

# プロジェクト実習・実験I 組み込みシステム レポート

明星大学 情報学部 情報学科

20J5-161 : 照屋奥介

提出日：2023年1月25日

## 1 実験装置

本実験で使用する装置は次のとおりである。

- レーザー光描画装置（メイン装置、以下「本装置」と呼称する）
- USB メモリ（プログラム格納用）
- ノート PC（VDT 装置<sup>1</sup>としての使用）

本装置についての外観と各部の名称を図 1 に示す。本装置は Linux をベースとした OS を搭載した RaspberryPi を組み込んだ装置であり、Java によって制御されたサーボモーターによって、レーザー光の軌跡を変化させて図形を描画させるものである。

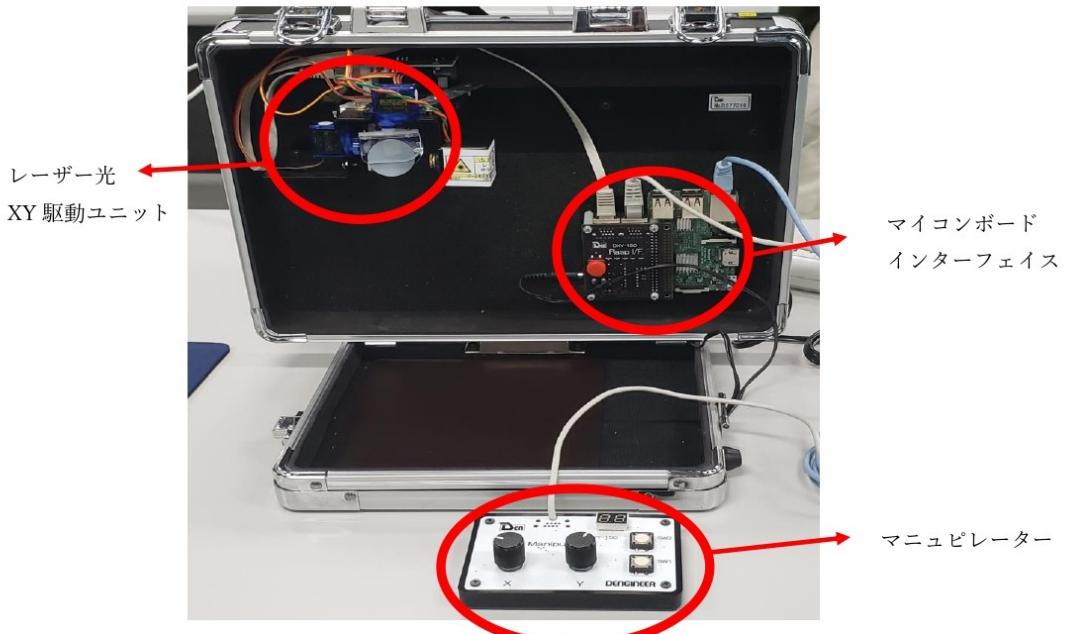


図 1: レーザー光描画装置の外観と各部名称

<sup>1</sup>VDT 装置については各自調べてこの部分に記述すること

## 2 予備実験 1

### 2.1 目的

マニピュレーターの ポテンショメーター X と プッシュスイッチ SW1 の値を読み取り、コンソールに表示させ、使い方の確認をする。

### 2.2 内容

本装置を制御する Java プログラムの開発と基本的な入力装置の動作を確認するため、リスト 1 のプログラムを読み込んで動作させ、入力装置を操作して変化を観察する。

リスト 1: PreKadai1\_PotXandSw1.java

---

```
1
2 /**
3  * クラス PreKadai_PotXandSw1 の注釈をここに書きます.
4 *
5  * @author (照屋奥介)
6  * @version ()
7 */
8
9 //import hardware.*;
10 import stub ..*;
11 public class PreKadai_PotXandSw1
12 {
13     private Hardware hardware = Hardware.getInstance(); //ハードウェアのインスタンス
14     を取得する
15     public void disPotXandSw1(){
16         while(true){
17             int potXvalue = hardware.manipulator.potX.getValue();
18             System.out.print("PotX=" + potXvalue);
19             System.out.print( " " ); //表示での適度なスペースをとる
20             System.out.print( "SW1 = " );
21             if ( hardware.manipulator.sw1.isPush() ) { //sw1 が押されているか調べ
22                 System.out.print( "Pushed" ); //押されていたら「Pushed」と表示す
23             } else {
24                 System.out.print( "Released" ); //押されてなければ「
25                 Released」と表示する
26             }
27             System.out.println(); //改行する
28         }
29     }
30 }
31 }
```

---

### 2.3 結果

ポテンショメーター X とプッシュスイッチ SW1 の操作とそれに対応するコンソールの表示の変化の様子をそれぞれ表 1 と表 2 に示す。また、これらを複合して操作した時でも、それぞれの他方の操作にかかわらず表 1 および表 2 の結果が得られた。

表 1: ポテンショメータ X の操作とコンソールの表示

ポテンショメータ X の操作	コンソールの表示
左に回す	値が減る
左に回しきる	値は 0 になる
右に回す	値が増える
右に回しきる	値は 255 になる

表 2: プッシュスイッチ SW1 の操作とコンソールの表示

プッシュスイッチ SW1 の操作	コンソールの表示
押している	「Pushed」と表示される
離している	「Released」と表示される

## 2.4 考察

表 1 の結果より、ポテンショメータ X から得られる値は 0～255 の数値であるとわかった。

表 2 の結果より、プッシュスイッチ SW1 は変化したときだけでなく、その時の状態を得られることがわかった。

ポテンショメータ X とプッシュスイッチ SW1 の操作は、互いに干渉することなく独立して動作していることが分かった。

## 3 予備実験 2

### 3.1 目的

ポテンショメーター X の値をサーボモーターへの指示値としてサーボモーターを動かす。また、SW1 によりレーザーポインタの光を ON / OFF する。これらにより、レーザー光の動きを確認する。

### 3.2 内容

本装置の出力装置であるレーザー光の動作を確認するため、図 2 に示したフローチャートをもとにリスト 1 のプログラムを作成し、ポテンショメーター X とプッシュスイッチ SW1 を操作して、レーザー光の動きを観察する。

リスト 2: PreKadai2\_ServoXandLaser.java

---

```

1
2 /**
3  * クラス PreKadai2_ServoXandLaser の注釈をここに書きます。
4 *
5  * @author (あなたの名前)
6  * @version (バージョン番号もしくは日付)
7 */
8
9 //import hardware.*; //実験で実機を使用する場合にはコメントを外してこれを使用する
10 import stub.*; //予習などで実機を使用せずに動作を確認する場合には、これを使用する
11 public class PreKadai2_ServoXandLaser
12 {
13
14     private Hardware hardware = Hardware.getInstance(); //ハードウェアのインスタンスを取
15     得する
16     public void moveServoXandLaser() {

```

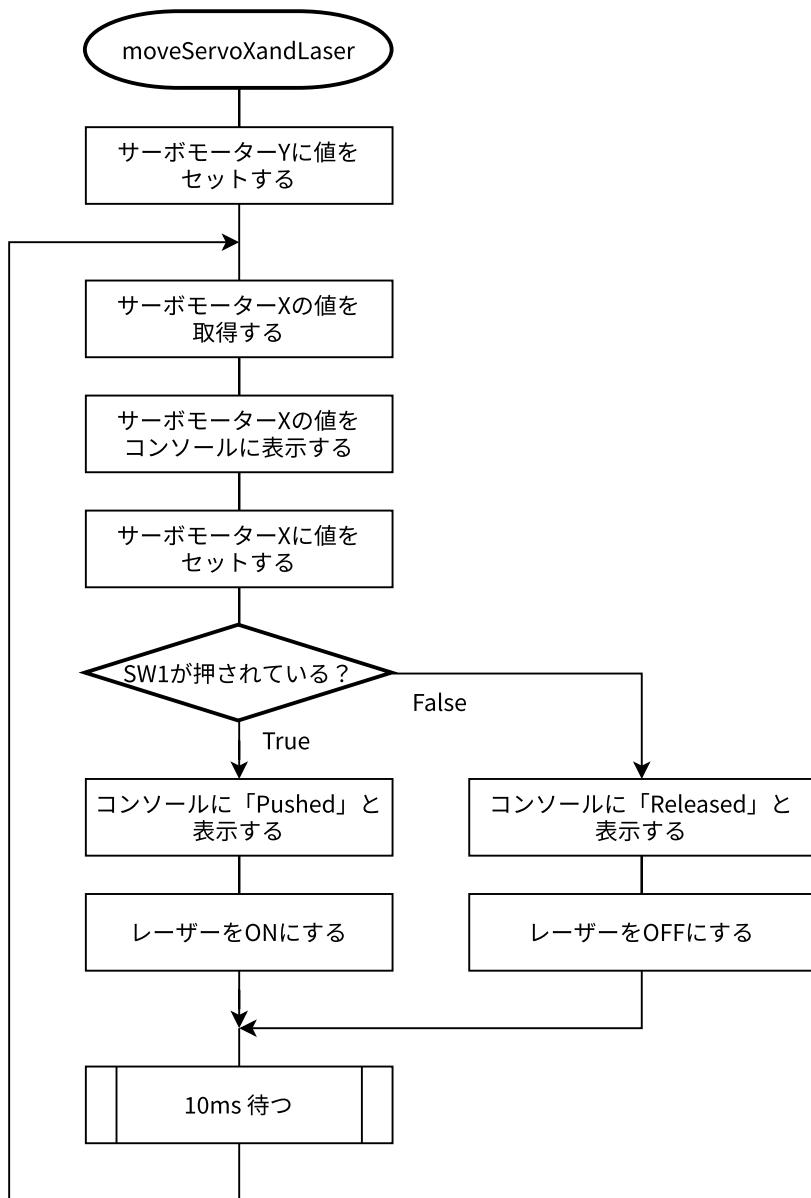


図 2: レーザーポインタの左右移動と ON/OFF のフローチャート

```

16 hardware.drawUnit.servoY.setValue(100); //サーボモーターYにとりあえず
17   の値をセットする。必要に応じて適宜調整する。
18   while ( true ) { //無限に繰り返す
19     int potXvalue = hardware.manipulator.potX.getValue(); //ポテンショメ
20       ターXの値を取得す
21     System.out.print( "PotX = " + potXvalue ); //potXの値を表示する
22     hardware.drawUnit.servoX.setValue( potXvalue ); //サーボモーター
23       Xに値をセットする
24     System.out.print( " " ); //表示での適度なスペースをとる
25     System.out.print( "SW1 = " );
26     if ( hardware.manipulator.sw1.isPush() ) { //sw1が押されているか調べる
27       System.out.print( "Pushed" ); //押されていたら「Pushed」と表示する
         hardware.drawUnit.laser.on(); //レーザーをonにする
     }else {
       System.out.print( "Released" ); //押されてなければ「
         Released」と表示する
  
