11-4章 ロボット実験４：ライントレース

本実験ではZumoを用いてライントレースを行う．Zumoはライントレースをするのに適した構造をしており，容易にライントレースを実行することができる．ライントレースはコンテスト課題を行う上でも重要なので，しっかり内容を理解してもらいたい．

実験目標

- Zumoでライントレースを実現するプログラムを作成する．
- P制御を用いたライントレースの原理を理解する．
- loop()関数を制御の基本単位としたプログラム作成．

11-4.1　サンプルプログラム

今回の実験で使用するサンプルプログラムの概要を説明しておく．本節の全体を理解しておかないと，課題をこなせないので熟読すること．

11-4.1.1　ファイル分割

Arduinoのプログラムは複数のファイルに分けて記述しても，それらのファイルを同じディレクトリに入れるだけで簡単に一つのプログラムに統合することができる．本実験では関数の機能ごとに複数のファイルに分割してプログラムを記述する．配布したサンプルプログラムではintegrateという名前のディレクトリに全てのArduinoプログラムファイルが置かれている．以下がそれぞれのプログラムファイルの概要である．

- integrate.ino: ディレクトリ名と同じ名前のプログラムファイルが必要で，このファイルにloop()関数とsetup()関数を記述する．その他の関数をここに記述することもできるが，なるべく機能ごとにファイル分割すべきである．
- color.ino: カラーセンサーに関係する関数を集めたファイル．
- linetrace.ino: ライントレースや課題のタスクに関係する関数を集めたファイル．※ 各自で中身を作成する（部分的に作成済み）．

11-4.1.2　メインプログラム

プログラムファイルintegrate.inoにはsetup()関数やloop()関数を始めとするプログラムの最上位に位置する関数を記述してある．今後作成するするであろう全ての関数から参照されるグローバル変数や定数もここで定義する．サンプルプログラムでは図1に記載された定数とグローバル変数が定義されている．

変数の名つけ方として，#defineで定義した定数は全て大文字で記述することにする．グローバル変数は，今後作成するすべて関数から参照する事を想定した最も基本的なものに限るべきである．このようなグローバル変数の変数名には末尾に'_G'をつけることにしてある．

`red_G, green_G, blue_G`	カラーセンサーで読み取った RGB 値（0-255）
`int mode_G:`	タスクのモードを表す状態変数
`timeInit_G, timeNow_G`	スタート時間 (ms)，経過時間 (ms)
`motorR_G, motorL_G`	左右の Zumo のモータに与える回転力

図 1: 全ての関数から参照される定数とグローバル変数.

ファイル integrate.ino には setup() 関数，loop() 関数，sendDate() 関数が含まれている．これらの関数の概要は以下の通りである．また，これらの関数も示しておくので，サンプルプログラムの対応を確認しながら概要を理解すること．

setup() 関数の概要：

- カラーセンサーのセットアップとキャリブレーション
- 状態変数 (mode_G）の初期値設定，その他初期値設定

loop() 関数の概要：

- カラーセンサーで RGB 値 (0-255) を取得．取得した RGB 値はグローバル変数 red_G, green_G, blue_G に格納される．今後作成する任意の関数内で RGB 値が必要な場合は，このグローバル変数を参照する．
- 現在の経過時間（ミリ秒単位）を取得．現在の経過時間はグローバル変数 timeNow_G に格納される．この値を共通時刻として任意の関数で用いる．
- 左右の Zumo モーターへの回転力の入力．この回転力はグローバル変数 motorR_G, motorL_G で与える．これらのグローバル変数への値の代入は任意の関数内で行われる．
- ライントレースに必要な具体的な処理（変数 motorR_G, motorL_G の値の決定など）は line-trace_bang_bang() 関数内で行う．今日の課題で他の関数も作成する．
- Processing に送るデータの送信.

sendData() 関数の概要：

- 50ms ごとにデータ送信．通信方式 2（4.1 通信実験（１）参照）を使用．ただし，やや簡略化してある．

integrate.ino: setup()

```
#include <Wire.h>
#include <ZumoMotors.h>
#include <Pushbutton.h>

ZumoMotors motors;
Pushbutton button(ZUMO_BUTTON);

float red_G, green_G, blue_G; // カラーセンサーで読み取った RGB 値（0-255）
int mode_G; // タスクのモードを表す状態変数
unsigned long timeInit_G, timeNow_G; //  スタート時間，経過時間
int motorR_G, motorL_G;  // 左右の Zumo のモータに与える回転力

void setup()
{
  Serial.begin(9600);
  Wire.begin();
  setupColorSensor(); // カラーセンサーの setup

  button.waitForButton(); // Zumo button が押されるまで待機
  calibrationColorSensorWhite(); // カラーセンサーのキャリブレーション (white)
  button.waitForButton(); // Zumo button が押されるまで待機
  calibrationColorSensorBlack(); // カラーセンサーのキャリブレーション (black)

  mode_G = 0;
  button.waitForButton(); // Zumo button が押されるまで待機
  timeInit_G = millis();
  motorR_G =0;
  motorL_G = 0;
}
```

