 1){ //自動キャリブレーション終了後
    //map()関数によるキャリブレーション
    red = map(red, r_min, r_max, 0, 255);
    //ここに Green 値・Blue 値の処理を記述
  }
}
```

8.2.3　色の識別

前節までの演習により，キャリブレーション後の各色の RGB 値をカラーセンサから取得できる様になった．次に，カラーセンサの計測データから，各色が正しく識別（分類）される様にプログラムを改良する．色の識別が行えれば，指定した色ごとにロボットの動作を決定したり，ライントレースによる移動にも応用できる．この演習では，色を識別するための一般的な手法として，①閾値法と②k 近傍法について説明し，プログラムで実装していく．色の識別では，事前に正しくキャリブレーションを行っておくことが必須となる．色の識別が上手くいかない場合は，再度，キャリブレーションを実行してみるとよい．

【①閾値法】

閾値を手動で設定することにより，各色の識別が行える．この演習では，各色の RGB 値を参考にしながら，色を正しく判定（分類）できる様に閾値を決めていく．例えば，白色をカラーセンサで計測する場合，RGB 値が全て 255（8 bit の場合）に近くなる．従って，カラーセンサの Red 値，Green 値，Blue 値の閾値をそれぞれ 240 辺りに設定すると，「白」の認識が上手くいくはずである．同様に，「黒」の認識では， RGB 値が全て 0 に近くなるので，閾値を 10 辺りに設定するよい．ただし，測定環境に依存して，最適な閾値が変化することも考慮する必要がある．

<演習 8.2.4>　閾値法による色の識別

この演習では，閾値法による色の識別を行っていく．最初に，白と黒を識別するためのサンプルスケッチ（S8.2.3）を実行する（演習 8.2.3 のサンプルスケッチ（S8.2.2）の一部（loop( )関数）を変更すればよい）．閾値法では，識別したい色が包含される様に，RGB 値の閾値（サンプルスケッチ内の赤字部分）を試行錯誤で設定しなければならない．カラーパターンの色をセンサで計測した際，白を認識したら「White」，黒なら「Black」，それ以外なら「Others」という文字が，シリアルモニタで正しく表示される様に閾値を調整せよ．ただし，計測前に自動キャリブレーションを実施しておく．

・S8.2.3 閾値法による色の識別のためのサンプルスケッチ（Arduino program）

```
void loop() {
  readRGB();
  Serial.print(red);  Serial.print(",");
  Serial.print(green);  Serial.print(",");
  Serial.println(blue);
  delay(100);
  //閾値の設定
  if (red > 240 && green > 240 &&  blue > 240) Serial.println("White");    //白の閾値
  else if (red < 10 && green < 10  && blue < 10 ) Serial.println("Black");  //黒の閾値
  else Serial.println("Others");
}
```

【②k 近傍法】

試行錯誤による閾値の設定だけでは，色を正しく認識することが難しい場合がある．そこで，閾値を簡単に決める方法として，機械学習アルゴリズムの一つである k 近傍法を紹介する．k 近傍法は，特徴空間の中で最も近いラベル（カテゴリ）へ統計的に分類する手法である．パターン認識にもよく用いられる．演習で使用する k 近傍法のアルゴリズムは，下記手順①～③に従う．通常，距離の算出にはユークリッド距離を用いるが，マンハッタン距離等を扱うこともできる．k 近傍法では，データが局所的に集中しているとき，分類が難しくなるので注意が必要となる．

① 識別したい色に関して，カラーセンサの RGB 値を複数回測定し，各成分の平均（代表値）を算出する．
② カラーセンサで計測している色の RGB 値と，識別に用いる色の間の距離を全て計算する．距離計算には，下記のユークリッド距離を用いる．

$$distance = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$　（RGB 値の場合，3 次元のため $n = 3$ となる）

③ 識別に用いる色（k 種類）の中から，最短となるユークリッド距離を探索し，その場合を最終的に識別（分類）した色とする．

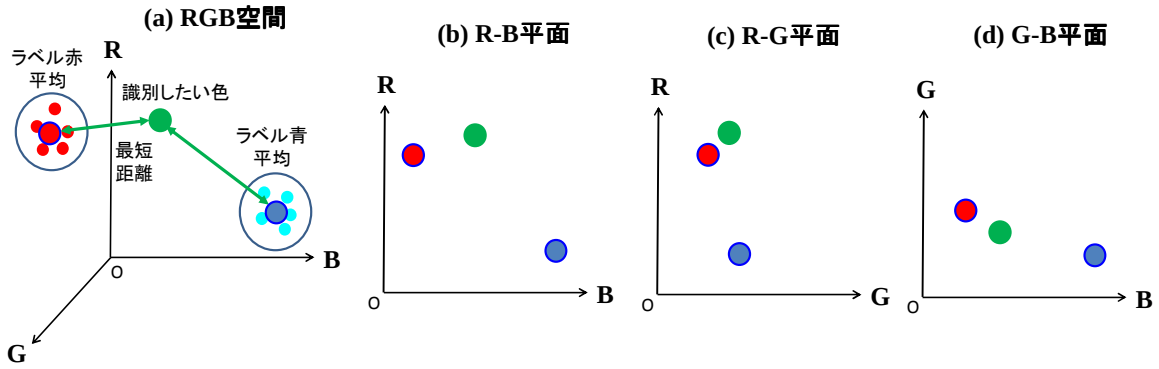

図 8.2.2 RGB 空間で計測した色と識別したい色の間のユークリッド距離