```

```

28         hardware.drawUnit.laser.off(); //レーザーをoffにする
29     }
30     System.out.println(); //改行する
31     wait_ms( 10 ); //10ms 待つ(サーボモータの仕様上必要)
32   }
33 }
34 private void wait_ms( int waitTime_ms ) { // ms 単位で指定するウェイトメソッド
35   try{
36     Thread.sleep( waitTime_ms );
37   }catch ( Exception e ) {
38     // nothing to do (この課題の用途では処理することがない)
39   }
40 }
41 }

```

---

### 3.3 結果

サーボモーターを時計回りに動かすと、X の値は上昇しレーザーの動きは右方向に進んだ。逆に反時計回りに動かすと X の値は加工しレーザー光の動きは左方向に進んだ。また、SW 1 を押すとレーザー光が点灯し、離すとレーザー光が消灯した。

### 3.4 考察

X の値の変化によってレーザー光の動きが変化することが分かった。

## 4 予備実験 3

### 4.1 目的

ポテンショメーター X , Y を使って X 軸用と Y 軸用のサーボモーターを動かして、レーザー光を自由に動かす。さらに、プッシュスイッチ SW1 を押したらレーザー光を on、SW2 を押したらレーザー光を off、SW1 と SW2 が同時に押されたら、プログラムを終了させる。これらより、レーザー光を縦横自在に動かしたり点灯消灯を制御したり、プログラムを終了させることを確認する。

### 4.2 内容

3 章の予備実験 2 ではレーザー光の動きは横方向 (X 方向) のみであったが、それに縦方向 (Y 方向) の動きも加えることで平面上の任意の点に移動できるようになる。また、プログラムを終了させる条件を追加することにより、強制終了することなく安全に停止させる機能も持たせる。これらを実現する動作のフローチャートを図 3、プログラムをリスト 3 に示し、動作に必要な複合条件について考察する。

リスト 3: PreKadai3\_DrawManual.java

```

1 /**
2  * クラス PreKadai2_ServoXandLaser の注釈をここに書きます。
3  *
4  * @author (あなたの名前)
5  * @version (バージョン番号もしくは日付)
6  */
7
8
9 //import hardware.*; //実験で実機を使用する場合にはコメントを外してこれを使用する

```

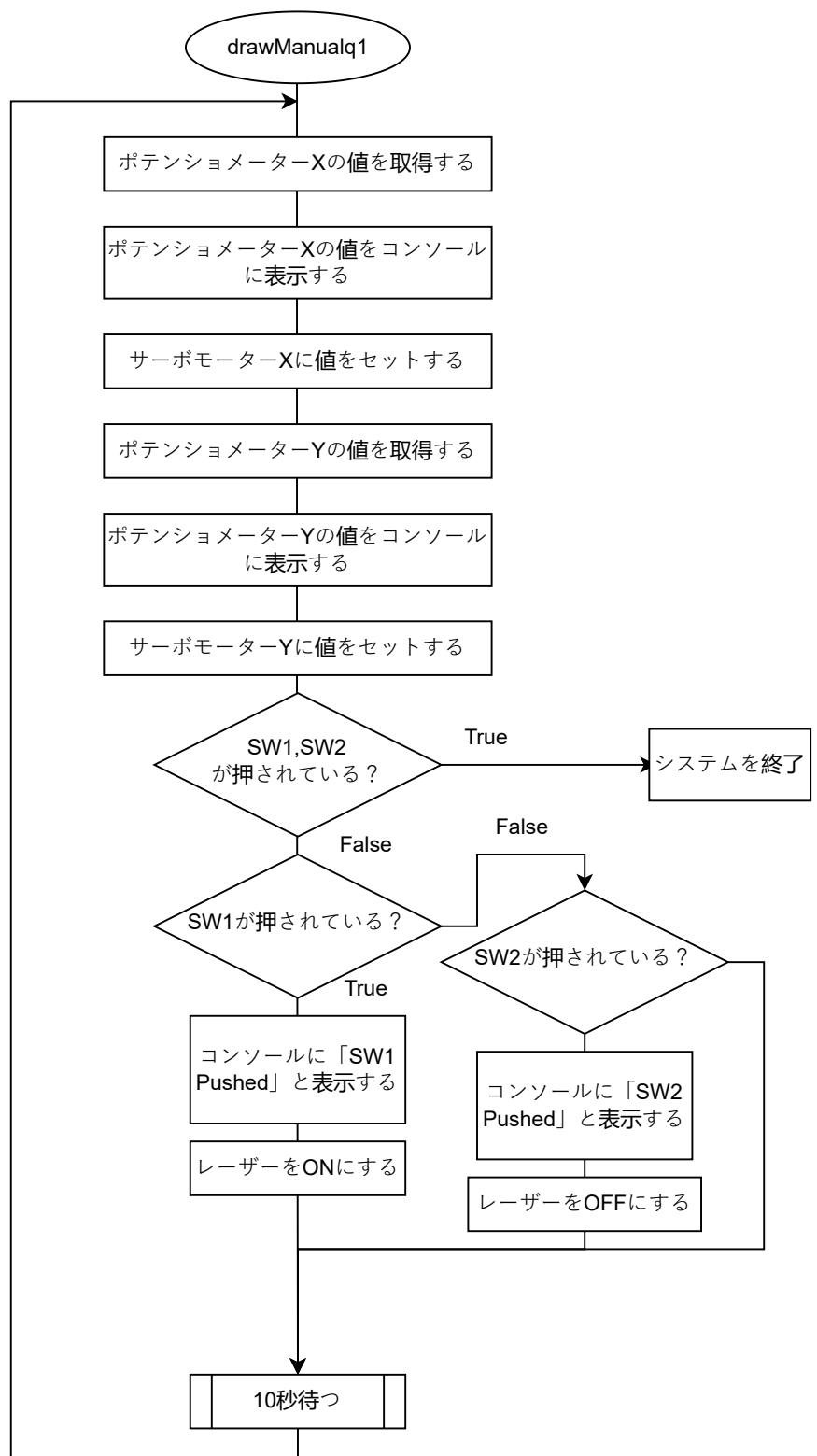


図 3: レーザーポインタの自由移動と ON/OFF のフローチャート

```

10 import stub.*; //予習などで実機を使用せずに動作を確認する場合には、これを使用する
11 public class PreKadai3_DrawManual
12 {
13
14     private Hardware hardware = Hardware.getInstance(); //ハードウェアのインスタンスを取
15         得する
16     public void drawManual() {
17         while ( true ) { //無限に繰り返す
18             int potXvalue = hardware.manipulator.potX.getValue(); //ポテンショメーター
19                 X の値を取得す
20             System.out.print( "PotX = " + potXvalue ); //potX の値を表示する
21             hardware.drawUnit.servoX.setValue( potXvalue ); //サーボモーター
22                 X に値をセットする
23             System.out.print( " " ); //表示での適度なスペースをとる
24             System.out.print( "SW1 = " );
25             if ( hardware.manipulator.sw1.isPush() ) { //sw1 が押されているか調べる
26                 System.out.print( "Pushed" ); //押されていたら「Pushed」と表示する
27                 hardware.drawUnit.laser.on(); //レーザーをon にする
28             }
29             System.out.print( " " ); //表示での適度なスペースをとる
30             int potYvalue = hardware.manipulator.potY.getValue(); //ポテンショメーター
31                 X の値を取得す
32             System.out.print( "PotY = " + potYvalue ); //potX の値を表示する
33             hardware.drawUnit.servoY.setValue( potYvalue ); //サーボモーター
34                 X に値をセットする
35             System.out.print( " " ); //表示での適度なスペースをとる
36             System.out.print( "SW2 = " );
37             if ( hardware.manipulator.sw2.isPush() ){
38                 System.out.print( "Pushed" ); //押されていたら「Pushed」と表示する
39                 hardware.drawUnit.laser.off(); //レーザーをoff にする
40             }
41             if ( hardware.manipulator.sw1.isPush() && hardware.manipulator.sw2.isPush()
42                 ) {
43                 System.out.print( "" );
44                 System.exit( 0 );
45             }
46             else{
47                 System.out.println(); //改行する
48                 wait_ms( 10 ); //10ms 待つ(サーボモータの仕様上必要
49             }
50         }
51     }
52     private void wait_ms( int waitTime_ms ) { // ms 単位で指定するウェイト
53         try{
54             Thread.sleep( waitTime_ms );
55         }catch ( Exception e ) {
56             // nothing to do (この課題の用途では処理することがない)
57         }
58     }
59 }

```

---

### 4.3 結果

ポテンショメーターやプッシュスイッチを操作した時の様子を、表 3 および表 4 に示す。

## 4.4 考察

ポテンショメーターの動作に関しては、XとYを同時に操作した場合は、その操作に応じた斜め方向の動きも観察できたので、2章で確認された結果と同様、それぞれが独立して制御できているとわかった。