integrate.ino: loop(), sendDate()

```
void loop()
{
  readRGB(); // カラーセンサでRGB値を取得(0-255)
  timeNow_G = millis() - timeInit_G; // 経過時間
  motors.setSpeeds(motorL_G, motorR_G); // 左右モーターへの回転力入力
  sendData(); // データ送信

  linetrace_bang_bang(); // ライントレースに関する処理（bang-bang制御）
}

// 通信方式2
void sendData()
{
  static unsigned long timePrev = 0;
  static int inByte = 0;

  // 50msごとにデータ送信（通信方式2），500msデータ送信要求なし-->データ送信.
  if ( inByte == 0 || timeNow_G - timePrev > 500 ||
       (Serial.available() > 0 && timeNow_G - timePrev > 50)) {
    inByte = Serial.read();
    inByte = 1;

    Serial.write('H');
    Serial.write(mode_G);
    Serial.write((int)red_G );
    Serial.write((int)green_G );
    Serial.write((int)blue_G );

    timePrev = timeNow_G;
  }
}
```

11-4.1.3　データの送受信

main() 関数から呼び出している sendData() 関数で PC にデータ（mode_G，カラーセンサーの RGB 値）を送信して Processing でデータを表示する．配布したプログラムでは通信方式 2（4.1 通信実験（1）参照）を使用している（やや簡略化してある）．この関数は loop() 関数の 1 イテレーションごとに呼ばれるが，約 50ms ごとに（正確には前回のデータ送信から 50ms 以上経過していれば）データを送信する．ただし，Processing からのデータ送信要求がないとデータを送信しない．データ送信要求が無い場合には通信が途切れているので，500ms 以上データ送信要求がない場合には（送信要求が無くても）データを送信する．このような送信方式を用いている理由は，データの送受信が途

切れた状態でデータ送信を行い続けると通信に不具合が生じるためである（理由は良く分かっていない）．PC で受信したデータを表示するためサンプルプログラムとして integrate_P.pde を配布しておくので後で利用する．

11-4.2　ライントレース

ライントレースとは，移動式ロボットが床などに描かれた線（ライン）に沿って走行することである．ラインに沿って移動するロボットをライントレーサーと呼ぶ．典型的なライントレーサーは光センサーを利用してラインを読み取りラインを追従する．それ以外にも，産業用の搬送ロボットでは床に電線を埋設し，金属テープを貼り，電磁波などを利用したライントレースを行うものもある．

多くの場合，ライントレーサーには 2〜数個の反射型フォトセンサーを装着してライントレースを行う．例えば 3 個の反射型フォトセンサーを使用した場合のライントレースの基本原理を図 2 に示す．反射型フォトセンサーは物体に光を当てその反射光を検知して物体の有無や色の濃淡を識別する．図中 (b) の状況では真ん中のフォトセンサーのみが黒を検知している，この場合ライントレーサーはラインの真上にいることになり，ライントレーサーは直進する．図中 (a) の状況では左のフォトセンサーのみが黒を検知している，この場合ライントレーサーはラインから右にずれていることになり，ライントレーサーは左に旋回する．同様に，図中 (c) の状況では右のフォトセンサーのみが黒を検知している，この場合ライントレーサーはラインから左にずれていることになり，ライントレーサーは右に旋回する．

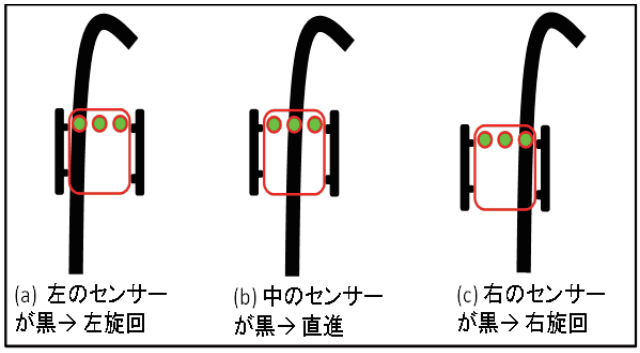

図 2: 3 個のフォトセンサーによるライントレース

本実験ではカラーセンサーを 1 つだけ用いてライントレースを行う必要があり，前述のような複数のセンサーが物理的に異なる位置に配置されていることを利用したライントレースを行うことができない．この場合でもラインの縁に沿ってライントレースすることが可能である．図 3 にその様子を示しているが，この例では（進行方向に対して）ラインの右側の縁に沿ってライントレースを行っている．基本的には，図中 (b) の状況で光センサーは白と黒を検知している，この場合ライントレーサーはラインの縁にいることになり，ライントレーサーは直進すればよい．一方，図中 (a) の状況では光センサーは黒を検知している，この場合ライントレーサーはラインの右縁から左にずれていることになり，ライントレーサーは右に旋回しなければならない．同様に，図中 (c) の状況では光センサーは白を検知している，この場合ライントレーサーはラインの右縁から右にずれていることになり，ライントレーサーは左に旋回しなければならない．

