要素技術とモデルを開発に使おう

要素技術をシステムに組込もう

ETロボコン実行委員会

2.3, 2019-04-12 16:47:10

sample03（ライントレーサー）を動かしてみよう

On/Off制御を使った簡単なライントレースのサンプルを動かしてみましょう。

1. sample03 をビルドし、EV3本体に転送します
2. 黒い線の上から走行するよう置きます
3. 反時計回りにラインの外周に沿って走行します
4. 左ボタンを押すと停止します
5. 環境に合わせて、カラーセンサーのしきい値（ mThreshold ）を調整しましょう

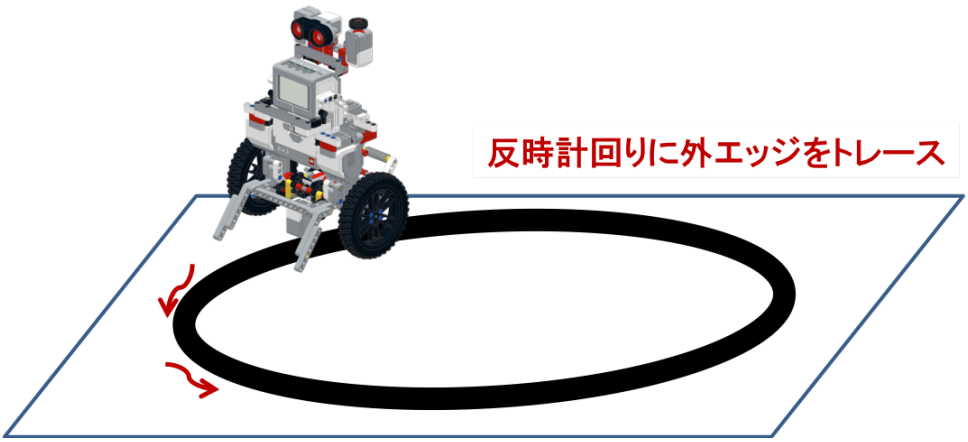

図 1.トレーサーの走行の様子

sample03のコードを見てみよう

- main_task とは別にトレース用のタスク（ tracer_task ）を使っています
- 1回分のトレース動作をする tracer_task を繰り返し動作させるために、周期ハンドラを使っているのがわかります

リスト 1. sample03/app.cfg（タスク構成）

```
INCLUDE("app_common.cfg");

#include "app.h"

DOMAIN(TDOM_APP) {
CRE_TSK( MAIN_TASK, { TA_ACT,  0, main_task,   MAIN_PRIORITY,   STACK_SIZE, NULL } );
CRE_TSK( TRACER_TASK, { TA_NULL,  0, tracer_task, TRACER_PRIORITY, STACK_SIZE, NULL });

EV3_CRE_CYC( TRACER_CYC, { TA_NULL, 0, tracer_cyc, 50, 1});
}

ATT_MOD("app.o");
ATT_MOD("util.o");
ATT_MOD("Tracer.o");
```

リスト 2. sample03/app/Tracer.h

```
#include "Motor.h"
#include "ColorSensor.h"
#include "util.h"

using namespace ev3api;

class Tracer {
public:
  Tracer();
  void run();
  void init();
  void terminate();

private:
  Motor leftWheel;
  Motor rightWheel;
  ColorSensor colorSensor;
  const int8_t mThreshold = 20;
  const int8_t pwm = (Motor::PWM_MAX) / 6;
};
```

リスト 3. sample03/app/Tracer.cpp (抜粋)

```
void Tracer::terminate() {
  msg_f("Stopped.", 1);
  leftWheel.stop();
  rightWheel.stop();
}

void Tracer::run() {
  msg_f("running...", 1);
  if(colorSensor.getBrightness() >= mThreshold) { ❶
    leftWheel.setPWM(0);
    rightWheel.setPWM(pwm);
  } else {                      ❷
    leftWheel.setPWM(pwm);
    rightWheel.setPWM(0);
  }
}
```