プッシュスイッチに関しては、押されたときに所定の動作をすればよいので、離しているときの処理は記述しないでよいとわかった。また、同時に押された時の動作は複合条件判定になるので、条件分岐のif文の条件式は、表5に示した論理演算の論理積を用いることになるとわかった。

## 5 実験1

### 5.1 目的

マニピュレーターのポテンショメーターXまたはプッシュスイッチSW1の値が変化したときの値を読み取り、それをコンソールに表示させる。

### 5.2 内容

マニピュレーターのポテンショメーターのXとSW1の値の変化を確認するため、図??に示したフローチャートをもとにリスト4のプログラムを作成し、ポテンショメーターXとプッシュスイッチSW1を操作する。その時のコンソールに表示される値の変化を観察する。

リスト 4: Kadai1\_PotXandSw1onEvent.java

```
1 import hardware.*; //実験で実機を使用する場合にはコメントを外してこれを使用する
2 // import stub.*; //予習などで実機を使用せずに動作を確認する場合には、これを使用する
3
4
5 public class Kadai1_PotXandSw1onEvent
6 {
7     private Hardware hardware = Hardware.getInstance(); //ハードウェアのインスタンスを取
8     得する
9     public void dispPotXandSw1(){
10         hardware.manipulator.potX.addListener( () -> potXchanged() );
11         hardware.manipulator.sw1.addListener( () -> sw1changed() );
12         while ( true ) { //無限に繰り返す
13             // nothing to do (この課題の用途では処理することがない)
14         }
15     }
16     // ポテンショメータX変化時の処理メソッド
17     private void potXchanged() {
18         int potXvalue = hardware.manipulator.potX.getValue();
19         //ポテンショメーターXの値を取得する
20         System.out.println( "PotX = " + potXvalue ); //potXの値を表示する
21     }
}
```

表 3: ポテンショメーターの操作とレーザー光の動き

ポテンショメーターの操作	レーザー光の動き
Xを左に回す	左に動く。つまみが10時方向あたりでこげ茶色の描画領域からはみ出る。
Xを右に回す	右に動く。つまみが2時方向あたりでこげ茶色の描画領域からはみ出る。
Yを左に回す	手前に動く。つまみが10時方向あたりでこげ茶色の描画領域からはみ出る。
Yを右に回す	奥に動く。つまみが2時方向あたりでこげ茶色の描画領域からはみ出る。

表 4: プッシュスイッチの操作とその時の動作

プッシュスイッチの操作	観測できた動作
SW1だけを押す	レーザー光が点灯し、ポテンショメータで操作した点が見えるようになる。
SW2だけを押す	レーザー光が消灯し、点が見えなくなる。
SW1とSW2を両方押す	プログラムが終了する。

表 5: sw1 と sw2 の同時押しを判定する条件の論理演算の真理値表

論理積

(Java での表記: &&)

条件 1	条件 2	結果
true	true	true
true	false	false
false	true	false
false	false	false

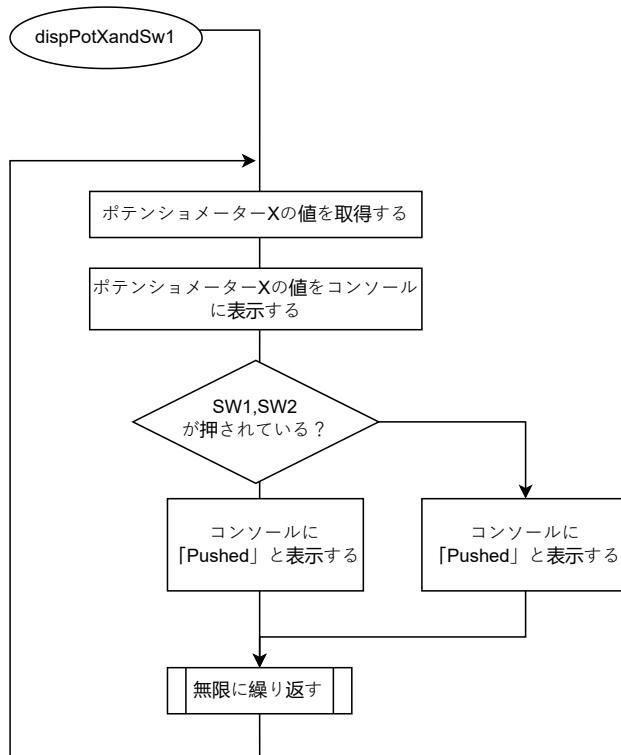


図 4: レーザーポインタの自由移動と ON/OFF のフローチャート

```

22
23 // プッシュスイッチSW1 変化時の処理メソッド
24 private void sw1changed() {
25     System.out.print( "SW1 = " );
26     if ( hardware.manipulator.sw1.isPush() ) { //sw1 が押されているか調べる
27         System.out.println( "Pushed" ); //押されていたら「Pushed」と表示する
28     } else {
29         System.out.println( "Released" ); //押されてなければ「Released」と表示する
  
```

```
30      }
31  }
32 }
```

---

### 5.3 結果

- 予備課題1ではX,Yの値が連続的に変化なしの場合でもコンソールに無限に表示されていた。
- 課題1では、XまたはSW1の値の変化が起こるときのみコンソールに連続的に表示されていた。

### 5.4 考察

結果より、予備課題1の時よりコンソールに表示される値の数が圧倒的に少なくなった。これは、イベントドリブンという処理方法を用いたことで、Xの値が変化するときのみにコンソールに表示されるようになるからであると考える。

## 6 実験2

### 6.1 目的

レーザー光をonにして、ポテンショメーターX,Yを使ってX軸用とY軸用のサーボモーターを動かして、レーザー光を動かす。その時、レーザー光が描画領域からはみ出さないようにするために、サーボモーターのXとYのそれぞれの最大値を割り出す。また、プッシュスイッチSW1を押したらサーボモーターX,Yへの指示値をコンソールに表示するさらに、SW2が押されたら、プログラムを終了させるようにする。

### 6.2 内容

イベントドリブンの仕組みを活用しレーザー光を動かすリストリスト5のプログラムを作成し、実機を用いて動作を確認する。また、レーザー光が描画領域からはみ出さないようにするために、サーボモーターXとYのそれぞれの最小値と最大値を割り出す。そしてSW2を押したらプログラムが終了することも確認する。

リスト 5: Kadai2\_SetDrawArea.java

---

```
1
2 import hardware.*; //実験で実機を使用する場合にはコメントを外してこれを使用する
3 // import stub.*; //予習などで実機を使用せずに動作を確認する場合には、これを使用する
4
5 public class Kadai2_SetDrawArea
6 {
7     private Hardware hardware = Hardware.getInstance(); //ハードウェアのインスタンスを取
    得する
8     public void setValueUnlimited(){
9         hardware.manipulator.potX.addListener( () -> potXchanged() );
10        hardware.manipulator.sw1.addListener( () -> sw1changed() );
11        hardware.manipulator.potY.addListener( () -> potYchanged() );
12        hardware.manipulator.sw2.addListener( () -> sw2changed() );
13        while ( true ) { //無限に繰り返す
14            // nothing to do (この課題の用途では処理することがない)
15        }
16    }
```

```

17 // ポテンショメータX 変化時の処理メソッド
18 private void potXchanged() {
19     int potXvalue = hardware.manipulator.potX.getValue();
20     //ポテンショメーターX の値を取得する
21     System.out.println( "PotX = " + potXvalue ); //potX の値を表示する
22     hardware.drawUnit.servoX.setValue( potXvalue ); //サーボモーターX に値をセットする
23     System.out.print( " " ); //表示での適度なスペースをとる
24 }
25
26
27 private void potYchanged() {
28     int potYvalue = hardware.manipulator.potY.getValue();
29     //ポテンショメーターY の値を取得する
30     System.out.println( "PotY = " + potYvalue ); //potY の値を表示する
31     hardware.drawUnit.servoY.setValue( potYvalue ); //サーボモーターY に値をセットする
32     System.out.print( " " ); //表示での適度なスペースをとる
33 }
34
35 private void sw1changed() {
36     System.out.print( "SW1 = " );
37     if ( hardware.manipulator.sw1.isPush() ) { //sw1 が押されているか調べる
38         System.out.println( "Pushed" ); //押されていたら「Pushed」と表示する
39         hardware.drawUnit.laser.on();
40     } else {
41         System.out.println( "Released" ); //押されてなければ「Released」と表示する
42         hardware.drawUnit.laser.off(); //レーザーをoff にする
43     }
44 }
45
46 private void sw2changed() {
47     System.out.print( "SW2 = " );
48     if ( hardware.manipulator.sw2.isPush() ) { //sw2 が押されているか調べる
49         System.out.print( "Pushed" );
50         System.exit( 0 );
51     } else {
52         System.out.println( "Released" ); //押されてなければ「Released」と表示する
53         hardware.drawUnit.laser.off(); //レーザーをoff にする
54     }
55 }
56 }

```

---

### 6.3 結果

リストリスト 5 のプログラムを実機で実装した結果、ポテンショメーターの動きどうりにレーザー光を動かすことができた。また、SW1、SW2 も正常に動作した。さらに使用した実験装置の描画領域の限界値を調べた結果を表 6 に示す。

### 6.4 考察

イベントドリブンの処理を使用することで、処理の効率が上がることが理解できた。また、使用する実験機材によって描画領域の限界値が異なることが分かった。

表 6: 使用実験機材の番号と描画領域の限界値

使用実験機材の番号	19
左の限界値	60
右の限界値	215
手前の限界値	45
奥の限界値	205

## 7 実験 3

### 7.1 目的

描画領域の限界値（サーボモーターの可動有効範囲）を設定し、設定した限界値で描画領域をはみ出すことがないか、図形を描いて確認する。

### 7.2 内容

座標データは、左下 → 左上 → 右上 → 右下 → 左下 → 右上 → 左上 → 右下の順で作成する。これらのデータは配列に格納し、描画の際に順番に参照して出力するようにするリストリスト 6 のプログラムを作成し、実機を用いて動作を確認する。また、制限がきちんととかかっていることを検証するために、データを 2 式用意し、1 式目は getMin(), getMax() で得た値で、もう 1 式は 0 と 255 で構成してそれぞれを描画し、結果が同じであることを確認する。