図 8.2.2(a)は，RGB 空間（3 次元）で，センシングした色と識別したい色の間のユークリッド距離計算を行う際の概念図である．図中の緑の点は分類したい測定対象（センシングした色）のデータを示す．最初に，識別対象となる点（緑）と，その他のラベル（赤か青）の各点との距離を算出する．k = 2 の場合，対象となる点（緑）の近傍から 2 つの点を取り出し，多数決により属性（どちらのラベルに属するか）が決定される．リアルタイムでの色の判別が求められる場合，全ての距離を逐次計算することが困難（計算コストの上昇）となる．そこで，識別したい色を事前に数回測定して平均（各ラベルの代表値）を求めておき，センシングした色（RGB 値）との距離のみを計算すればよい様にする．

<演習 8.2.5>　k 近傍法による色の識別

k 近傍法に基づき，色の識別を行う．この方法では，識別したい色の RGB 値の平均と，センサで感知した色の距離を計算することで，最短距離となる色を判定する．最初に，カラーパターンの白と黒を 3 回計測（シリアルモニタから数値情報を確認）し，表 8.2.3 に記録せよ．連続して計測する際は，一度，対象となるカラーパターンの色からセンサをずらし，再度，近付けて測定してみる．次に，測定した白と黒の RGB 値（3 回分）の平均をそれぞれ求めて表 8.2.3 に記録し，サンプルスケッチ（S8.2.4）の変数（ave_colors[][]）に代入せよ．ここで，平均値を求めたのは，毎回の計測で生じる誤差や観測雑音による影響を，なるべく少なくするためである．演習 8.2.4 と同様に，カラーパターンの色をセンサで計測する際，白を認識したら「White」，黒なら「Black」，それ以外なら「Others」という文字が，シリアルモニタで正しく表示されることを確認せよ．ただし，計測前に自動キャリブレーションを実施しておく．

表 8.2.3 カラーパターンの各色に対する RGB 値とその平均

カラーセンサ / カラーパターン	Red 値				Green 値				Blue 値			
	1	2	3	平均	1	2	3	平均	1	2	3	平均
黒												
白												

赤												
緑												
青												
シアン												
マゼンタ												
イエロー												

・S8.2.4 k 近傍法による色の識別のためのサンプルスケッチ（Arduino program）

```
////////////////////////////////////////////////////////////
//NN 関数のために新たに追加する項目
#define   max_colors  4  //NN: 識別したい色の数
unsigned int ave_colors[max_colors][3] = {{240, 240, 240},  //白識別：red, green, blue 値
                                          { 10,  10,  10},  //黒識別：red, green, blue 値
                                          { 64,  64,  64},  //Other1：中間色 1
                                          {192, 192, 192}}; //Other2：中間色 2
////////////////////////////////////////////////////////////
void setup() {
 （省略）
}
void loop() {
 （省略）
 //NN 関数の呼び出し
 Nearest_Neighbor();
 delay(500);
}
int Nearest_Neighbor() {
  int count = 0;
  int color = -1;
  float minDistance = 9999999;
  float distance;
  for (int i = 0; i < max_colors; i++) {  // i は設定したカラーのラベル
    distance = sqrt( (red - ave_colors[i][0]) * (red - ave_colors[i][0])
                   + (green - ave_colors[i][1]) * (green - ave_colors[i][1])
                   + (blue - ave_colors[i][2]) * (blue - ave_colors[i][2]));
    if ( distance < minDistance ) {        //最短距離の判定
