3.2 可視化実験（２）: Arduinoとの連携とグラフの描画

実験目標 (今回と次回):

- Processingを使った基本的な描画法を理解し，各自で基本的な描画プログラムを作成できるようになる．
- ArduinoとProcessingをシリアル通信で連携して情報をやり取りするための仕組みを理解し，各自が通信のためのプログラムを書けるようになる．
- Arduinoで読み取ったセンサーの値のデータなどをProcessingに送信して，送信したデータをリアルタイムで可視化する方法を学ぶ．
- Processingへの入力情報をArduinoに送信してArduinoを操作する方法を学ぶ．

3.2.1 Arduinoから Processingへのシリアル通信：複数の値を送信

可視化実験（１）の演習や課題では，Arduinoから Processingに１回の送信でデータを１個（1バイト）ずつ送信していたが，実際には１回の送信で複数個のデータを送信したい場合も多いであろう．例えば複数のセンサーから得られた情報をArduinoから Processingに送信したい場合には，複数個のデータを一度に送る必要がある．本節では一度に複数個のデータを送受信する方法を解説する．

簡単な例として，3つの１バイトの値（0〜255の整数）x, y, zをArduinoから Processingに繰返し送信する場合を考えよう．Arduino側では単純にloop()関数内で

```
Serial.write(x);
Serial.write(y);
Serial.write(z);
```

のように3つ組のデータをシリアルポートに送信したとする．Processingには3つ組データが順次リアルポートに送られてくるが，どの値がx（あるいはy,z）に対応するのか対応をとりながら読み込む必要がある．例えば，Processing側では（データが3つ組で送信されてくることを知っていたとしても）値の読み取りのタイミングがずれるとyzx, yzx, ... のように3つ組データを切り分けてしまうかも知れない．このような状況を回避するために，Arduino側と Processing側で次のようにデータの送受信を行う（関連部分のみ記述）．ただし，下記は通信方式１（可視化実験（１）参照）を用いた場合の例を示している．

```
// Arduino側：
Serial.write('H');
Serial.write(x); // x, y, zは(0-255の整数)
Serial.write(y);
Serial.write(z);
```

```
// Processing 側：
void serialEvent(Serial p) {
  if( p.available() >= 4 ){ // 受信バッファに到着しているデータ数が 4 以上なら
    if ( p.read() == 'H' ) {
      x = p.read();
      y = p.read();
      z = p.read();
      p.clear();  // 念のため受信バッファをクリア
    }
  }
}
```

（解説）Arduino（送信側）からは 3 つ組のデータの先頭にデータの先頭を表す識別子を付けて送信する．この例では識別子は文字'H' と定義した．Processing でデータを受信して serialEvent() 関数が呼び出された時点では 3 つ組データが途中までしか到着していないかもしれない．このような場合にデータの読み込みを避けるため if 文の条件が (p.available() >= 4) となっている．さらに，受信バッファにデータが 4 つ以上到着している場合でも，yz'H'x のようにデータが到着している可能性もある．このような場合は if 文の条件 (p.read() == 'H') により，1 バイトだけデータを読み込んで，そのデータは破棄される．結局，そのうち'H'xyz の順でデータを読み込むことができ，その場合のみ x,y,z の値が更新される．また，保険として 3 つ組データの読み込み後に p.clear() を実行して，受信バッファをクリアしている（なくても通常問題ない）．

＜演習 3.2.1 ＞ 複数データの送信（通信方式 1）

次の「タスク 2」を通信方式 1 で実現する Arduino スケッチと Processing スケッチを作成して実行せよ．

タスク 2 ：2 つの電圧値情報の可視化

- 図 3.17 に示す回路を Arduino とブレッドボードを用いて構成し，2 つの半固定抵抗を回して変化する 2 点 S0 と S1 の電圧値を Arduino の A0 ポートと A1 ポートで観測（要するに，先週のタスク 1 で抵抗を 2 つにした場合に相当）．
- Arduino で観測した点 S0 と S1 の電圧値（を 0-255 に AD 変換した値を）を Processing に送信し，S0 の電圧値（に比例する値）を横幅，S1 の電圧値（に比例する値）を縦幅，とする楕円として電圧値をリアルタイムに可視化．
- ウィンドウサイズは (256,256) とし，ウィンドウの中心を楕円の中心とする．楕円の横と縦の幅はそれぞれ電圧値が 0V 時 0，電圧値が 5V の時 256 となるように表示．