リスト 6: Kadai3\_DrawArea.java

```

1
2 import hardware.*; //実験で実機を使用する場合にはコメントを外してこれを使用する
3 //import stub.*; //予習などで実機を使用せずに動作を確認する場合には、これを使用する
4
5 public class Kadai3_CheckDrawArea
6 {
7     private Hardware hardware = Hardware.getInstance(); //ハードウェアのインスタンスを取
    得する
8     private int[] xA = new int[8]; //getMin(),getMax()での座標データ配列
9     private int[] yA = new int[8]; //getMin(),getMax()での座標データ配列
10    // 左下 左上 右上 右下 左下 右上 左上 右下
11    private int[] xB = { 0, 0, 255, 255, 0, 255, 0, 255};
12    private int[] yB = { 0, 255, 255, 0, 0, 255, 255, 0};
13    private int[] drawX; //描画用データ配列(X 座標)
14    private int[] drawY; //描画用データ配列(Y 座標)
15    public void checkDrawArea() {
16        hardware.manipulator.sw1.addListener( () -> selectDataA() );
17        hardware.manipulator.sw2.addListener( () -> selectDataB() );
18        hardware.drawUnit.servoX.setMinMax( 60, 215); //X 座標の最小値と最大値を入れる
19        hardware.drawUnit.servoY.setMinMax( 45, 205); //Y 座標の最小値と最大値を入れる
20
21        int xMin = hardware.drawUnit.servoX.getMin(); //X 座標の最小値を取得する
22        int xMax = hardware.drawUnit.servoX.getMax(); //X 座標の最大値を取得する
23        int yMin = hardware.drawUnit.servoY.getMin(); //Y 座標の最小値を取得する
24        int yMax = hardware.drawUnit.servoY.getMax(); //Y 座標の最大値を取得する
25
26        xA[0] = xMin; //左下 //座標データの格納
27        yA[0] = yMin;
28        xA[1] = xMin; //左上
29        yA[1] = yMax;

```

```

30         xA[2] = xMax; //右上
31         yA[2] = yMax;
32         xA[3] = xMax; //右下
33         yA[3] = yMin;
34         xA[4] = xMin; //左下
35         yA[4] = yMin;
36         xA[5] = xMax; //右上
37         yA[5] = yMax;
38         xA[6] = xMin; //左上
39         yA[6] = yMax;
40         xA[7] = xMax; //右下
41         yA[7] = yMin;
42
43     selectDataA(); // 描画するデータにxA[],yA[]を選択する
44     hardware.drawUnit.laser.on();
45     while ( true ) {
46         for ( int i=0 ; i<drawX.length ; i++ ) {
47             hardware.drawUnit.servoX.setValue( drawX[i] );
48             hardware.drawUnit.servoY.setValue( drawY[i] );
49             UtilityTools.wait_ms( 2000 ); //サーボモーターの移動を考慮したウェイト
50         }
51     }
52 }
53
54 private void selectDataA() {
55     drawX = xA;
56     drawY = yA;
57     hardware.raspIfBoard.setLed( 0x08 );
58     for ( int i=0 ; i<drawX.length ; i++ ) {
59         System.out.printf( "(%3d,%3d)" , drawX[i] , drawY[i] );
60     }
61     System.out.println();
62 }
63
64 private void selectDataB() {
65     drawX = xB;
66     drawY = yB;
67     hardware.raspIfBoard.setLed( 0x01 );
68     for ( int i=0 ; i<drawX.length ; i++ ) {
69         System.out.printf( "(%3d,%3d)" , drawX[i] , drawY[i] );
70     }
71     System.out.println();
72 }
73 }

```

---

### 7.3 結果

getMin() と getMax() で得た式で実行すると描画領域に収まったが 0,255 で構成された式は描画距離からはみ出てしまった。

### 7.4 考察

結果から、機材によって描画領域の限界値に違いがあるため、あらかじめ変数を用意して実装することで違いが生まれないようにできると考える。

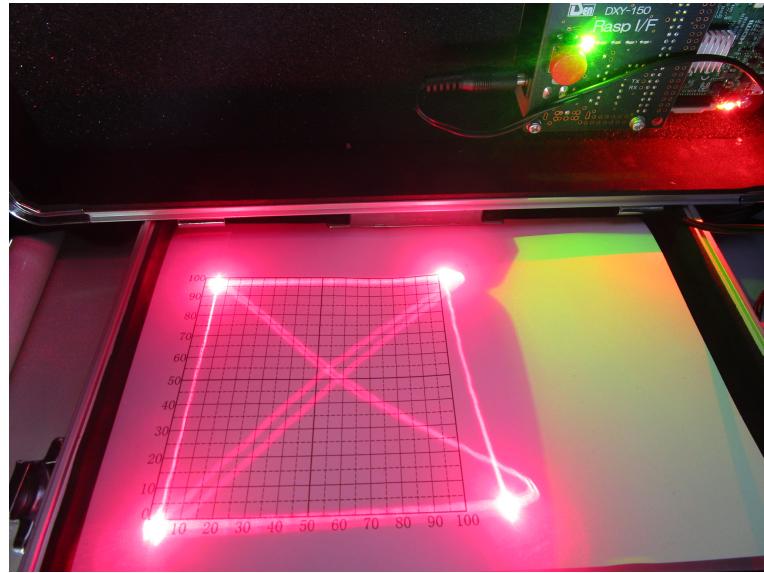


図 5: 課題 3 の図形

## 8 実験 4

### 8.1 目的

使用している機械が違っても、描画領域への座標指定の数値は同じ値で指定できるように、共通化できる仕組みを実装する。データの指定方法は、描画領域の左下の座標を(0,0)、右上の座標を(100,100)とし、内部は線形とする。

### 8.2 内容

1つの座標データを { x 座標 , y 座標 , レーザー光の On/Off } という配列にまとめ、その座標データをさらに配列とするリストリスト 7 のプログラムを作成し Figure クラスとする。この Figure クラスをリストリスト 8 のプログラムを作成し実機を用いて実行する。

リスト 7: ./Figure.java

---

```

1 public class Figure
2 {
3     static final int ON = 1; // 「ON」
4     static final int OFF = 0; // 「OFF」
5
6     static int[][] drawAreaCheck = { //
7         { 0, 0, ON }, // 左下
8         { 0, 100, ON }, // 左上
9         { 100, 100, ON }, // 右上
10        { 100, 0, ON }, // 右下
11        { 0, 0, ON }, // 左下
12        { 100, 100, ON }, // 右上
13        { 0, 100, ON }, // 左上
14        { 100, 0, ON }, // 右下
15    };
16
17     static int[][] sample = { //
18         { 15, 40, OFF }, // 0
19         { 65, 85, ON }, // 
20         { 80, 70, ON }, //

```

```

21      { 15, 40, ON } , //
22      { 45, 50, OFF } , //
23      { 45, 10, ON } , //
24      { 60, 10, ON } , //
25      { 60, 55, ON } , //
26  };
27 }

```

---

リスト 8: ./source/Kadai4\_CheckDrawArea2.java

```

1 import hardware.*; //実験で実機を使用する場合にはコメントを外してこれを使用する
2 //import stub.*; //予習などで実機を使用せずに動作を確認する場合には、これを使用する
3
4 public class Kadai4_CheckDrawArea2
5 {
6     private Hardware hardware = Hardware.getInstance(); //ハードウェアのインスタンスを取
7     得する
8
9     public void checkDrawArea() {
10         System.out.println( "Kadai 4 Start!" );
11         hardware.manipulator.sw2.addListener( () -> systemExit() );
12
13         hardware.drawUnit.servoX.setMinMax(68 , 202); //X 座標の最小値と最大値を入れる
14         hardware.drawUnit.servoY.setMinMax(57 , 177); //Y 座標の最小値と最大値を入れる
15
16         while ( true ) {
17             for ( int[] point : Figure.drawAreaCheck ) {
18                 hardware.drawUnit.servoX.setValue( convertX( point[0] ) );
19                 hardware.drawUnit.servoY.setValue( convertY( point[1] ) );
20                 if ( point[2] != 0 ) {
21                     hardware.drawUnit.laser.on();
22                 } else {
23                     hardware.drawUnit.laser.off();
24                 }
25                 System.out.println( "(" + point[0] + "," + point[1] + ")" );
26                 UtilityTools.wait_ms( 1000 );
27             }
28         }
29
30         private int convertX( int x ) {
31             int xMin = hardware.drawUnit.servoX.getMin();
32             int xMax = hardware.drawUnit.servoX.getMax();
33             return((xMax-xMin)/100)*x+xMin;
34         }
35
36         private int convertY( int y ) {
37             int yMin = hardware.drawUnit.servoY.getMin();
38             int yMax = hardware.drawUnit.servoY.getMax();
39             return (((yMax-yMin)/100)*y+yMin);
40         }
41
42         private void systemExit() {
43             System.out.println( "Kadai 4 End." );
44             System.exit( 0 );
45         }
46     }

