組み込み機器制御プログラミングのための フレームワークの提案と 学生実験用機器への実装

丹治 昭夫[1,a)]　丸山 一貴[,b)]

概要：組み込み機器制御プログラムにおいては、各部品の仕様に基づいたプログラミングが求められる。一方で、これらはミクロな視点であり、全体の構成上の機能としての部品同士の関係性や意味を把握するには困難である。この問題を解決するため、我々はハードウェアの構成要素をオブジェクトと捉え、オブジェクト指向の考え方に基づくフレームワークを実装してモデル化し、ミクロな視点を隠蔽することで、学習者が制御プログラムの学習時に、装置の構成をイメージした実装ができると考えた。この考えに基づいて、Raspberry Pi 3 を中心に、2 つのサーボモーターとレーザーポインタで構成した図形描画装置を使用し、実際に物を動かす組み込み機器の制御プログラミングを実験的に検証する仕組みを作成した。使用言語は Java とし、ハードウェア構成とオブジェクト指向プログラミングの関係性や、イベントドリブンのスタイルなどを検証することができる。

キーワード：Raspberry Pi 3，組み込み制御，プログラミング実験，Java

1. はじめに

近年 IoT に関する関心が高まっており，話題になることもしばしばある．それに伴い自らが IoT 機器を作成する際に便利なマイコンボードやライブラリも充実してきている. しかしながら、これらを活用してすぐに IoT 機器を思い通りに実現できるかというと、それもまた難しくハードルが高いのも事実である。よってできるだけ単純な装置によって仕組みを理解し、プログラムとハードウェアの関係性を学ぶ必要がある。

また、ハードウェア制御のプログラミングの題材としては動きを伴う物が面白さという面で優位性があるが、実験の対象者が多くなると、ロボットカーのようなものだと、動作させるための場所や暴走時の安全性、電源の確保、試行錯誤とプログラム修正作業の効率という点で問題となることがある。

そこで、卓上で完結しつつ動きがあり、動作状況の観察や、成果物の記録を残しやすいものとして、レーザー光による描画装置を考案し、ハードウェア制御プログラムの基本的な必要事項を実験できる装置として作成した。

さらに今回は、学習の対象を主にプログラミング側とし、オブジェクト指向の考え方を取り入れて、機器を構成する要素をオブジェクトと対応させてモデル化したフレームワークを準備すること

[1] 有限会社 デンパン
[2] 明星大学 情報学部
[a)] tech@den.la
[b)] kazutaka.maruyama@meisei-u.ac.jp

で、学習者が細かな部品レベルの設定や操作を意識することなく、装置の構成をイメージしながらプログラムできる環境を用意し、結果の検証が容易で成果に結びつけやすい環境を構築することとした。

2. オブジェクト指向とハードウェア

組み込み機器のプログラミングを行う上では扱うハードウェアの知識や表 1 に示すような低階層のプログラミングが欠かせない。このことから、教育や学習の場面でも比較的早い段階でこれらのことに言及するか、定型句であるがごとく機械的な記述として丸暗記させてしまいがちになっている。しかし、これらのプログラミングはテクニカルな記述となるため、ある程度のプログラム言語の素養がないと、何をしているか全然イメージできず思考停止に陥り、目的を見失ったり、挫折したりする要因になっている。

表 1 低階層のプログラミング知識の要素

知識の要素	内容
I/O の設定	入出力ピンの用途に応じた各種設定
通信プロトコル	通信の仕様に従ったデータ転送手順
正論理と負論理	信号レベルの H/L のどちらが有効か
回路構成	ハードウェアのモジュールで必要な信号の種類など
データシート	構成部品の特性の表やグラフの意味の把握
チャタリング対策	入力における信号のバタつきの抑制と処理

一方で、学習の初期段階では、装置を手で動かす代わりにソフトウェアでコントロールするイメージを持たせ、装置全体をどのように動かすかが重要なポイントとなるはずである。