❶ 明るい時(ライン外)の処理、左に曲がっています

❷ 暗い時(ライン上)の処理、右に曲がっています

リスト 4. sample03/app.cpp

```
// tag::tracer_def[]
#include "app.h"
#include "Tracer.h"
using namespace ev3api;

Tracer tracer;                  // Tracerのインスタンスを作成

void tracer_cyc(intptr_t exinf) {
  act_tsk(TRACER_TASK);
}

void tracer_task(intptr_t exinf) {
  if (ev3_button_is_pressed(LEFT_BUTTON)) {
    wup_tsk(MAIN_TASK);         // 左ボタン押下でメインを起こす
  } else {
    tracer.run();               // 走行
  }
  ext_tsk();
}
// end::tracer_def[]
// tag::main_task[]
void main_task(intptr_t unused) {
  tracer.init();
  ev3_sta_cyc(TRACER_CYC);
  slp_tsk();                    // 起きたら、走行をやめる
  ev3_stp_cyc(TRACER_CYC);
  tracer.terminate();
  ext_tsk();
}
// end::main_task[]
```

sample03のクラス図を描いてみよう

sample03 の構造を表すクラス図を描いてみよう

- sample01 のクラス図を複製して修正するとよいでしょう
- タスク(main_task 、tracer_task)と周期ハンドラ(tracer_cyc)は C++ のクラスではありませんが、この演習の便宜上クラスを使って表しておきましょう
- app パッケージを追加して、そこに Tracer クラスを追加しましょう
    - 名前空間で親を表示しておくとパッケージ名が可視化できます
- ev3apiライブラリのクラスの詳細(属性やメソッド)は非表示にしておくとコンパクトになるでしょう
- 属性の初期値をプロパティから入力して、表示するように設定するとよいでしょう

作成したsample03のクラス図

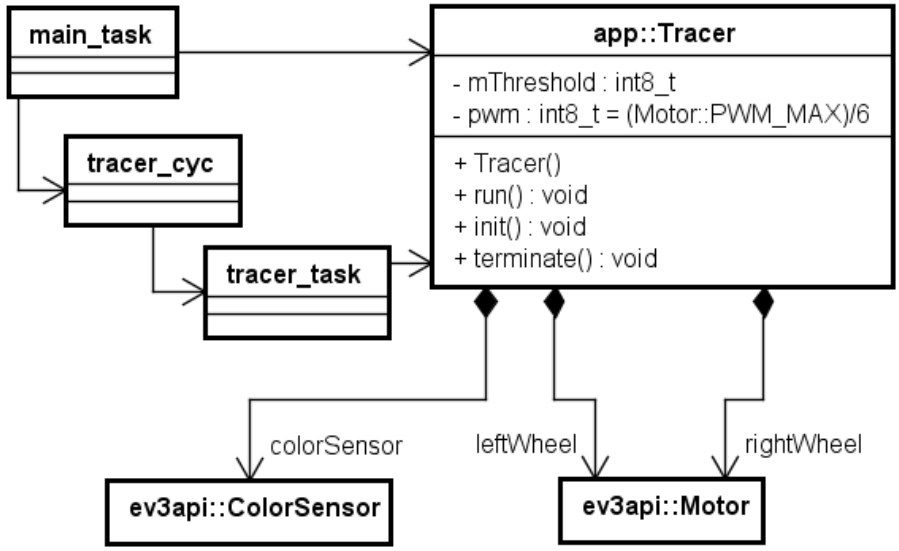

図 2. sample03 のクラス図

sample03の振舞いの図を描いてみよう

クラス図だけでは、どのように走行しているかわかりません。
あるメソッドがどのような振舞いをするかを図に描くには、アクティビティ図やステートマシン図を使います。

ここでは、Tracer クラスの run メソッドの振舞いを表すのにアクティビティ図を使ってみましょう。

Tracer クラスの run メソッドのアクティビティ図の作成手順:

- run メソッドの処理の流れがわかるよう、アクティビティ図を使ってみます
- メニューから「図」→「アクティビティ図」→「新規アクティビティ図」で作成します
- プロパティのベースタブで、図の名前を「 Tracer クラスの run メソッドのアクティビティ図」としましょう
- 処理には「アクションノード」を、条件分岐には「ディシジョンノード」を使います
    - 条件による分岐では、判断条件を「ガード条件」で表します
- 処理の開始場所がわかるよう「開始ノード」を使います
- 処理の終了場所が判るよう「終了ノード」を使います