```

---

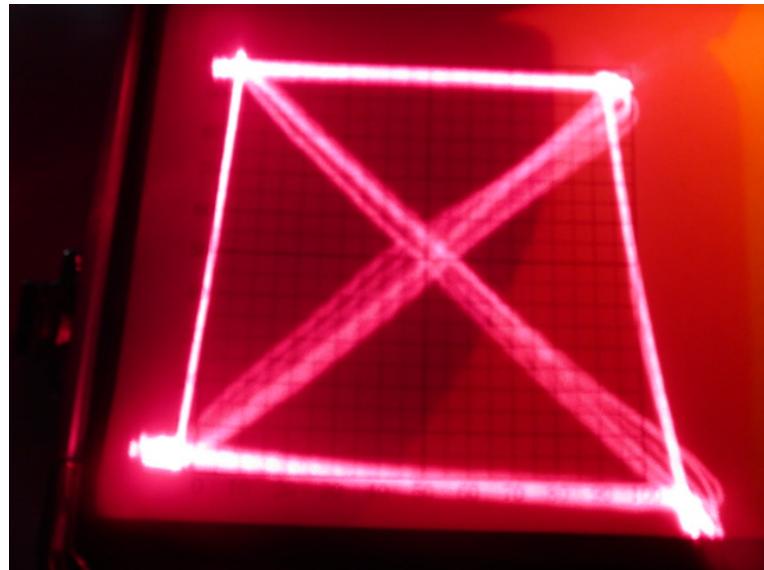


図 6: 課題 4 の図形

### 8.3 結果

正規化データを用いるときの変換式は以下のとおりである。

$$\text{結果} = \beta \cdot \text{原因} + \alpha$$

$$\text{結果} = \frac{xMax - xMin}{100} \cdot x + xMin$$

「原因」は正規化データ、「結果」は実機の指定データを表す

定義域は正規化データの下限値から上限値である

(1)

図 6 に示したように、課題 3 の図と同じ図形を描画することができた。

### 8.4 考察

## 9 実験 5

### 9.1 目的

描かせたい図形を描画領域上でトレースして頂点情報をプロットし、図形データを作成するツールを作成する。また、作成した図形データを確認するツールも併せて作成する。

### 9.2 内容

リスト 9: Kadais\_FigureTraceTool

---

```

1
2 /**
3 * クラス Kadais_FigureTraceTool の注釈をここに書きます.
4 *
5 * @author (あなたの名前)
6 * @version (バージョン番号もしくは日付)
7 */
8 //import hardware.*; //実験で実機を使用する場合にはコメントを外してこれを使用する
9 import stub.*; //予習などで実機を使用せずに動作を確認する場合には、これを使用する

```

```

10
11 public class Kadai5_FigureTraceTool
12 {
13     private Hardware hardware = Hardware.getInstance(); //ハードウェアのインスタンスを取
14     得する
15     private int[] xB = { 0, 0, 100, 100, 0, 100, 0, 100};
16     private int[] yB = { 0, 100, 100, 0, 0, 100, 100, 0};
17
18     public void figureTraceTool(){
19         hardware.drawUnit.servoX.setMinMax(68 , 202); //X 座標の最小値と最大値を入れる
20         hardware.drawUnit.servoY.setMinMax(57 , 177); //Y 座標の最小値と最大値を入れる
21
22         int potXvalue = hardware.manipulator.potX.getValue(); //ポテンショメーター
23         X の値を取得
24         System.out.print("PotX = " + potXvalue ); //potX の値を表示する
25         hardware.drawUnit.servoX.setValue( potXvalue ); //サーボモーターX に値をセットする
26         System.out.print(" " ); //表示での適度なスペースをとる
27         int potYvalue = hardware.manipulator.potY.getValue(); //ポテンショメーター
28         Y の値を取得
29         System.out.print( "PotY = " + potYvalue ); //potX の値を表示する
30         hardware.drawUnit.servoY.setValue( potYvalue ); //サーボモーターY に値をセットする
31         System.out.print( " " ); //表示での適度なスペースをとる
32         System.out.println(); //改行する
33         if ( hardware.manipulator.sw1.isPush() )
34         { //sw1 が押されているか調べる
35             System.out.print( "Pushed" ); //押されていたら「Pushed」と表示する
36             hardware.drawUnit.laser.on(); //レーザーをon にする
37         }else {
38             System.out.print( "Released" ); //押されてなければ「Released」と表示する
39             hardware.drawUnit.laser.off(); //レーザーをoff にする
40         }
41     }
42 }

```

---

### 9.3 結果

結果は図7、図8、図9のようになった。

## 10 実験6

### 10.1 目的

描画領域に、円を描く。

### 10.2 内容

Java の「Math」という数学関数を利用するためのクラスを用いて三角関数を使用し、描画領域に半径40の円を描くリスト10のプログラムを作成し、実機を用いて動作を確認する。

リスト 10: Kadai6\_DrawCircle1.java

---

```

1
2 import hardware.*; //実験で実機を使用する場合にはコメントを外してこれを使用する
3 //import stub.*; //予習などで実機を使用せずに動作を確認する場合には、これを使用する