基本手順：

1. 図 3.17 に示す回路を Arduino とブレッドボードを用いて構成する．
2. 先週の演習 3.1.7 で作成した Arduino スケッチ (serial_method1_1byte) と Processing スケッチ (serial_method1_1byte_circle) を修正してタスク 2 を実現するスケッチを作成．作成したスケッチ名は Arduino スケッチ (serial_method1_1byte_multi) と Processing スケッチ (serial_method1_1byte_ellipse) とせよ．

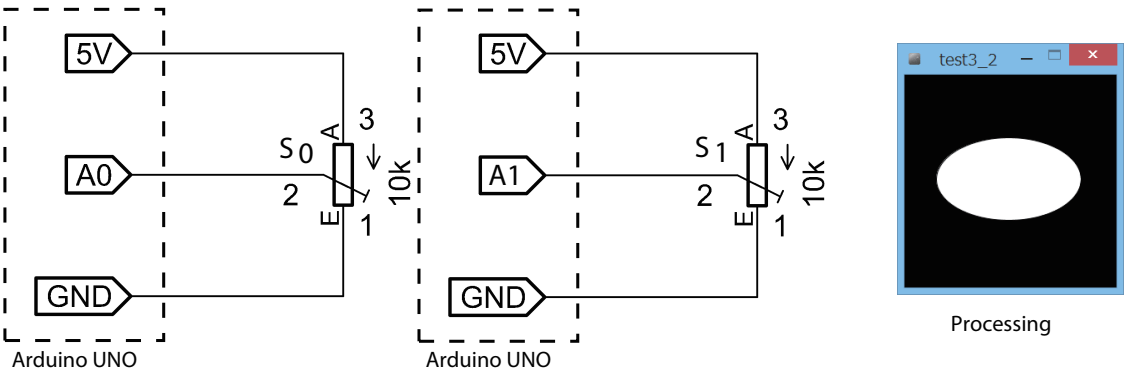

図 3.17: 2 つの半固定抵抗により変化する 2 点 S0 と S1 の電圧値を楕円の横幅と縦幅でリアルタイムに可視化.

3. Arduino を PC に接続して Arduino スケッチを書き込む．次に Processing スケッチを実行して電圧値のデータを可視化する．Processing を実行した状態では Arduino にスケッチを書き込めないので，Arduino スケッチを書き込む時には Processing は実行していないこと（IDE は開いていても良い）．
4. 2 つの半固定抵抗を回してリアルタイムに正しく可視化が実現できることを確認.

注意：Processing でシリアル通信を使用したスケッチを実行中にシリアルケーブルを絶対に抜かないこと．同様に Arduino でシリアル通信を使用している時にも（シリアルモニタを見ているときなど）シリアルケーブルを抜かないこと．シリアルポートが使用不能になることがあります.

3.2.2 Arduino から Processing へのシリアル通信：大きな値を送信

これまでの演習や課題では，Arduino で読み込んだ電圧値（0-5V）を 0-1023 の整数に AD 変換し，さらにこの値を 4 で割って 0-255 の整数に変換してから Processing に送信した．ここで，値を 4 で割った理由は Serial.write() 関数が 1 バイト（8 ビット）ずつシリアルポートにデータを送信するためである（1 バイトで表現できる整数は 0〜255 である）．しかし，もともと 1024 段階の解像度をもつデータを 256 段階の解像度に情報を落としてしまうのはもったいない．本節では int 型の整数値をシリアル通信で送信する方法を解説する．また，float 型で表された小数値を送信する方法についても解説する.

Arduino の int 型データはマイコン内部で 2 バイト（16 ビット）の 2 進数で表現され，10 進数で-32768〜32767 の範囲の整数値を表現できる[1]．図 3.18 に int 型の整数の 16 ビット表現と，対応する 10 進数の値の例をいくつか示す．16 ビット中最初のビットは符号を表すビット（0 で正数，1 で負数）で，残りの 15 ビットで 32768 $(= 2^{15})$ 種類の整数値を表現する.