カラーセンサーにより RGB 値を得ることができるが，ligntNow（今回使用するプログラムの変数）

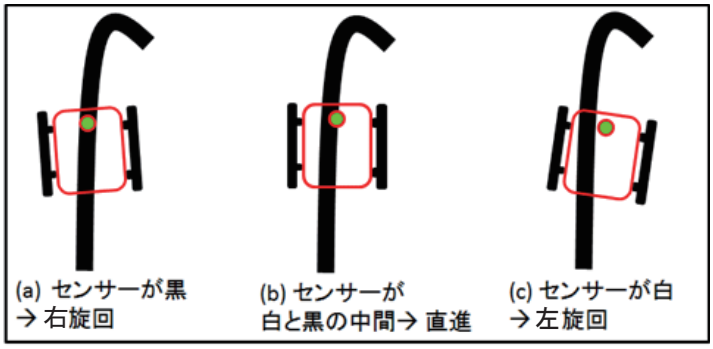

図 3: 1 つのカラーセンサーでライントレースを行う場合の概念図

を現在の RGB 値の平均値とする．すなわち，(カラーセンサーのキャリブレーションが出来ていれば）lightNow の値が 0 なら黒，255 なら白となる．また，カラーセンサが読み取る部分の黒と白の割合に応じて lightNow の値は 0 から 255 の範囲を連続的に変化する．基本的には lightNow の値が常に 127.5 となるように制御することで，ラインの縁に沿ってライントレースを行うことができる．例えば，ラインの右縁に沿ってライントレースをしたい場合であれば，127.5 ≤ lightNow の場合（白成分が多い）は右にずれていることになるので左に旋回，lightNow < 127.5 の場合（黒成分が多い）は左にずれていることになるので右に旋回するような仕組みをつくることで，原理的にはライントレースすることができる．このように，観測値の目標値からのずれが正か負かに応じて，2 種類の制御量を選択する制御法を **bang-bang 制御**と呼ぶ．

原理的には bang-bang 制御でもライントレーサーはラインに沿って移動することがでるが，動きがカクカクしてしまいスムーズにラインに沿って移動することができない（図 4 左参照）．この問題点を軽減するためにはライントレーサーにもっと滑らかな制御を加える必要がある．例えば後述する **P 制御**を用いることで，ライントレースをスムーズに実行することができる（図 4 右参照）．

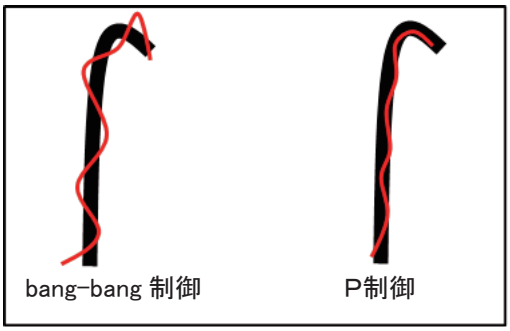

図 4: ライントレース：bang-bang 制御（左）と P 制御（右）

11-4.3　bang-bang 制御と P 制御

ライントレースを滑らかに実行するために，本実験では P 制御を用いることにする．以下では，カラーセンサーを 1 つ用いたライントレースを bang-bang 制御および P 制御で実行する方法を説明す

る．ここでは，白の背景に描かれた線（色は任意）の右縁に沿ってライントレースすることを考える．まず，以下の変数および定数を定義することにする．

- lightNow: 現在の RGB 値の平均値（変数）．
- lightMin: ライントレースする線上にカラーセンサーを置いた時の lightNow の値（定数）．
- lightMax: 背景色（本実験では白地）にカラーセンサーを置いた時の lightNow の値（定数）．

カラーセンサーがライントレースする線にちょうど半分だけかかっている時の lightNow の値を計測しておいて，lightNow の目標値としても良いが，ここでは (lightMin + lightMax)/2 をその値として用いることにする．

まずは，bang-bang 制御を考えよう．この場合は，lightNow $\leq$ (lightMin + lightMax)/2 の時に（線により過ぎなので）右回転，lightNow $>$ (lightMin + lightMax)/2 の時に（線から離れすぎなので）左回転するような制御でライントレースを実行するれば良い．bang-bang 制御を行うには loop() 関数から次の関数 linetrace_bang_bang() を実行すれば良いだろう．ただし，以下の点を補足しておく．