```

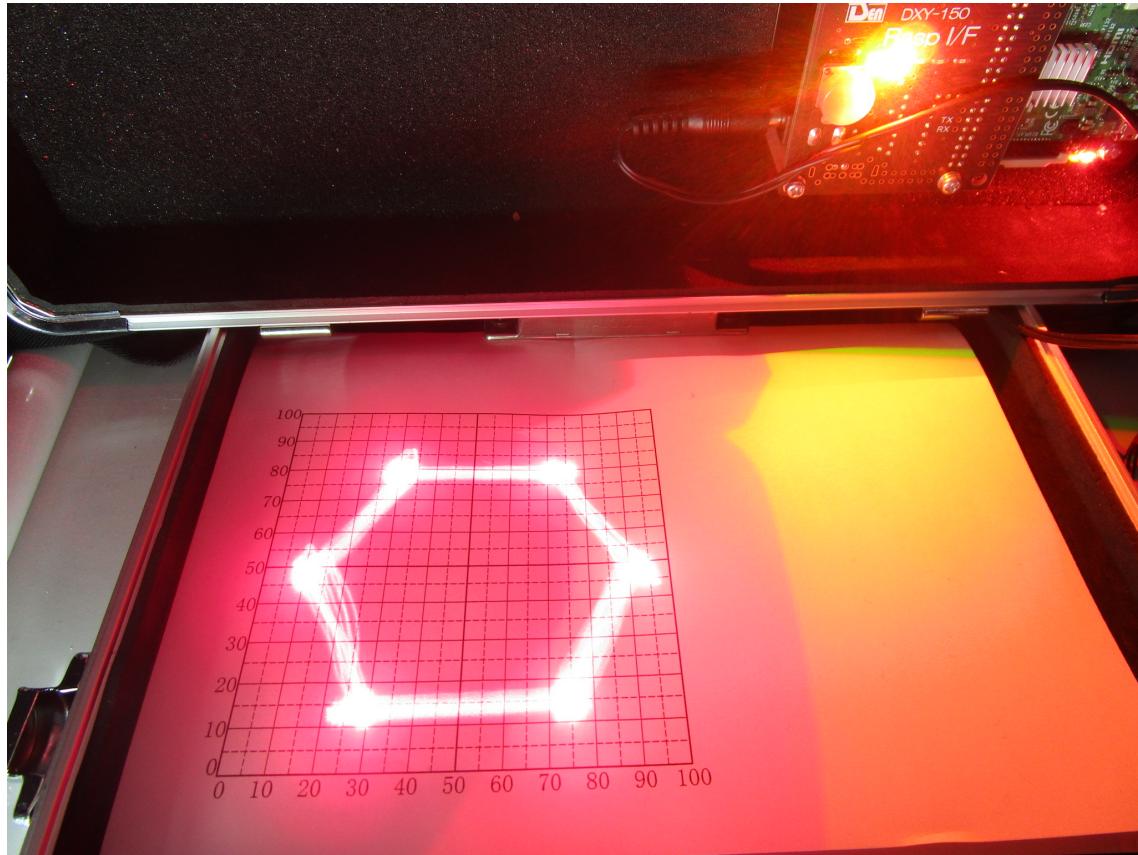


図 7: 課題 5-a

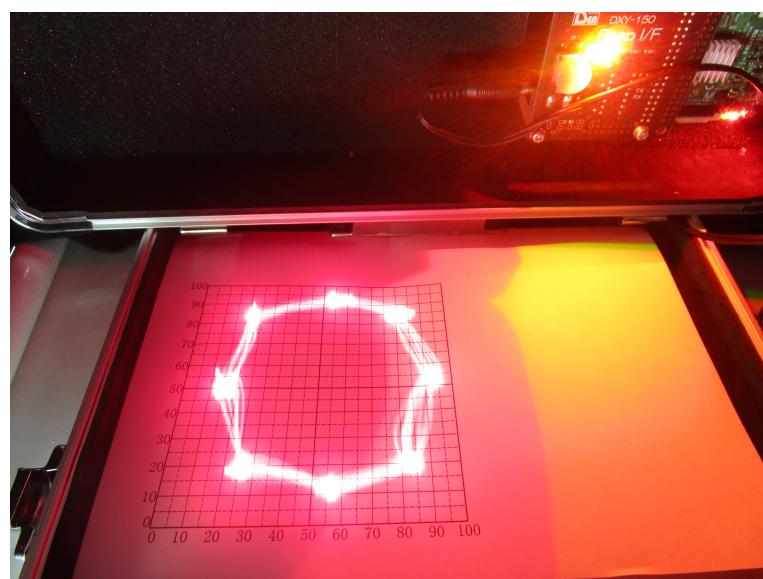


図 8: 課題 5-b

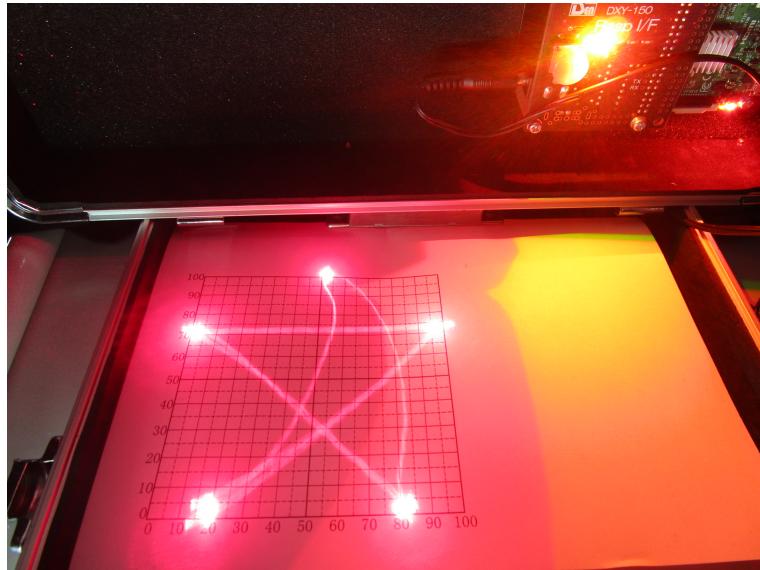


図 9: 課題 5-f

```

4
5 public class Kadai6_DrawCircle1
6 {
7     private Hardware hardware = Hardware.getInstance(); //ハードウェアのインスタンスを取
8     得する
9
10    public void drawFigure() {
11        System.out.println( "Kadai 6 Start!" );
12        hardware.manipulator.sw2.addListener( () -> systemExit() );
13
14        hardware.drawUnit.servoX.setMinMax( 60, 215 ); //X 座標の最小値と最大値を入れる
15        hardware.drawUnit.servoY.setMinMax( 45, 205 ); //Y 座標の最小値と最大値を入れる
16        double r = 40; // 半径 r
17        double x0 = 50; // 中心座標 x0
18        double y0 = 50; // 中心座標 y0
19        hardware.drawUnit.laser.on();
20
21        while ( true ) {
22            for ( int th=0 ; th<360 ; th += 3 ) { //中心角を 3° 間隔で 360° まわす。
23                double radian = Math.toRadians( th );
24                double x = x0 + r * Math.cos( radian );
25                double y = y0 + r * Math.sin( radian );
26                hardware.drawUnit.servoX.setValue( convertX( (int)x ) );
27                hardware.drawUnit.servoY.setValue( convertY( (int)y ) );
28                UtilityTools.wait_ms( 10 );
29            }
30        }
31
32        private int convertX( int x ) {
33            int xMin = hardware.drawUnit.servoX.getMin();
34            int xMax = hardware.drawUnit.servoX.getMax();
35            return ( x+73); //この部分に変換式を記述する
36        }
37
38        private int convertY( int y ) {

```



図 10: 円の図形

```
39     int yMin = hardware.drawUnit.servoY.getMin();
40     int yMax = hardware.drawUnit.servoY.getMax();
41     return (y+70); //この部分に変換式を記述する
42 }
43
44 private void systemExit() {
45     hardware.reset();
46     System.out.println( "Kadai 6 End." );
47     System.exit( 0 );
48 }
49 }
```

---

### 10.3 結果

描画された円は、図 10 のようになった。

### 10.4 考察

結果から Java の「Math」クラスで三角関数を利用することで滑らかな曲線を描けることが分かった。

## 11 実験 7

### 11.1 目的

描画領域に、同心円を描く。

### 11.2 内容

課題 6 を参考に、半径 20 と半径 40 の同心円を描くリスト 11 のプログラムを作成し、実機を用いて動作を確認する。なお、円と円の間は線でつながらないように、きちんと分離させる。

リスト 11: Kadai7\_DrawCircle2.java

---

```
1
2 import hardware.*; //実験で実機を使用する場合にはコメントを外してこれを使用する
3 //import stub.*; //予習などで実機を使用せずに動作を確認する場合には、これを使用する
4
5 public class Kadai7_DrawCircle2
6 {
7     private Hardware hardware = Hardware.getInstance(); //ハードウェアのインスタンスを取
8         得する
9
10    public void drawFigure() {
11        System.out.println( "Kadai 7 Start!" );
12        hardware.manipulator.sw2.addListener( () -> systemExit() );
13
14        hardware.drawUnit.servoX.setMinMax( 60, 215 ); //X 座標の最小値と最大値を入れる
15        hardware.drawUnit.servoY.setMinMax( 45, 205 ); //Y 座標の最小値と最大値を入れる
16        double r1 = 40; //外側の円の半径
17        double r2 = 20; //内側の円の半径
18        double x0 = 50; // 中心座標 x0
19        double y0 = 50; // 中心座標 y0
20        hardware.drawUnit.laser.on();
21        while(true){
22            for ( int th=0 ; th<360 ; th += 3 ) { //中心角を 3° 間隔で 360° まわす。
23                double radian = Math.toRadians( th );
24                double x = x0 + r2 * Math.cos( radian );
25                double y = y0 + r2 * Math.sin( radian );
26
27                hardware.drawUnit.servoX.setValue( convertX( (int)x ) );
28                hardware.drawUnit.servoY.setValue( convertY( (int)y ) );
29                UtilityTools.wait_ms( 10 );
30            }
31            hardware.drawUnit.laser.off(); //レーザーをoff にする
32            for ( int th=0 ; th<360 ; th += 3 ) { //中心角を 3° 間隔で 360° まわす。
33                double radian = Math.toRadians( th );
34                double x = x0 + r1 * Math.cos( radian );
35                double y = y0 + r1 * Math.sin( radian );
36                hardware.drawUnit.servoX.setValue( convertX( (int)x ) );
37                hardware.drawUnit.servoY.setValue( convertY( (int)y ) );
38                UtilityTools.wait_ms( 10 );
39            }
40            hardware.drawUnit.laser.on();
41            for ( int th=0 ; th<360 ; th += 3 ) { //中心角を 3° 間隔で 360° まわす。
42                double radian = Math.toRadians( th );
43                double x = x0 + r1 * Math.cos( radian );
44                double y = y0 + r1 * Math.sin( radian );
45                hardware.drawUnit.servoX.setValue( convertX( (int)x ) );
46                hardware.drawUnit.servoY.setValue( convertY( (int)y ) );
47                UtilityTools.wait_ms( 10 );
48            }
49            hardware.drawUnit.laser.off(); //レーザーをoff にする
50            for ( int th=0 ; th<360 ; th += 3 ) { //中心角を 3° 間隔で 360° まわす。
51                double radian = Math.toRadians( th );
52                double x = x0 + r2 * Math.cos( radian );
53                double y = y0 + r2 * Math.sin( radian );
54                hardware.drawUnit.servoX.setValue( convertX( (int)x ) );
```



図 11: 同心円の図形

```
55         hardware.drawUnit.servoY.setValue( convertY( (int)y ) );
56         UtilityTools.wait_ms( 10 );
57     }
58     hardware.drawUnit.laser.on();
59 }
60 }
61
62 private int convertX( int x ) {
63     int xMin = hardware.drawUnit.servoX.getMin();
64     int xMax = hardware.drawUnit.servoX.getMax();
65     return((xMax-xMin)/100)*x+xMin;
66 }
67
68 private int convertY( int y ) {
69     int yMin = hardware.drawUnit.servoY.getMin();
70     int yMax = hardware.drawUnit.servoY.getMax();
71     return (((yMax-yMin)/100)*y+yMin);
72 }
73
74 private void systemExit() {
75     System.out.println( "Kadai 7 End." );
76     System.exit( 0 );
77 }
78 }
```

---

### 11.3 結果

描画された同心円は、図 11 のようになった。

## 11.4 考察

# 12 実験8

### 12.1 目的

描画領域に、課題6とトロコイドを参考にして、星形や花形などを描く。

### 12.2 内容

トロコイドを参考にして星形や花形を作図するプログラムを作成し、実機を用いて動作を確認する。

リスト 12: Kadai8\_DrawStar.java

```
1 import hardware.*; //実験で実機を使用する場合にはコメントを外してこれを使用する
2 //import stub.*; //予習などで実機を使用せずに動作を確認する場合には、これを使用する
3
4
5 public class Kadai8_DrawStar
6 {
7     private Hardware hardware = Hardware.getInstance(); //ハードウェアのインスタンスを取
8     得する
9
10    public void drawFigure() {
11        System.out.println( "Kadai 8 Start!" );
12        hardware.manipulator.sw2.addListener( () -> systemExit() );
13
14        hardware.drawUnit.servoX.setMinMax( 60, 215 ); //X座標の最小値と最大値を入れる
15        hardware.drawUnit.servoY.setMinMax( 45, 205 ); //Y座標の最小値と最大値を入れる
16        double rc = 50; // 定円の半径
17        double rm =30; // 同円の半径
18        double rd =50; //描画点の半径
19        double x0 = 50; // 中心座標 x0
20        double y0 = 50; // 中心座標 y0
21        hardware.drawUnit.laser.on();
22
23        while ( true ) {
24            for ( int th=0 ; th<360*3 ; th += 3 ) { //中心角を3°間隔で360°まわす。
25                double radian = Math.toRadians( th );
26                double x =x0+ (rc-rm)*Math.cos(radian) + rd * Math.cos((rc-rm)*radian/
27                                rm );
28                double y =y0+ (rc-rm)*Math.sin(radian) - rd * Math.sin((rc-rm)*radian/
29                                rm );
30                hardware.drawUnit.servoX.setValue( convertX( (int)x ) );
31                hardware.drawUnit.servoY.setValue( convertY( (int)y ) );
32                UtilityTools.wait_ms( 10 );
33            }
34        }
35
36        private int convertX( int x ) {
37            int xMin = hardware.drawUnit.servoX.getMin();
38            int xMax = hardware.drawUnit.servoX.getMax();
39            return(((xMax-xMin)/100)*x+xMin); //この部分に変換式を記述する
40        }
41
42        private int convertY( int y ) {
```