Arduino の int 型の変数 a に対して，Serial.write(a) のようにデータ送信を行うと，2 バイトのデータの下位 1 バイトのみが送信される．また，Processing で int a = p.read() とデータを受信すると，受信した 1 バイトのデータが int 型変数 a の下位 1 バイトに代入され，それ以外の上位バイトには 0 が代入される．よって，int 型変数 a の値が 0〜255 である場合には，この方法で Arduino から Processing に値を送信することができる.

[1] 多くの計算機では int 型のデータは 4 バイトで表現可能な-2147483648〜2147483647 の範囲の整数を表す.

16 ビット表現	対応する 10 進数
00000000 00000000	0
00000000 00000001	1
00000000 00000010	2
⋮	⋮
00000000 11111111	255 $(= 2^8 - 1)$
⋮	⋮
01111111 11111111	32767 $(= 2^{15} - 1)$
10000000 00000000	-32768 $(= -2^{15})$
10000000 00000001	-32767 $(= -2^{15} + 1)$
10000000 00000010	-32766 $(= -2^{15} + 2)$
⋮	⋮
10000000 11111111	-32513 $(= -2^{15} + 255)$
⋮	⋮
11111111 11111111	-1 $(= -2^{15} + 32767)$

図 3.18: 16 ビットで表される int 型データの 2 進数表現と 10 進数の値の対応関係

Arduino から Processing へ int 型変数 a の値（-32768〜32767）をシリアル通信で送信するには，送信側（Arduino）で送信データを上位 1 バイトと下位 1 バイトに分割し，それぞれを 1 バイトのデータとして送信する．受信側（Processing）では受取った 2 つの 1 バイトのデータを元の整数値に復元すれば良い．図 3.19 にこのようなデータの送受信方式の概念図を示す．

図 3.19: 2 バイトで表される整数 (950) を 1 バイトごとに分割して送受信する方法 (http://yoppa.org/ より転載)

以下に，図 3.19 の概念図に示した int 型データの送受信を行うスケッチ例（通信方式 1，関連部分のみ）を示す．

```
// Arduino 側：
int val, high, low;
...
val = 950;
high = val >> 8; // high に val の上位 8 ビットを代入
low = val & 255; // low に val の下位 8 ビットを代入
Serial.write('H');
Serial.write(high);
Serial.write(low);

// Processing 側：
int val, high, low;
...
void serialEvent(Serial p) {
  if( p.available() >= 3 ) { // 受信バッファに到着しているデータ数が 3 以上なら
    if ( p.read() == 'H' ) {
      high = p.read();
      low = p.read();
      val = (high << 8 ) + low;
      if( 32767 < val ) // この場合は負数に直す（値が正であることが既知なら省略可）
        val -= 65536; // 65536 = 10^16
} } }
```

（解説）>>は右ビットシフトを行う演算子で，左辺の数値の各桁を右に指定ビット数移動させる．同様に，<<は左ビットシフトを行う演算子で，左辺の各桁を左に指定ビット数移動させる．&は左辺と右辺のビット単位の AND 計算をする演算子である．ただし，これらの演算は 2 進数表現に対して行われる．例えば，950 という整数に対して，

```
950 >> 8 // 上位 8 ビットの取得
950 & 255 // 下位 8 ビットの取得
```

という演算は，950 の 2 進数表現 00000011 10110110 に対して，それぞれ次のような操作を行う．

```
0000001110110110 >> 8 --演算結果--> 0000000000000011 (10 進数で 3)
0000001110110110 & 0000000011111111 --演算結果--> 0000000010110110 (10 進数で 182)
```

Processing 側でデータを復元する場合には，受信した上位 8 ビットと下位 8 ビットのデータを次のように復元する（2 進数で表現した場合）．

```
(0000000000000011 << 8) + 0000000010110110
--演算結果--> 0000001100000000 + 0000000010110110 = 0000001110110110
```

上記の演算を 10 進数で表せば,

```
(3 << 8) + 182 --演算結果--> 3*256 + 182 = 950
```

となる.