- 右回転または左回転のスピードはパラメータ K_p で調整する．
- 右回転または左回転する場合でも，ある程度前進しながら回転させたいので，プログラムのような式で motorL_G と motorR_G の値を決定している．
- 「各自で設定」とコメントされている数値は各自で設定する．
- パラメーターとコメントされている数値は適宜変更する．

bang-bang 制御: ライントレース

```
void linetrace_bang_bang()
{
  static float lightMin = // 各自で設定
  static float lightMax = // 各自で設定
  static float speed = 100; // パラメーター
  static float Kp = 2.0; // パラメーター
  float lightNow;
  float speedDiff;

  lightNow = (red_G + green_G + blue_G ) / 3.0;
  if ( lightNow < (lightMin + lightMax) / 2.0 ) // 右回転
    speedDiff = -Kp * speed;
  else // 左回転
    speedDiff = Kp * speed;
  motorL_G = speed - speedDiff;
  motorR_G = speed + speedDiff;
}
```

次に上のプログラムを修正して P 制御を行うプログラムを作成する（関数名は linetrace_P() とする）．P 制御では lightNow の値の (lightMin + lightMax)/2 からのずれ具合に比例した大きさで回転力（speedDiff の値）を決定すれば良い．具体的には図 5 のような関係となるように lightNow

の値から speedDiff の値を算出する．つまり，カラーセンサーが完全にラインに乗っている場合は speedDiff $= -K_p \times$speed，完全に白を見ている場合は speedDiff $= K_p \times$speed となるように，lightNow の値を線形変換する．K_p の値が大きくなるほど，lightNow の値の目標値からのずれに対する左右の回転速度の変化量が大きくなる．speedDiff の値の計算には map() 関数を使うと便利である．

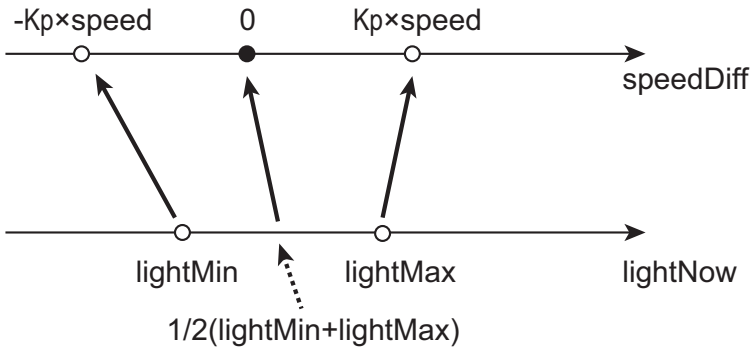

図 5: P 制御：lightNow と speedDiff の関係（矢印のないところも連続的に変化する）．

以下の課題では図 6 と同じ絵柄が印刷されたフィールド上で Zumo を動かすタスクをプログラミングする．

図 6: 実験で使用するフィールド

＜課題 11-4.1＞ 基本動作の確認

Arduino のサンプルプログラム（integrate.ino）の基本動作を確認する．また，Processing のサンプルプログラム（integrate_P.pde）を用いて Zumo の状態を表示できることを確認する．以下の実験手順に従い動作を確認せよ．また，実際の動作と Arduino プログラムとの対応関係を把握すること．

実験手順：

1. loop() 関数内の linetrace_bang_bang() 関数はコメントアウトして，プログラムを Zumo に書き込む．
2. Processing でプログラム integrate_P.pde を起動してから，Zumo の電源を入れる．

3. 以下の手順で Zumo を実行（無線通信をできるようにしておく）.
    - Zumo button を押すとカラーセンサーのキャリブレーション（白）が始まる．3 秒間直進するので，その間常に白の上を走行するように Zumo を置くこと．これにより，カラーセンサーで白を見た時の RGB 値（最大値）が大体 255, 255, 255 となる.
    - 再び Zumo button を押すとカラーセンサーのキャリブレーション（黒）が始まる．3 秒間直進するので，その間常に黒の上を走行するように Zumo を置くこと．これにより，カラーセンサーで黒を見た時の RGB 値（最小値）が大体 0, 0, 0 となる.
    - もう一度 Zumo button を押すと setup が終了して，loop() 関数に入る.
    - 現状では Zumo は動かない．ただし，RGB 値を観測して，sendData() 関数で 50ms ごとにデータ（modo_G と RGB 値）を Processing に送信している.
4. Processing には Zumo から送信されてきた RGB 値がグラフおよびコンソールに表示されているはずである．modo_G の値は常に 0.
5. フィールド上，Zumo を手で移動して RGB 値が正しく変化しているか確認する．後の課題で RGB 値の計測が必要な時はこれを用いて計測する.
6. Processing でグラフを表示中，‘s’ を押すと表示結果が固定される．その後，‘c’ を押すと表示が動き出す．これを確認せよ.