作成したrunメソッドのアクティビティ図

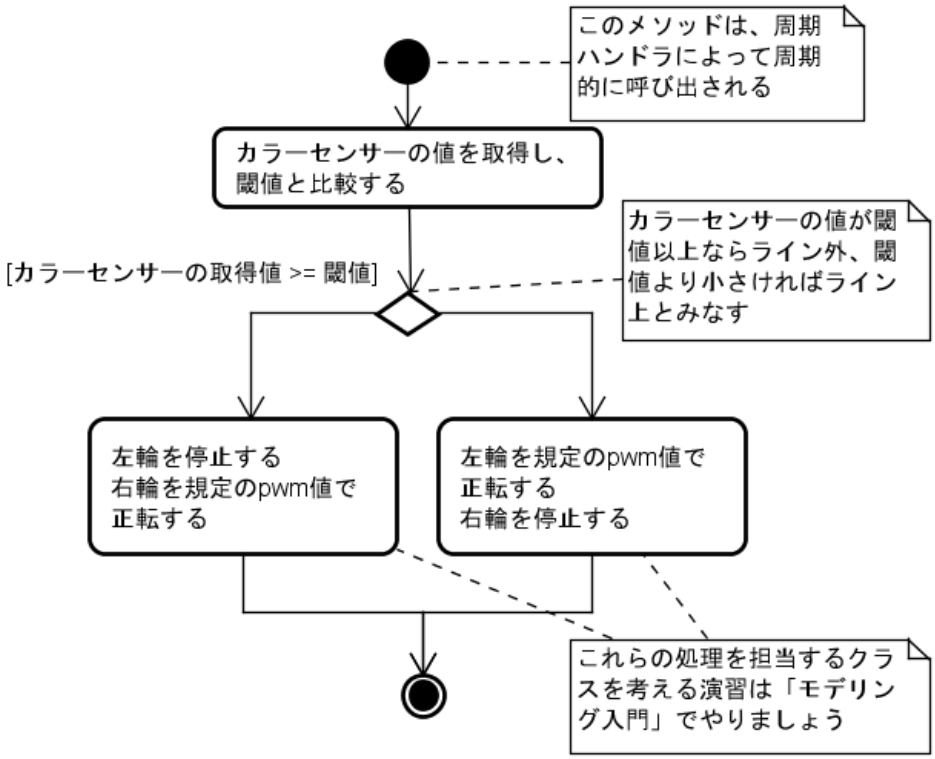

図 3. Tracer クラスの run メソッドのアクティビティ図

sample03の課題を検討しよう

sample03 のライントレースは…

- ライン上かライン外かで左右のモーターをOn/Offする方法で左右に曲がりながら走行していました
- 左右に揺れながら走行するので、姿勢が安定せず速度を上げると不安定になってしまうでしょう

> もう少し滑らかに走行するには、どんなことを調べたり、考えたりすればよいでしょうか。

カラーセンサーの値を詳しく調べてみよう

もう少し滑らかに走行する方法を考えるために、カラーセンサーの動作を少し調査してみましょう。

測り方1:組込みのPort Viewを使う

1. SDカードを挿入していない状態でEV3本体を起動します
2. 「Port View」を選択して、中央ボタンを押します
3. 左右ボタンを操作してカラーセンサーのポートを選択します
4. カラーセンサーの取得値が表示されます
5. ロボットを手でライン上からライン外へゆっくり移動させながら、カラーセンサーの取得値がどのように変化するか確認しましょう
6. センサーとラインの位置と調べた値との関係をまとめて図にしておくとよいでしょう

測り方2：helloev3を使う

1. EV3RTの配布パッケージには、workspaceに「helloev3」というサンプルが含まれています
2. 「helloev3」をmakeしてEV3本体に転送し、メニューから選んで起動します
3. 「Connect Device」→「Connect Sensor」で「Sensor Port 3」を「Color Sensor」に設定します
4. 最初のメニューに戻って「Test sensor」→「Sensor port 3」を選択します
5. 「Reflect」を選択すると、赤外線ランプが点灯し、センサーが取得した明るさが表示されます
6. ロボットを手でライン上からライン外へゆっくり移動させながら、カラーセンサーの取得値がどのように変化するか確認しましょう
7. センサーとラインの位置と調べた値との関係をまとめて図にしておくとよいでしょう