そこで、ハードウェア寄りの低階層プログラミング部分を隠蔽してハードウェアの各モジュールとの対象性を確保した抽象化をすることにより、学習者の視点のフォーカスが合わせられると考えた。これはまさに「オブジェクト指向」であり、実際のハードウェアのモジュールがソフトウェアでのオブジェクトとして位置づけられ、ハードウェアとソフトウェアの連携をイメージさせる物としても最適である。

さらに副次的な効果として、

- 同種のハードウェアモジュールを追加した際にソフトウェア側でもオブジェクトの追加で対応できる拡張性
- オブジェクト間の独立性によるマルチタスク化
- ハードウェアとソフトウェアの対象性の良さによるメンテナンスの平易化
- モジュール化に伴う再利用のしやすさと標準化

というメリットも生まれる。

3. Raspberry Pi 3

1 章に述べた装置を実現する組み込み用のマイコンとして、Raspberry Pi 3 を採用した (図 1, 図 2)。

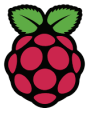

図 1 Raspberry Pi のロゴ

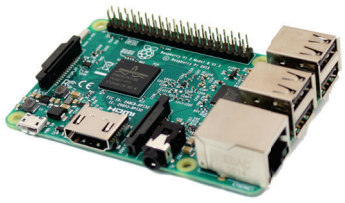

図 2 Raspberry Pi 3 のボード

このマイコンは、以下の特徴を備えている。

- LinuxOS が準備されている
- 外部インターフェースが充実している
- 比較的ポピュラーで資料が多い
- 外部ハードウェアの作成が容易である
- プログラミング言語が選べる

これらの特徴は、一般的な PC のようにプログラミングがしやすく、IC 単体としてのマイコンの

ようにオリジナルのハードウェアを制御しやすいという両者の利点を兼ね備えているので、今回のような用途に適している。

4. レーザー光描画装置

今回の提案を実装するために作成した学習用機器は、レーザー光を板状の平面に照射し、その軌跡を制御することにより図形を描画する「レーザー光描画装置」である。ハードウェア構成は、表 2 および図 3 の通りである。また、装置の外観を図 4 に示す。ここでは図 3 と図 4 の機能的なレイアウトの対称性に注意されたい。

表 2 レーザー光描画装置の構成要素

要素	分類
マイコンボード 「Raspberry Pi 3」	処理装置
ラズベリーパイ I/F ボード LED3 LED2 LED1 LED0 （汎用出力）	インターフェース
描画ユニット サーボモーター X 軸 サーボモーター Y 軸 レーザーポインター	出力装置
マニピュレーター ポテンショメーター X ポテンショメーター Y SW1 SW2	入力装置

5. ソフトウェアのフレームワーク

次にソフトウェアでの制御を行うための実装を考える。ハードウェアを構成する部品やモジュールはオブジェクトとみなせるので、図 3 はオブジェクト図と見ることもできる。また、同種の部品は同じ機能となるのでクラスとして定義する。

クラス化されたモジュールはハードウェア構成と同じになるようにインスタンス化させる必要があるが、その部分について今回提案するフレームワークでは、GOF のデザインパターンでいう “builder pattern” をもとにして階層化し、装置全体をコンポーネント化したうえで、1 つのインスタンスとして参照できるような実装とした。