＜課題 11-4.2＞ bang-bang 制御によるライントレース

bang-bang 制御により，図 6 のフィールドの青線（一部，赤，緑）右縁に沿って反時計回りにライントレースするプログラムを作成せよ．前課題で行ったカラーセンサーのキャリブレーション後,（3 回目の Zumo ボタンを押す前に）青線の右縁に Zumo を手で移動してから（3 回目の Zumo ボタンを押して）ライントレースを開始する．サンプルプログラム内のパラメータ K_p の値と speed の値を変更して，ライントレースの挙動がどのように変化するか観察せよ.

ヒントとコメント：

- 11-4.3 節の説明を良く理解すること.
- ラインの色に赤，緑が含まれるが，全部青線と思ってライントレースしても特に問題はない.
- 現状，青線の十字路部分はうまくライントレースできなくて良い（後の課題で対処する）.
- bang-bang 制御によるライントレースを行う関数は 11-4.3 節で説明した関数 linetrace_bang_bang() 関数をそのまま用いる（コメントアウトしてある場合はコメントアウトを外す）．ただし，lightMin, lightMax の値は各自で設定する.

＜課題 11-4.3＞ P 制御によるライントレース

前課題と同様のライントレースを行う．ただし，P 制御によりライントレースするプログラムを各自で作成する．パラメータ K_p の値と speed の値を変更して，ライントレースが比較的うまく行く値に調整せよ．ライントレースがスムーズにできるようになってきたら，赤，緑，緑,（緑），赤の区間（直線区間と曲線区間のそれぞれ）を通過した際の RGB 値の変化（processing のグラフ）を記録しておく（次の課題で使用）.

ヒントとコメント：

- 11-4.3 節の説明を良く読むこと.

- 前課題で用いた linetrace_bang_bang() 関数をベースとして，P 制御用の関数 linetrace_P() を作成する．
- K_p の値の違いによる挙動の変化を観察する．
- なるべく早い速度でスムーズにライントレースできるようにパラメーター値を調整する．
- 急カーブもライントレースできることが望ましいが，難しければできなくても良い．
- 現状，青線の十字路部分はうまくライントレースできなくて良い（後の課題で対処する）．

11-4.4 タスクのプログラミングポリシー

Arduio でロボットなどの制御を行う場合，次のようなプログラミングポリシーに従ってプログラムを構成すべきである．これを **loop() 関数を制御の基本単位としたプログラミング方針**と呼ぶことにする（そのような一般名称があるわけではない）．

- ロボット制御の基本単位を loop() 関数の 1 イテレーション（1 回の繰り返し）に対応せる．すなわち，loop() 関数のイテレーションごとに，制御に必要なアクションを実行する（例えば，左右モーターへの回転力入力など）．それ以外にも，常時実行する必要のある処理は loop() 関数の 1 イテレーションごとに実行する．
- loop() 関数の実行周期（=制御周期）が長くなり過ぎないようにする (長くて数 ms 程度)．例えば制御周期が 1 秒となった場合には，1 秒間の制御不能期間が生じるため，その間に崖から落ちてしまうかもしれない．具体的には次のようなことに注意する必要があるだろう．
  - loop() 関数から呼び出される各関数に入ってから抜けるまでの処理時間が長くならないようにする．例えば，時間のかかる繰り返し処理などを行わない．同様に，loop() 関数内でも時間のかかる繰り返し処理などは行わない．
  - delay 関数は使用しない．ただし，delay 関数の使用が必要な場合もあるので（超音波センサの応答待ち等），その場合はこの限りではない．また，プログラムの試作段階では delay 関数を使用しても良いだろう．

本節の実験では図 6 と同じ絵柄が印刷されたフィールド上で単純なタスク（タスク A）を実行する．与えられたサンプルプログラムを例として，loop() 関数を制御の単位としたプログラムの書き方に慣れよう．

タスク A

1. 適当な位置（青線上）からスタートして外周を反時計回りに（右縁に沿って）ライントレース．
2. 赤線を通過した後に青線に戻ったら 1.0 秒停止する．その後，ライントレースを再開．
3. 緑線を通過した後に青線に戻ったら 0.5 秒停止する．その後，ライントレースを再開．

loop() 関数を制御の基本単位としたプログラミング方針では，次のようにプログラムを記述するのが良いだろう．

- タスクを単純な処理に分割し，分割された処理に番号（モードと呼ぶことにする）を割り振る．
- モードに応じて処理を分岐させる．
- 各モードの処理を実行中，指定した条件を満たした時に別のモードに遷移（状態遷移）する．

例えば，タスク A では次のように処理を分割してモードを定義すると良いだろう．下記の itemize の番号はモード (mode) を表わすものとする．ただし，mode=0 からスタートする．