滑らかに走る方法を考えてみよう

ラインの境界付近でのカラーセンサーの値:

- カラーセンサーの取得値は、白と黒の中間の値で徐々に変化しています
- 中間の値の場合、まだラインの境界付近から大きくずれていないといえます
- 大きくずれていないならば、旋回する度合いをその分小さくすると、ブレが減って滑らかになりそうです

旋回する度合いを小さくするには:

- 左右の旋回を切替えるとき、PWM値をいきなり「0」にして止めてしまわないで、中間値の大きさに応じて小さくしてみたらどうでしょうか
- カラーセンサーの目標値を白と黒の真ん中にしておき、実測値との差を求め、差の大小に応じて左右のモーターのPWM値を加減すれば、ズレに応じて旋回の度合いを調整できそうです

プログラムで使えるよう整理しよう

いま考えているような制御の方法を「比例制御」といいます。
前のページで考えたことを、プログラムで計算できるよう整理しましょう。

- 偏差 ＝ カラーセンサーの測定値 - カラーセンサーの目標値
- 調整値 = 偏差に比例した値（偏差に係数をかけた値）
- 調整後の左のモーターのPWM値 = 基準のPWM値 - 調整値
- 調整後の右のモーターのPWM値 = 基準のPWM値 + 調整値

調整値を求めるときに使う係数を「比例ゲイン（Kp）」、
調整後の左右のモーターのPWM値を「操作量」といいます。

これでうまくいきそうでしょうか

アイディアは整理できましたが…

- まだ、効果的な方法かどうか確かめられていません
- どのようにプログラムを作れば実現できるかわかっていません
- どのくらいの値に調整すればよいかわかっていません

アイディアの妥当性を確かめるには

実験しましょう！

考えたことを実際に試してみて、効果や実現方法を決めましょう。

sample04を作成しよう（比例制御の実験）

1. サンプルコードの sample03 ディレクトリをそっくりコピーして sample04 ディレクトリを作ります
2. 考えた方法に合わせて app/Tracer.cpp の run メソッドを編集します
3. 編集が済んだら、期待したとおりに動くかどうか確認してみましょう
4. うまくいかない時は、定数の値をいろいろ変更して実験してみるとよいでしょう

作成したコードは次ページに掲載してあります。
定数は、みなさんで調整してみてください。

実験用に作成したrunメソッドの例

リスト 5. sample04/app/Tracer.cpp（runメソッドの部分を抜粋）

```
18 void Tracer::run() {
19   const float Kp = 0.83;        ❶
20   const int target = 10;        ❷
21   const int bias = 0;           ❸
22
23   msg_f("running...", 1);
24   int diff = colorSensor.getBrightness() - target;
25   float turn = Kp * diff + bias;
26   leftWheel.setPWM(pwm - turn);
27   rightWheel.setPWM(pwm + turn);
28 }
```

❶ 比例係数（1より小さい値で検討してみましょう）

❷ 白と黒の中間値を目標値として設定（実際に測った値を使ってみましょう）

❸ 調整値にいつも一定の値（バイアス）を追加しておきたい場合に設定します

実験はうまくいきましたか？

これでうまくいったので、オッケーでしょうか。

- `sample04` で実験して実現したことは、モデル図のどこに反映されていますか
- `sample03` のモデル図となにが変わったか考えてみましょう

まだ実験しか終わっていなかった！

どうやら、実験は成功しましたが…、
その結果を、システムに組込んで使う仕事がまだだったようです。

実験の成果をモデル図に反映する

実験した結果を反映したモデル図を作成しましょう。

1. astah* で `sample03` のプロジェクトを「ファイル」→「プロジェクトの別名保存」で `sample05` として保存します
2. クラス図の名前を `sample05のクラス図` に変更します
3. アクティビティ図を編集して、実験の時に確立した処理（次の2つを順に処理する）に合わせます
    - 比例制御の調整値の計算をしているところ
    - 実際に走行させるところ
4. 調整値の計算は、走行そのものとは別の処理です名前をつけて分けたほうがよいでしょう
    - 処理を分けて `calc_prop_value` メソッドを作り、これを使うようにしましょう