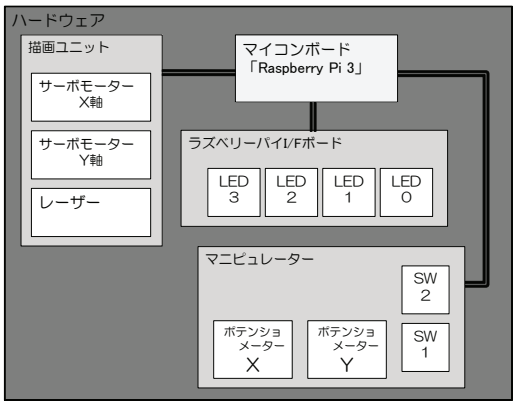

図 3 ハードウェア構成のブロック図

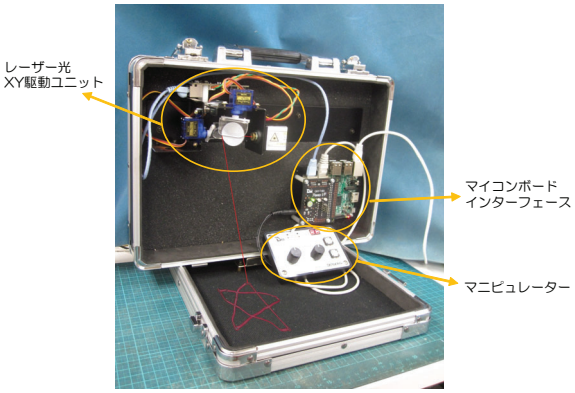

図 4 レーザー光描画装置の外観
(画像合成によるレーザー光の演出あり)

これにより、ハードウェアモジュールと同様の名称で制御プログラムを記述することができるようになる。

作成したフレームワークの記述面での利点を示すために、コード例の図 5 と図 6 にて比較する。

まず、大きな違いとして挙げられる点は初期化部分である。図 5 ではハードウェアの細かな設定部分をユーザーサイドにて記述しているが、図 6 ではフレームワークで実装済みにしているので隠蔽化されており、記述ミスや思考の妨げを適切に回避できている。

次に、出力部分の記述であるが、図 5 では信号出力方式を意識したメソッドを使用し、ハードウェ

```
import com.pi4j.io.gpio.*;
import com.pi4j.wiringpi.Gpio;

//サーボの初期化部分
 Gpio.pwmSetMode( Gpio.PWM_MODE_MS );
 Gpio.pwmSetRange( 2000 );
 Gpio.pwmSetClock( 48 ); //19.2MHz / 48 = 400kHz

 servoX = gpio.provisionPwmOutputPin( SERVO_X_PIN );
 servoY = gpio.provisionPwmOutputPin( SERVO_Y_PIN );

//サーボへの出力部分
 servoX.setPwm( pwmValueX + OFFSET_X );
 servoY.setPwm( pwmValueY + OFFSET_Y );
         //(以下省略)
```

図 5　フレームワークなしのソースコード例

```
import hardware.*; //実機使用時
//import stub.*; //シミュレータ使用時

//ハードウェアのインスタンスを取得する
 private Hardware hardware = Hardware.getInstance();

//サーボへの出力部分
 hardware.drawUnit.servoX.setValue( valueX );
 hardware.drawUnit.servoY.setValue( valueY );
         //(以下省略)
```

図 6　フレームワークを活用したソースコード例

アの特性上の設定であるオフセット値の追加をしているが、図 6 では単に出力したい値をセットするという、より平易な表現になっている。

また、モジュールが階層化された実装になっていることにより、どの部分のモジュールに含まれているものかということがソースコード上に表れることから、ハードウェアの構成がイメージしやすいう利点も生まれている。

フレームワーク化したことによる利点の 2 つ目として、ハードウェアの状態の監視を適切に行えるという点が挙げられる。

通常の記述方法では、次に挙げる点により、効率が落ちる場合がある。

- 値を取りに行ってからハードウェア処理をするので、A/D 変換などでは処理待ちが生じる
- 入力装置において、値の変化が生じたかどうかを絶えず監視する必要がある

今回のフレームワークでは、以上の点に対応するように図 7 に示すような実装になっている。

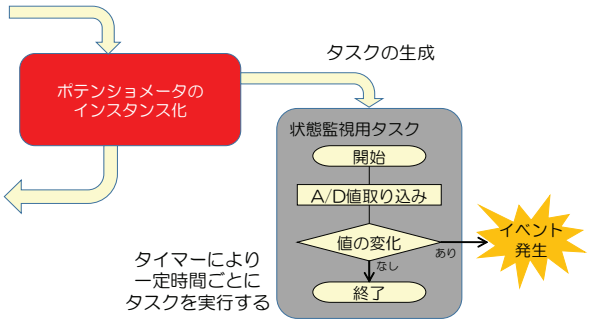

図 7　ハードウェアの状態監視を適切にする実装方法

この実装では、状態監視用タスクを立ち上げ、バックグラウンドで A/D 変換に必要な処理時間ごとに値を更新している。これにより、メイン処理からの値の取得時には、タスクに保存されている値を瞬時に返すことよって処理待ちを軽減している。また、状態に変化が生じたときにはタスク側からイベントを発生させることにより、メイン処理側を状態監視処理から解放している。