0. 何もせずに mode=1 へ遷移．形式的にこのようにしておく．例えば，後から必要な処理（初期設定など）を追加したりできる．
1. （青線上の）ライントレースを実行．赤線または緑線と判断したら mode=2 へ遷移．ただし，後 (mode=3 で）何ミリ秒停止するか記録しておく（赤なら 1 秒，緑なら 0.5 秒）．
2. （赤または緑線上の）ライントレースを実行．青線と判断したら，mode=3 へ遷移．
3. 記録していた時間だけ停止．その後，mode=1 へ遷移．

タスク A を行うためのサンプルプログラム（task_A()）は作成してあるので，後の課題ではこれを利用する．プログラムを良く読んで中身をしっかりと理解すること（解説も良く読むこと）．このプログラムを理解しないと，その後の課題ができない．

task_A() 関数

```
void task_A()
{
  static int stop_period;    // static 変数であることに注意
  static long int startTime; // static 変数，時間計測は unsigned long int
  char color;

  switch ( mode_G ) {
    case 0:
      mode_G = 1;
      break;   // break 文を忘れない（忘れるとその下も実行される）

    case 1:
      linetrace_P(); // ライントレース（各自で作成）
      color = identify_RGB(); // ラインの色を推定 (R:赤，G:緑，B:青，-:それ以外)
      if ( color == 'R' ) { // red
        stop_period = 1000; // 後で停止する期間
        mode_G = 2;
      }
      else if ( color == 'G' ) { // green
        stop_period = 500; // 後で停止する期間
        mode_G = 2;
      }
      break;

    case 2:
      linetrace_P(); // ライントレース
      if ( identify_RGB() == 'B' ) { // brue
        startTime = timeNow_G; // mode_G=3 に遷移した時刻を記録
        mode_G = 3;
      }
      break;

    case 3:
      motorL_G = 0; // 停止
      motorR_G = 0;
      if ( timeNow_G - startTime > stop_period ) // 指定時間経過したら
        mode_G = 1;
      break;
  }
}
```

解説：

- main() 関数から task_A() 関数が呼ばれるが，この関数に長くとどまるような処理はしていない．main() 関数の周期は数ミリ秒程度になっている．
- linetrace_P() 関数は各自で作成したものを利用する．この中で motorL_G と motorR_G の値がセットされる．
- task_A() 関数の中では motorL_G と motorR_G の値を決定するだけ．モーターに回転力を与える命令は main() 関数内で行っている．
- 関数を抜けても覚えておかなくてはならない値を保持するローカル変数は static 宣言する．static 宣言していないローカル変数の値は関数を抜けると失われてしまう．static 宣言された変数はその関数が初めて呼ばれた時にのみ初期化される（初期化の値が設定されていなければ 0 がセット）．
- identify_RGB() 関数は現在の RGB 値からライントレースをしている線の色を推定：赤，青，緑，それ以外に対して，それぞれ char 型の値 'R'，'G'，'B'，'-' を返す．

次に，ラインレース中にどの色の線をライントレースしているか，判定する方法について説明する．サンプルプログラムでは identify_RGB() 関数がこれを行っている（上記参照）．図 7 は図 6 フィールドの青線の外側を反時計回りにライントレースした際の曲線上の (赤-緑-緑-緑-赤) 区間の RGB 値の変化を表したグラフである（各自，課題 11-4.3 で保存しているはずである）．例えば，赤線上であれば RGB 値の R の値が大きくなる（緑と青も同様）．グラフを見て分かるように基本的には RGB 値の最も大きな値に対応する色の線上をライントレースしていると判断できそうである．しかし，完全に白を見ている場合，RGB 値は全て 255 となるので，RGB 値のいずれかが偶然最大となってしまう．このような問題を回避するために，identify_RGB() 関数では R 値 > 1.2×G 値 かつ R 値 > 1.2× B 値 の時，赤線上（他の色でも同様）と判断するように作成してある．ただし，1.2 はパラメータ値である．また，この関数よりもっと良い判定法があると思われるので，この関数に固執する必要はない．

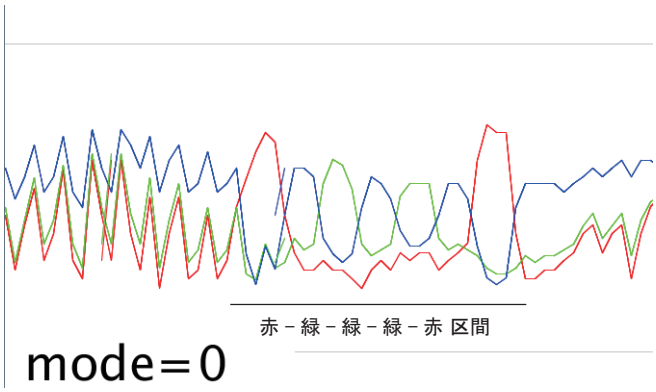

図 7: RGB 値のグラフ

identify_RGB() 関数

```
char identify_RGB()
{
  float alpha = 1.2; // パラメーター
  // _*_G はグローバル変数
  if ( blue_G > alpha * red_G && blue_G > alpha * green_G )
    return 'B';
  else if ( red_G > alpha * blue_G && red_G > alpha * green_G )
    return 'R';
  else if ( green_G > alpha * red_G && green_G > alpha * blue_G )
    return 'G';
  else
    return '-';
}
```