実験を反映したアクティビティ図

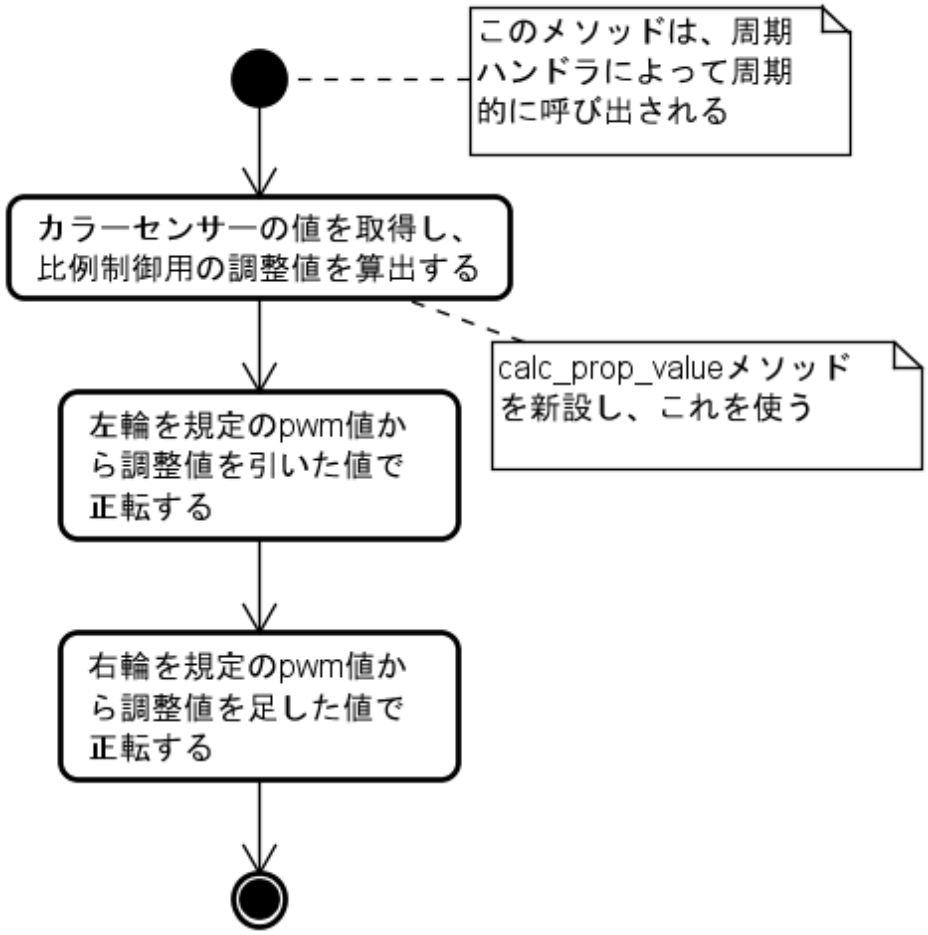

図 4. 実験を反映した Tracer クラスの run メソッドのアクティビティ図

実験を反映したクラス図

- クラス図を編集して `Tracer` クラスに `calc_prop_value` メソッドを追加しましょう。
- 内部で使う操作なので private なメソッドにしてあることに注意しましょう。