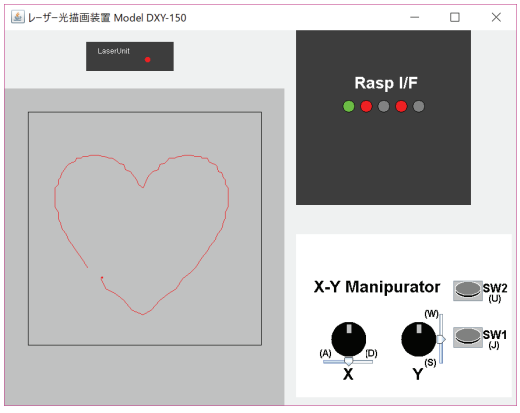

図 8　シミュレーターの画面

フレームワーク化したことによる利点の 3 つ目として、シミュレーターを作成できたという点が挙げられる。

ハードウェアモジュールが抽象化されの独立性が高まったことから、ハードウェアの動作をシミュ

レートするソフトウェアを作成し、モジュールごと入れ替えることにより、学習者が実機がない環境でもある程度の動作確認ができる環境を整えることができた。(図 8)

これは、予習や復習の質が高まることにつながるので、大変有益である。

6. 学生実験での運用

本装置は、実際の学生実験で採用し運用している。運用環境の構成を表 3 に示す。

表 3 学生実験での運用環境の構成

マイコンボード	Raspberry Pi 3 クロス開発環境でなくてよい
コンソール	WindowsPC のリモートデスクトップ 他の実験との機材の共用
プログラム開発	BlueJ プレインストール済みの Java の開発環境

表 3 に挙げた “BlueJ”(図 9)は以下の特徴を備えている。

- 教育用に開発された Java の統合開発環境である
- クラス図ライクなプロジェクト画面で分かりやすい
- エディタ上でブロックが色分けされる
- クラスのインスタンス化が マウス操作で できる
- メソッドの実行が マウス操作で できる
- Raspberry Pi(Raspbian)に最初から入っている
- windows 版や Mac 版もあり、予習復習に活用できる

図 9 BlueJ のロゴ

BlueJ で実装したフレームワークのモジュールは、図 10 のようになる。依存関係が分かりやすく表現されているので、学習面でもメンテナンス面でも効果的である。

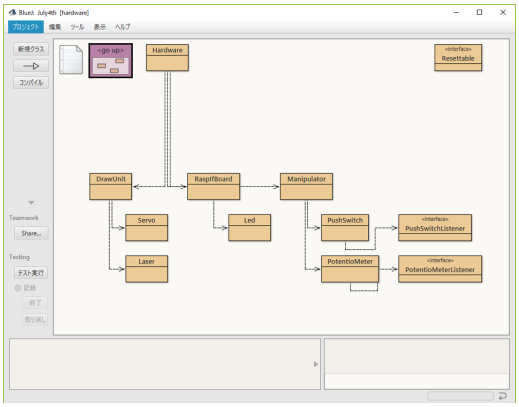

図 10 BlueJ のプロジェクト画面
(フレームワークのハードウェアモジュール)

学生実験で運用している実験手順は、表 4 の通りである。

表 4 学生実験での実験手順

手順(課題)	内容
フレームワークの 使い方	インスタンスの説明 メソッドの使い方 ハードウェア要素の活用方法の確認
キャリブレーション	機械ごとの原点出し 描画範囲限界の設定
正規化	機材の個体差を吸収するための補正
各種図形の描画	座標データによる描画 数式による制御による描画