＜課題 11-4.4＞ タスク A の実行

タスク A を実行するプログラムを実行して挙動を確かめよ．また，これまで課題と同様 Processing で Zumo の状態（RGB 値と mode の値）を表示せよ．また，Zumo を動かしている状態で，Zumo ボタンを押すと停止し，もう一度 Zumo ボタンを押すと mode=0 に戻るように改良せよ（キャリブレーションなしで実験をやり直せる）．

ヒントとコメント：

- main() 関数内も変更すること．
- Zumo の実行と Proccesing の実行の基本手順はこれまでの課題と同様である．
- 課題 11-4.3 で保存した processing のグラフを元に，identify_RGB() 関数内のパラメーター alpha の適切な値を見積り設定せよ．
- Processing で Zumo の状態が表示されるので，mode の遷移が意図したものになっているか確認せよ．mode の遷移が予想と異なる場合は，何がうまくいっていないかのヒントが得られる．
- サンプルプログラムの内容をしっかり理解すること．
- Zumo ボタンを押して mode=0 に戻るためのプログラムは main() 関数内に以下を記述すればよい．

```
if( button.isPressed() ){ // Zumo ボタンが押されたら
   mode_G = 0;
   motors.setSpeeds(0, 0);
   delay(200);
   button.waitForButton(); // Zumo ボタンが押されるまで待つ
}
```

11-4.5 課題タスク

次のタスク（タスク B）を考えよう．

タスク B

1. 適当な位置（青線上）からスタートして外周を反時計回りに（右縁に沿って）ライントレース．
2. 途中，赤-赤と通過したら，2 回目の赤を見た直後から 1 秒間直進する．※その後にある十字路をスムーズに通過するのが目的なので，これがで上手くできるように適宜秒数は設定して良い．
3. 途中，赤-緑（数回）-赤と通過したら，2 回目の赤を見た直後から左に 90° 回転し，その後直進する．黒線を通過して白を検知したら（黒枠内に入った）1 秒間停止する．その後，緑の回数だけ直進（0.5 秒）-ストップ（0.5 秒）を繰り返す．その後は停止し続ける．

＜課題 11-4.5 ＞ タスク B の実行

タスク B を実行するプログラムを作成して実行せよ．作成する関数の関数名は task_B() とする．また，これまで課題と同様 Processing で Zumo の状態（RGB 値と mode の値）を表示して動作を確認せよ．

ヒントとコメント：

- 状態（mode）の遷移図を書いてからプログラムを作成する．状態の遷移図は予習段階で作成しておくのが望ましい．状態遷移は分岐することもある．
- Processing で Zumo の状態を表示して，mode の状態遷移が意図したものになっているか確認する．
- ある mode に本当に入っているかどうか確認したい場合，その mode 内で

```
motors.setSpeeds(0, 0); // 停止
delay(10000);
```

  のように記述して，Zumo を停止させてみるのも一つの手である．
- 緑の回数だけ直進（0.5 秒）-ストップ（0.5 秒）を繰り返す部分は，とりあえず delay() 関数を使って実装するのが簡単である（ただし，loop() 関数を制御の基本単位としたプログラミング方針に反するので最終的には修正する）．具体的には下記のようになるだろう．

```
case ○:
  for( int i = 0; i < n_green; ++i ){ // n_green は緑の回数
    motors.setSpeeds(100, 100); // 直進
    delay(500);
    motors.setSpeeds(0, 0);     // 停止
    delay(500);
  }
  mode = ○+1; // 遷移先のモード
  break;
```

- 90° 回転については，本来は地磁気センサーを使って実装すべきであるが，今回は単純化して，指定時間だけ回転で代用してよい．
- 緑の回数をカウントする変数は static 変数で定義する（なぜか分からなければ誰かに聞く）．
- 現在の RGB 値が指定した RGB 値に近い値になったかどうかを判定する関数 identify_color(int red, int green, int blue) をサンプルプログラム（linetrace.ino）に用意してあるので使用して良

い．この関数は引数として与えた RGB 値と現在の RGB 値との 3 次元空間上の距離が指定した閾値（d2_max）以下なら 1，それ以外なら 0 を返す関数である．使用する場合はプログラムの中身を確認すること．