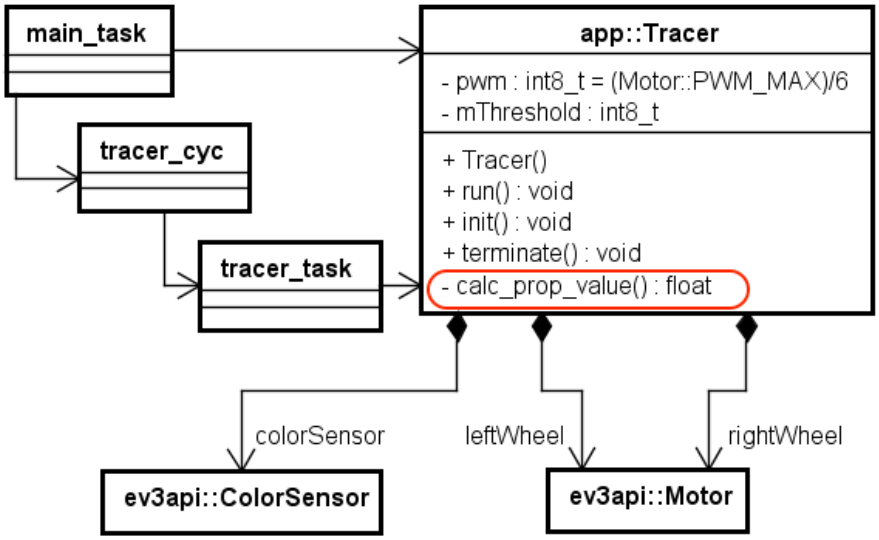

図 5. sample05 のクラス図

sample05のコードを作成する

モデル図に合わせて、コードを変更しましょう。

1. サンプルコードの sample04 ディレクトリをそっくりコピーして sample05 ディレクトリを作りましょう
2. クラス図に従って Tracer.h に calc_prop_value メソッドを追加します
3. Tracer.cpp にも calc_prop_value メソッドを追加します
4. Tracer クラスの run メソッドを calc_prop_value メソッドを使うように修正します

作成したコードは次ページに掲載してあります。

コードが作成できたら、ビルドして、動作を確認しましょう。

リスト 6. sample05/app/Tracer.h

```
15 private:
16   Motor leftWheel;
17   Motor rightWheel;
18   ColorSensor colorSensor;
19   const int8_t mThreshold = 20;
20   const int8_t pwm = (Motor::PWM_MAX) / 6;
21
22   float calc_prop_value();      ❶
23 };
```

❶ calc_prop_value メソッドを追加した

リスト 7. sample05/app/Tracer.cpp (calc_prop_value メソッド)

```
19 float Tracer::calc_prop_value() {
20   const float Kp = 0.83;        ❶
21   const int target = 10;        ❷
22   const int bias = 0;
23
24   int diff = colorSensor.getBrightness() - target; ❸
25   return (Kp * diff + bias);                       ❹
26 }
```

❶ 比例係数

❷ 目標値とした白・黒の中間値

❸ 偏差を求める

❹ 調整値を求めて返す

リスト 8. sample05/app/Tracer.cpp (run メソッド)

```
29 void Tracer::run() {
30   msg_f("running...", 1);
31   float turn = calc_prop_value(); ❶
32   int pwm_l = pwm - turn;       ❷
33   int pwm_r = pwm + turn;       ❷
34   leftWheel.setPWM(pwm_l);
35   rightWheel.setPWM(pwm_r);
36 }
```

❶ 比例制御の調整値を求める

❷ 基準値と調整値を使って操作量を求める

ここまでのまとめ

sample05 のモデル図とコードを作成するまでにどんなことをやったでしょうか。

- 元になる sample03 を動かしてみました
  - 周期ハンドラを使って周期的に処理しました
  - ファイルをクラスごとに分け、パッケージにディレクトリを割当てました
- 滑らかに走行するためのアイディアを実験しました(sample04)
  - カラーセンサーの値の変化を調べました
  - 調べた結果を使って滑らかに走行できるか実験しました
- 実験した結果を使ってモデルを更新しました(sample05)
  - 更新したモデル図に合わせてコードを作成し、動作を確認しました

要素技術がモデルの要素になっていること

モデル図に要素技術を活用している様子がわかるようにするには：

- モデル図上に要素技術を実現している構成要素として登場しているべきでしょう
- モデル図に登場するには、クラスやメソッドなど、モデル図の要素として表す必要があります

要素技術をシステムに組込む手順

次のような検討手順で進めてみるとよいでしょう:

1. アイディアを出す、必要な情報を取得する・調査します
2. 実験してアイディアの効果を確かめます
3. 得られた結果を元のシステムのモデル図に「名前をつけて」組込みます
4. 更新したモデル図に合わせてコードを作成します

本資料について

資料名：要素技術とモデルを開発に使おう: 要素技術をシステムに組込もう（技術教育資料）

この文書は、技術教育「要素技術とモデルを開発に使おう」に使用するETロボコン公式トレーニングのテキストです。

2.3, 2019-04-12 16:47:10, 2019年用