      minDistance = distance;
      color = i;
    }
   }
  if (color == 0) Serial.println("White");
  else if (color == 1) Serial.println("Black");
  else if (color == 2 || color == 3) Serial.println("Others");
  else Serial.println("Not detected!");   //例外処理：未検出の場合
}
```

【実施課題】

<課題 8.2.1>　手動及び自動キャリブレーションによる相違

表 8.2.1 及び表 8.2.2 の値を比較することで，手動キャリブレーションと自動キャリブレーションの違い（または，同じか）を確認し，その結果が得られた理由を考察せよ．また，Processing の表示と一致するかどうかも報告（結果の表示等）し，得られた結果の精度について考察せよ．

<課題 8.2.2>　自動キャリブレーション時間の影響

自動キャリブレーションでは，RGB 値の最小・最大の決定に要する時間を 5 秒に設定した．この時間をさらに短くした場合（例：1 秒）と，さらに長くした場合（例：10 秒）で，結果がどの様に変化するか計測せよ．また，得られた結果に差が生じた場合，その理由についても考察せよ．

<課題 8.2.3>　閾値法による色の識別

演習 8.2.4 の閾値法により，カラーパターンの赤・緑・青，シアン・マゼンタ・イエローの各色を正しく識別できる様にプログラムを変更せよ．シリアルプロッタによる RGB 値の変化も参考にするとよい．ここで設定した各色のパラメータは，後期のロボット実験でも参考にできる．

<課題 8.2.4>　k 近傍法による色の識別

演習 8.2.5 の k 近傍法により，カラーパターンの赤・緑・青，シアン・マゼンタ・イエローの各色を正しく識別できる様に，表 8.2.3 を完成させながらプログラムを変更せよ．シリアルプロッタによる RGB 値の変化も参考にするとよい．

<課題 8.2.5>　Processing 側での色の識別結果の表示

閾値法または k 近傍法のどちらか一つを選び，Arduino で識別した色をシリアル通信で Processing 側に送信し，センシングした色とともに識別結果を表示できる様にプログラムを変更せよ．

<課題 8.2.6>　色の境界域のセンシング

課題 8.2.5 で作成したプログラムを用いて，カラーパターンの境界域全てについてセンシング（例：赤と緑の境界部分にカラーセンサを設置）を行い，Processing の表示（RGB 値と図形の描画色）により，色を正しく識別できているか確認せよ．その結果から，どの様な条件のとき，色の誤認識が起こるのか，また，どの様にすれば誤認識を改善できるか，ロボットの動きを想定しながら考察せよ．

<課題 8.2.7>　カラーセンサの移動速度と読み取り値の変化

課題 8.2.5 で作成したプログラムを用いて，手でセンサを素早く（または，ゆっくり），カラーパターンの上で移動させたとき，RGB 値がどの様に変化しているか確認せよ．その際，ロボットが動きながら，色をセンシングする場面を想定するとよい．また，移動速度に応じて，カラーセンサが誤認識（または，読み飛ばし）を起こす場合，その理由と改善策を考察せよ．

【発展課題 8.2】

(1) k **近傍法の自動化**：識別したい色の RGB 値の平均を，毎回，紙に記録するのではなく，カラーセンサで色をセンシングしながら自動的に平均を算出し，k 近傍法が実行できる様にプログラムを変更せよ．

(2) 課題 8.2.4 で測定した各色の距離の関係（3 次元空間）を，Processing（通信なしで可）で 3 次元上にプロットして確認できる様にせよ．また，どの様な色（例：白・黒・赤・緑・青・シアン・マゼンタ・イエロー等）の識別が難しくなるか考察せよ．その際，3 次元プロットの視点を変更し，2 次元上のプロット（例：図 8.2.2(b)-(d)）をしてみるとよい．また，識別したい色とプロットする点の色を同じにするとわかりやすい．

【レポート 8.2】締切：7 月 26 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】

・map( ) 関数：

範囲 [low1, high1] 内の値 value が，範囲 [low2, high2] に線形マッピングされた値を返す．

【構文】 map(value, low1, high1, low2, high2)
【パラメータ】 整数または浮動小数
【戻り値】 (float) 線形マッピング後の値

【参考図書】

- Arduino **をはじめよう 第 2 版**，Massimo Banzi 著，**オライリージャパン**，2012.