11-4.6 発展課題

＜発展課題 11-4.1 ＞ 状態を維持する関数

課題 11-4.4 で必要になったように，あるモードの状態を指定した時間だけ維持したい（その後，別のモードに遷移）ということは多い．そのような場合に有効な関数として，サンプルプログラム（linetrace.ino）には maintainState(..) 関数というものを作成してある．この関数とその説明が以下に記載されているので，これを用いて課題 11-4.5 のプログラムを作成せよ．

maintainState() 関数

```
int maintainState( unsigned long period )
{
  static int flagStart = 0; // 0:待ち状態，1:現在計測中
  static unsigned long startTime = 0;

  if ( flagStart == 0 ) { // 待ち状態の場合は計測開始
    startTime = timeNow_G; // 計測を開始した timeNow_G の値を覚えておく
    flagStart = 1; // 現在計測中にしておく
  }

  if ( timeNow_G - startTime > period ) { // 経過時間が指定時間を越えた
    flagStart = 0; // 待ち状態に戻しておく
    startTime = 0; // なくても良いが，形式的に初期化
    return 1;
  }
  else
    return 0;
}
```

解説：

- maintainState(unsigned long period) 関数は最初に呼び出されて以降，再度呼び出された場合に，引数で指定する時間 period(ms) が経過していなければ 0 を返し，指定した時間を経過していれば 1 を返す．1 を返した後は待ち状態となる（次に呼び出された時は最初に呼び出されたとみなす）．
- 関数を抜けても覚えておかなくてはならない値を保持するローカル変数は static 宣言する．static 宣言していないローカル変数の値は関数を抜けると失われてしまう．static 宣言された変数はその関数が初めて呼ばれた時にのみ初期化される（初期化の値が設定されていなければ 0 がセット）．
- 関数で行う一連の処理が終了した時（この場合は，指定時間が経過して値 1 を返した時），static で定義された変数の値を次に呼び出される時に備えて初期値に戻している．

- この関数を 2 か所（以上）から同時に使用してはならない．そうした使い方をすると意図した動作ではなくなってしまう（なぜそうなのか考えよ）．

＜発展課題 11-4.2＞ タスク C

下記のタスク（タスク C）を実行するプログラムを作成して実行せよ．

タスク C

1. タスク B と同じ．
2. タスク B と同じ．
3. 途中，赤-緑（数回）-赤と通過したら，最後の赤の直後から緑の回数だけライントレース（0.5 秒）－ストップ（0.5 秒）を繰り返す．その後，ライントレースを再開．

【レポート 11-4（2019 年 10 月 24 日出題，2019 年 10 月 31 日 (12:50) 締切】

※ プログラムには詳細なコメントを記述すること．プログラムを提出するレポートについてはコメントを考察として採点する．

レポート 11-4.1

課題 11-4.1（基本動作の確認）ではカラーセンサーのキャリブレーションを行っている．具体的にどのようなキャリブレーションを行っているのか，プログラムの中身を読んで説明せよ．

レポート 11-4.2

課題 11-4.2（bang-bang 制御によるライントレース）において，K_p の値の違いによってライントレースがどのように変化したか記述せよ．また，bang-bang 制御の問題点を説明せよ．

レポート 11-4.3

課題 11-4.3（P 制御によるライントレース）に関し，下記 (a)-(d) ごとに分けてレポートを作成せよ：(a) 作成したプログラム（作成部分のみ）を報告せよ．(b) bang-bang 制御と P 制御の場合でライントレースの動作がどのように違ったか記述し，なぜそのようになるのか原理を説明せよ．(c) K_p の値の違いによってライントレースがどのように変化したか記述せよ．(d) 高速かつスムーズにライントレースを行うことのできた K_p の値と speed の値を報告せよ．また，赤，緑，緑,（緑），赤の区間（直線区間と曲線区間のそれぞれ）を通過した際の RGB 値の変化（Processing のグラフ）を貼り付け，どのような特徴があるか考察せよ．

レポート 11-4.4

課題 11-4.4（タスク A の実行）では loop() 関数を制御の基本単位としたプログラミング方針を学んだ．このプログラミング方針の利点（これを使わなかった場合の問題点）を記述せよ．実験結果と問題点（あれば）を述べよ．

レポート 11-4.5

課題 11-4.5（タスク B の実行）で作成したプログラムを報告せよ．また，どのような状態遷移を行ったのか説明せよ（フローチャートでも良いし，テキストのタスク A の状態遷移の説明（p.322）のような感じでもよい）．結果，工夫した点，問題点を述べよ．

発展課題レポート 11-4.1

発展課題 11-4.1（状態を維持する関数）で作成したプログラム（作成部分だけ）を報告せよ．maintainState() 関数の解説で，この関数を 2 か所以上から同時に呼び出した場合に意図した動作をしないと書かれているが，その理由を説明せよ．

発展課題レポート 11-4.2

発展課題 11-4.2（タスク C）で作成したプログラムを報告せよ．また，どのような状態遷移を行ったのか説明せよ．