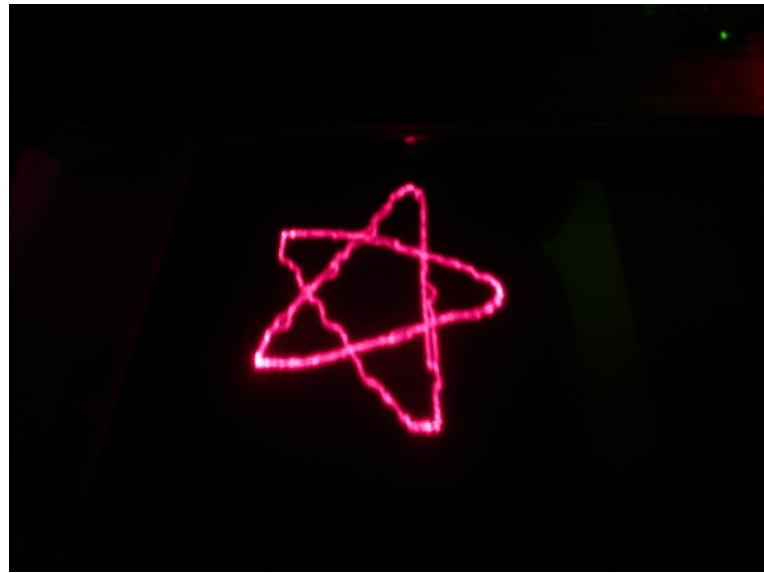


図 12: トロコイドを使った星型のレーザー光描画

```

41     int yMin = hardware.drawUnit.servoY.getMin();
42     int yMax = hardware.drawUnit.servoY.getMax();
43     return (((yMax-yMin)/100)*y+yMin); //この部分に変換式を記述する
44 }
45
46 private void systemExit() {
47     hardware.reset();
48     System.out.println( "Kadai 8 End. " );
49     System.exit( 0 );
50 }
51 }
```

---

### 12.3 結果

リスト 12 で作成したプログラムでは、月を作図することができた。しかし、花形のレーザー光描写を作成には失敗した。

### 12.4 考察

トロコイドにもいくつかの種類が存在し星形は、内側トロコイドで作成することができた。内側トロコイドの式は以下となる。

$$\begin{cases} x = x_0 + (r_c + r_m) \cos \theta - r_d \cos \left( \frac{r_c + r_m}{r_m} \theta \right) \\ y = y_0 + (r_c + r_m) \sin \theta - r_d \sin \left( \frac{r_c + r_m}{r_m} \theta \right) \end{cases} \quad (2)$$

## 13 実験 9

### 13.1 目的

描画領域に、ハート形を描く。

## 13.2 内容

なめらかな自由曲線を描くための数値補間曲線「ベジエ曲線」を用いて補間処理をするクラスリスト 13 を使ってハートを描くリスト 14 のプログラムを作成し、実機を用いて動作を確認する。

リスト 13: ./Bezier.java

```
1  /**
2   * ベジエ曲線
3   *
4   * @author (あなたの名前)
5   * @version (バージョン番号もしくは日付)
6   */
7  import java.util.LinkedList;
8  import java.util.ArrayList;
9
10 public class Bezier
11 {
12     static final int DIVISION = 50; //曲線補間の分割数(曲線の分解能)
13
14     private LinkedList<Point> points = new LinkedList<>(); //曲線の座標列
15
16     public void addPoint( int x, int y ) { //曲線に必要な座標を追加するメソッド
17         points.add( new Point( (double)x , (double)y ) );
18     }
19
20     public ArrayList<int[]> getPoints() {
21         ArrayList<int[]> resultPoints = new ArrayList<>();
22         for( int i=0 ; i<=DIVISION ; i++ ) {
23             Point point = getBezierPointInner( i*(1.0/DIVISION) , points );
24             int[] xyw = { (int)point.x , (int)point.y , 1 };
25             resultPoints.add( xyw );
26         }
27         return ( resultPoints );
28     }
29
30     // ベジエ曲線の補間座標を生成するメソッド(再帰的に実現)
31     private Point getBezierPointInner( double t , LinkedList<Point> ps ) {
32         if ( ps.isEmpty() ) {
33             return ( new Point( 0,0 ) );
34         } else if ( ps.size() == 1 ) {
35             return ( ps.getFirst() );
36         } else {
37             LinkedList<Point> pHeads = new LinkedList<>();
38             pHeads.addAll( ps );
39             pHeads.removeLast();
40             Point p0 = getBezierPointInner( t , pHeads );
41
42             LinkedList<Point> pTails = new LinkedList<>();
43             pTails.addAll( ps );
44             pTails.removeFirst();
45             Point p1 = getBezierPointInner( t , pTails );
46
47             return ( new Point( (1.0-t)*p0.x + t*p1.x , (1.0-t)*p0.y + t*p1.y ) );
48         }
49     }
50
51     private class Point { //(x,y)座標を構成する内部クラス
```

```

52     double x;
53     double y;
54     public Point( double x, double y ) {
55         this.x = x;
56         this.y = y;
57     }
58 }
59 }
```

---

リスト 14: ./source/Kadai9\_DrawHeart.java

```

1
2
3 import hardware.*; //実験で実機を使用する場合にはコメントを外してこれを使用する
4 // import stub.*; //予習などで実機を使用せずに動作を確認する場合には、これを使用する
5
6 import java.util.ArrayList;
7
8 public class Kadai9_DrawHeart
9 {
10     private Hardware hardware = Hardware.getInstance(); //ハードウェアのインスタンスを取
11     得する
12
13     public void drawFigure() {
14         System.out.println( "Kadai 9 Start!" );
15         hardware.manipulator.sw2.addListener( () -> systemExit() );
16
17         hardware.drawUnit.servoX.setMinMax(68 , 202); //X 座標の最小値と最大値を入れる
18         hardware.drawUnit.servoY.setMinMax(57 , 177); //Y 座標の最小値と最大値を入れる
19
20         ArrayList<int[]> drawPoints = new ArrayList<>();
21         Bezier bezier;
22         int offsetX = 50; //図形の中心座標 x
23         int offsetY = 50; //図形の中心座標 y
24
25         bezier = new Bezier(); //1本目の曲線を生成する
26         bezier.addPoint( offsetX - 0 , offsetY - 40 ); //始点の座標 (1)
27         bezier.addPoint( offsetX - 80 , offsetY + 20 ); //制御線の座標 (2)
28         bezier.addPoint( offsetX - 20 , offsetY + 60 ); //制御線の座標 (3)
29         bezier.addPoint( offsetX - 0 , offsetY + 20 ); //終点の座標 (4)
30         drawPoints.addAll( bezier.getPoints() ); //補間した座標列を取得し描画データに追
31         加
32
33         bezier = new Bezier(); //2本目の曲線を生成する
34         bezier.addPoint( offsetX + 0 , offsetY + 20 ); //始点の座標 (5)
35         bezier.addPoint( offsetX + 20 , offsetY + 60 ); //制御線の座標 (6)
36         bezier.addPoint( offsetX + 80 , offsetY + 20 ); //制御線の座標 (7)
37         bezier.addPoint( offsetX + 0 , offsetY - 40 ); //終点の座標 (8)
38         drawPoints.addAll( bezier.getPoints() ); //補間した座標列を取得して描画データに
39         追加
40
41         while ( true ) {
42             for ( int[] point : drawPoints ) {
43                 hardware.drawUnit.servoX.setValue( convertX( point[0] ) );
44                 hardware.drawUnit.servoY.setValue( convertY( point[1] ) );
45                 if ( point[2] != 0 ) {
46                     hardware.drawUnit.laser.on();
```

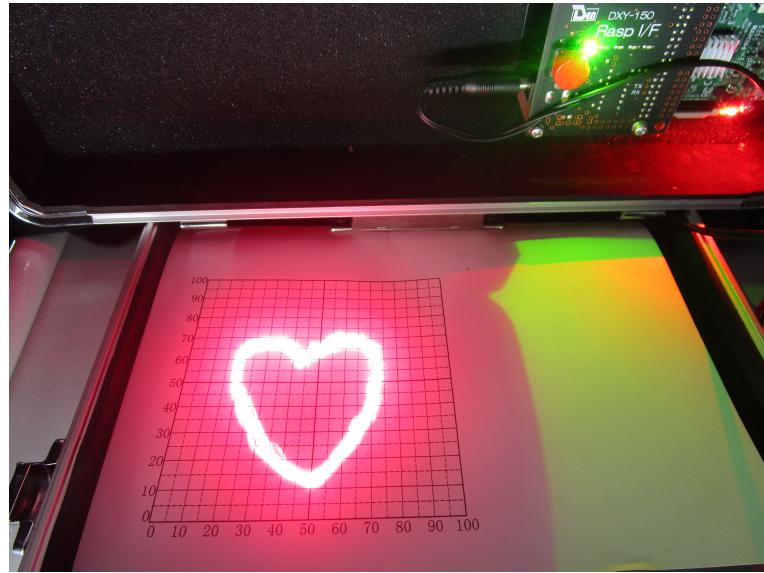


図 13: ハート形のレーザー光描画

```

44         } else {
45             hardware.drawUnit.laser.off();
46         }
47         UtilityTools.wait_ms( 10 );
48     }
49 }
50 }
51
52 private int convertX( int x ) {
53     int xMin = hardware.drawUnit.servoX.getMin();
54     int xMax = hardware.drawUnit.servoX.getMax();
55     return ( x+xMin); //この部分に変換式を記述する
56 }
57
58 private int convertY( int y ) {
59     int yMin = hardware.drawUnit.servoY.getMin();
60     int yMax = hardware.drawUnit.servoY.getMax();
61     return ( y+yMin); //この部分に変換式を記述する
62 }
63
64 private void systemExit() {
65     hardware.reset();
66     System.out.println( "Kadai 9 End." );
67     System.exit( 0 );
68 }
69 }
```

---

### 13.3 結果

2つの曲線から成るハート型は図 13 のようになった。

## 13.4 考察

実際にレーザー光を観察すると、2本の曲線からハート型が成り立つという意味が理解することができた。制御点の座標を変えることで変幻自在に曲線を描けるのではないかと考える。