実験課題にて描画する図形を図 11, 図 12, 図 13 に示す。

図 11 は、設定した描画範囲を確認する図形で、キャリブレーションおよび正規化の項目で実行したものである。図 12 は、数式により座標を決定して描画させる課題のものである。この例で使用している数式は、三角関数で表現できるトロコイド図形である。図 11 は、入力装置のマニピュレーターにて各頂点をプロットし、そのデータ列を順次拾って描画させる課題のものである。

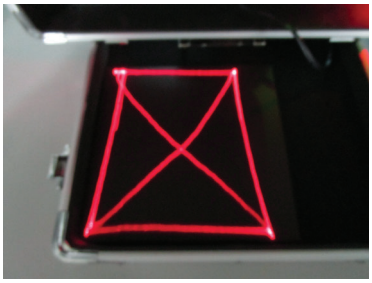

図 11　描画範囲の確認をする図形

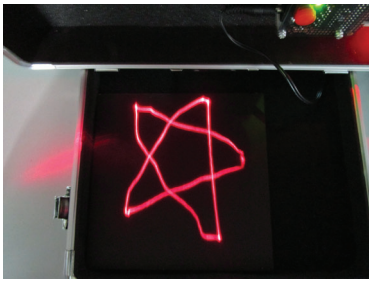

図 12　数式計算による図形

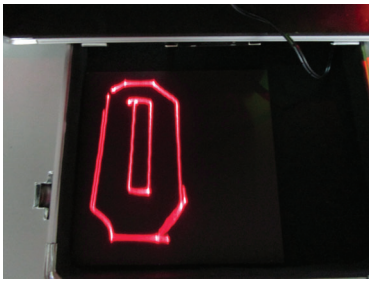

図 13　頂点データの順次読み出しによる図形

これら以外でも、頂点間をなめらかな曲線で補完するベジェ曲線にて描画する図形も確認している。

7.　現時点での課題

作成したレーザー光描画装置とそのフレームワークにおいて、学生実験での運用で見えてきた課題を以下に挙げる。

- サーボモーターの分解能の粗さ
- 描画点の移動速度の遅さ
- オーバーシュートによる描画図形の乱れ
- イベントドリブンの利点についての学習者への訴求不足
- 対象学習者のプログラミングスキルとの乖離

前 3 者については、ハードウェア面での課題であり、より良い部品の選定やソフトウェアドライバでの設定の詰めなどが改善策として挙げられる。

後 2 者については、運用上の課題であり、実験課題の設定方法や実験の時期の工夫が必要となる。

8.　まとめ

本件での提案は、ねらいとして、組み込み機器を制御するためのプログラミングを、学習者が本質的な部分での試行錯誤で習得することにあった。それを実現するための手段として、オブジェクト指向とハードウェアを対応させて、全体の動作が容易にイメージできるようにするため、詳細部分を隠蔽化し抽象化したモジュールでハードウェアをコントロールすることとした。フレームワークを作成して提供することにより、学生実験用の環境として実現させた。

9.　質疑応答

――Question――

Raspberry Pi で開発言語に Java を選択した理由は？（広島市立大学・河野氏）

――Answer――

オブジェクト指向による実現を目指したので、対応している言語が候補となった。その候補として

- Java
- Python
- C++

を挙げて検討した。この中で、Raspberry Pi に開発環境が標準搭載され、GUI にて分かりやすく表示される環境を持つ Java の採用となった。また、イベントドリブンなど学習面で必要な要素の実装や受講者が同実験で回る他のテーマ（Android アプリの開発）との兼ね合いも考慮した。さらに、本装置の開発者自身のスキルによるところも大きい。

参考文献

[1] 江崎徳秀 , 石井モルナ , 内村完爾: **Java で Raspberry Pi 入門**. リックテレコム, 2016.

[2] 中垣健志 , 林満也: **スラスラわかる Java**. 翔泳社, 2014.