## 14 実験 10

### 14.1 目的

画領域に、三日月 や 富士山 などを描く。

### 14.2 内容

ベジェ曲線でのレーザー光の描画を利用して様々な形を作図するリスト 15 とリスト 16 のプログラムを作成し、実機を用いて動作を確認する。

リスト 15: Kadai10\_DrawMtFuji.java

```
1  
2 import hardware.*; //実験で実機を使用する場合にはコメントを外してこれを使用する  
3 //import stub.*; //予習などで実機を使用せずに動作を確認する場合には、これを使用する  
4  
5  
6 import java.util.ArrayList;  
7  
8 public class Kadai10_DrawMtFuji  
9 {  
10     private Hardware hardware = Hardware.getInstance(); //ハードウェアのインスタンスを取  
    得する  
11  
12     public void drawFigure() {  
13         System.out.println( "Kadai 10 Start!" );  
14         hardware.manipulator.sw2.addListener( () -> systemExit() );  
15  
16         hardware.drawUnit.servoX.setMinMax(68 , 202); //X 座標の最小値と最大値を入れる  
17         hardware.drawUnit.servoY.setMinMax(57 , 177); //Y 座標の最小値と最大値を入れる  
18  
19         ArrayList<int[]> drawPoints = new ArrayList<>();  
20         Bezier bezier;  
21         int offsetX = 50; //図形の中心座標 x  
22         int offsetY = 50; //図形の中心座標 y  
23  
24         bezier = new Bezier(); //1本目の曲線を生成する  
25         bezier.addPoint( offsetX - 60 , offsetY - 30 ); //始点の座標 (1)  
26         bezier.addPoint( offsetX - 30 , offsetY - 10 ); //制御線の座標 (2)  
27         bezier.addPoint( offsetX - 25 , offsetY + 10 ); //制御線の座標 (3)  
28         bezier.addPoint( offsetX - 20 , offsetY + 30 ); //終点の座標 (4)  
29         drawPoints.addAll( bezier.getPoints() ); //間した座標列を取得し描画データに追加  
30  
31         bezier = new Bezier(); //2本目の曲線を生成する  
32         bezier.addPoint( offsetX - 20 , offsetY + 30 ); //始点の座標 (5)  
33         bezier.addPoint( offsetX - 15 , offsetY + 25 ); //制御線の座標 (6)  
34         bezier.addPoint( offsetX + 15 , offsetY + 25 ); //制御線の座標 (7)  
35         bezier.addPoint( offsetX + 20 , offsetY + 30 ); //終点の座標 (8)  
36         drawPoints.addAll( bezier.getPoints() ); //補間した座標列を取得して描画データに  
    追加
```

```

37
38     bezier = new Bezier(); //2本目の曲線を生成する
39     bezier.addPoint( offsetX + 20 , offsetY + 30 ); //始点の座標 (9)
40     bezier.addPoint( offsetX + 25 , offsetY + 10 ); //制御線の座標 (10)
41     bezier.addPoint( offsetX + 30 , offsetY - 10 ); //制御線の座標 (11)
42     bezier.addPoint( offsetX + 60 , offsetY - 30 ); //終点の座標 (12)
43     drawPoints.addAll( bezier.getPoints() ); //補間した座標列を取得して描画データに
44     追加
45
46     bezier = new Bezier(); //2本目の曲線を生成する
47     bezier.addPoint( offsetX + 60 , offsetY - 30 ); //始点の座標 (13)
48     bezier.addPoint( offsetX + 30 , offsetY - 30 ); //制御線の座標 (14)
49     bezier.addPoint( offsetX - 30 , offsetY - 30 ); //制御線の座標 (15)
50     bezier.addPoint( offsetX - 60 , offsetY -30 ); //終点の座標 (16)
51     drawPoints.addAll( bezier.getPoints() ); //補間した座標列を取得して描画データに
52     追加
53
54     while ( true ) {
55         for ( int[] point : drawPoints ) {
56             hardware.drawUnit.servoX.setValue( convertX( point[0] ) );
57             hardware.drawUnit.servoY.setValue( convertY( point[1] ) );
58             if ( point[2] != 0 ) {
59                 hardware.drawUnit.laser.on();
60             } else {
61                 hardware.drawUnit.laser.off();
62             }
63         }
64     }
65
66     private int convertX( int x ) {
67         int xMin = hardware.drawUnit.servoX.getMin();
68         int xMax = hardware.drawUnit.servoX.getMax();
69         return ( x+xMin ); //この部分に変換式を記述する
70     }
71
72     private int convertY( int y ) {
73         int yMin = hardware.drawUnit.servoY.getMin();
74         int yMax = hardware.drawUnit.servoY.getMax();
75         return ( y+yMin ); //この部分に変換式を記述する
76     }
77
78     private void systemExit() {
79         hardware.reset();
80         System.out.println( "Kadai 10 End." );
81         System.exit( 0 );
82     }
83 }
```

リスト 16: Kadai10\_DrawMoon

---

```

1
2 import hardware.*; //実験で実機を使用する場合にはコメントを外してこれを使用する
3 //import stub.*; //予習などで実機を使用せずに動作を確認する場合には、これを使用する
4
5 import java.util.ArrayList;
```

```

6
7 public class Kadai10_Drawmoon
8 {
9     private Hardware hardware = Hardware.getInstance(); //ハードウェアのインスタンスを取
10    得する
11
12    public void drawFigure() {
13        System.out.println( "Kadai 9 Start!" );
14        hardware.manipulator.sw2.addListener( () -> systemExit() );
15
16        hardware.drawUnit.servoX.setMinMax(68 , 202); //X 座標の最小値と最大値を入れる
17        hardware.drawUnit.servoY.setMinMax(57 , 177); //Y 座標の最小値と最大値を入れる
18
19        ArrayList<int []> drawPoints = new ArrayList<>();
20        Bezier bezier;
21        int offsetX = 50; //図形の中心座標 x
22        int offsetY = 50; //図形の中心座標 y
23
24        bezier = new Bezier(); //1本目の曲線を生成する
25        bezier.addPoint( offsetX - 0 , offsetY -20 ); //始点の座標 (1)
26        bezier.addPoint( offsetX +20 , offsetY - 0 ); //制御線の座標 (2)
27        bezier.addPoint( offsetX +20 , offsetY + 20 ); //制御線の座標 (3)
28        bezier.addPoint( offsetX - 0 , offsetY + 40 ); //終点の座標 (4)
29        drawPoints.addAll( bezier.getPoints() ); //間した座標列を取得し描画データに追加
30
31        bezier = new Bezier(); //2本目の曲線を生成する
32        bezier.addPoint( offsetX + 0 , offsetY + 40 ); //始点の座標 (5)
33        bezier.addPoint( offsetX + 40, offsetY + 20 ); //制御線の座標 (6)
34        bezier.addPoint( offsetX + 40, offsetY + 0 ); //制御線の座標 )
35        bezier.addPoint( offsetX + 0 , offsetY - 20 ); //終点の座標 (8)
36        drawPoints.addAll( bezier.getPoints() ); //補間した座標列を取得して描画データに
37        追加
38
39        while ( true ) {
40            for ( int[] point : drawPoints ) {
41                hardware.drawUnit.servoX.setValue( convertX( point[0] ) );
42                hardware.drawUnit.servoY.setValue( convertY( point[1] ) );
43                if ( point[2] != 0 ) {
44                    hardware.drawUnit.laser.on();
45                } else {
46                    hardware.drawUnit.laser.off();
47                }
48                UtilityTools.wait_ms( 10 );
49            }
50        }
51
52        private int convertX( int x ) {
53            int xMin = hardware.drawUnit.servoX.getMin();
54            int xMax = hardware.drawUnit.servoX.getMax();
55            return ( x+xMin); //この部分に変換式を記述する
56        }
57
58        private int convertY( int y ) {
59            int yMin = hardware.drawUnit.servoY.getMin();

```

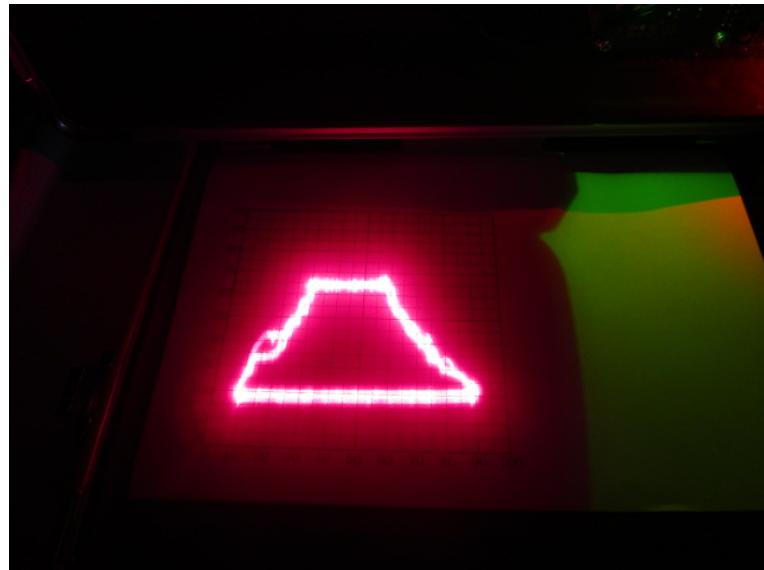


図 14: 富士山型のレーザー光描写

```
60         return ( y+yMin); //この部分に変換式を記述する
61     }
62
63     private void systemExit() {
64         hardware.reset();
65         System.out.println( "Kadai 10 End." );
66         System.exit( 0 );
67     }
68 }
```

---

### 14.3 結果

富士山を描くときに使われる曲線の数は 4 本、三日月を描くときに使われる曲線数は 2 本だった。それぞれの作図の結果は図 14 と図 15 のようになった。

### 14.4 考察

ここまで様々なプログラムを用いてレーザー光の動きを観察してきた。世の中の機械による発光はこれらの組み込みシステムによってプログラムが処理されて光が動いたり点滅したりしているということが実感できた。



図 15: 三日月型のレーザー光描写