次に float 型の小数を Arduino から Processing にシリアル通信で送信する場合を考える．具体例として，小数点第 2 位までの値を送受信するためのスケッチ例（通信方式 1，関連部分のみ記述）を以下に示す.

```
// Arduino 側：
float f;
int val, high, low;
....
f = 12.3456;
val = (int)(100.0 * f); //  100 倍して int 型にキャスト（切り捨て）
high = val >> 8; // high に val の上位 8 ビットを代入
low = val & 255; // low に val の下位 8 ビットを代入
Serial.write('H');
Serial.write(high);
Serial.write(low);


// Processing 側：
float f;
int val, high, low;
....
void serialEvent(Serial p) {
  if( p.available() >= 3 ){ // 受信バッファに到着しているデータ数が 3 以上なら
    if ( p.read() == 'H' ) {
      high = p.read();
      low = p.read();
      val = (high << 8 ) + low;
      if( 32767 < val )
        val -= 65536; // 65536 = 10^16
      f = (float)val / 100.0;  // 100 倍されて送られてきた値を元に戻す
    }
  }
}
```

上記の方法では送受信したい小数値を 100 倍してから int 型にキャストして，2 バイトの int 型データとしてデータを送信している．受信側では受信した int 型データを 100 で割ってもとの小数に戻している．すなわち，この方法では-327.68〜327.67 の範囲の小数しか送信できないことに注意が必要である.

＜演習 3.2.2 ＞ 大きな値の送信（複数）

次の「タスク 2 M」を通信方式 1 で実現する Arduino スケッチと Processing スケッチを作成して実行せよ（タスク 2 との差異は赤字で示されている）.

タスク 2 M：2 つの電圧値情報の可視化（複数），大きな値 (0-1023)

- 図 3.17 に示す回路を Arduino とブレッドボードを用いて構成し，2 つの半固定抵抗を回して変化する 2 点 S0 と S1 の電圧値を Arduino の A0 ポートと A1 ポートで観測.
- Arduino で観測した点 S0 と S1 の電圧値（を 0-1023 に AD 変換した値を）を Processing に送信し，S0 の電圧値（に比例する値）を横幅，S1 の電圧値（に比例する値）を縦幅，とする楕円として電圧値をリアルタイムに可視化.
- ウィンドウサイズは 1024 × 1024 とし，ウィンドウの中心を楕円の中心とする．楕円の横と縦の幅はそれぞれ電圧値が 0V 時 0，電圧値が 5V の時 1024 となるように表示.

基本手順：

1. 図 3.17 に示す回路を Arduino とブレッドボードを用いて構成する（構成済み）.
2. 演習 3.2.1 で作成した Arduino スケッチ (serial_method1_1byte_multi) と Processing スケッチ (serial_method1_1byte_ellipse) を修正してタスク 2 Mを実現するスケッチを作成．作成したスケッチ名は Arduino スケッチ (serial_method1) と Processing スケッチ (serial_method1_ellipse) とせよ.
3. 半固定抵抗を回して正しく可視化できているか確認せよ．また，Processing で受信データを println() 関数で表示して結果を確認せよ．※ println(val0, val1); のように記述すれば良い.

3.2.3 温度センサによる温度計測の可視化（概要）

基礎実験 2.5 では図 3.20 のような回路を組んで温度センサによる温度の計測を行った．温度センサの出力電圧を V_0[mV]，温度を T[℃] とすると，

$$V_0 = 10 \times T + 600$$

の関係がある．変形すれば温度は

$$T = \frac{(V_0 - 600)}{10}$$

となる.

温度センサを手でつまんでセンサの温度が 25 ℃から 30 ℃まで変化するとする．この時，電圧 V_0 は 850mV から 900mV に変化するので，電圧値を Arduino のアナログポートから読み取って 0-1023 の値に AD 変換した結果は，25 ℃と 30 ℃の時にそれぞれ 174($=\lfloor \frac{850}{5000} \times 1024 \rfloor$) と 184($=\lfloor \frac{900}{5000} \times 1024 \rfloor$) になる．ここで $\lfloor\ \rfloor$ は小数の切り捨てを行う床関数である．よって，この温度範囲では 10 段階程度の解像度でしか温度変化を観測できないことになる．同じ温度センサを用いてより高解像度で温度変化を観測するためには，観測電圧 V_0 を電子回路で増幅してから Arduino のアナログポートで読み取る方法が自然である．電圧値の増幅はオペアンプと呼ばれる電子回路を用いて簡単に実行できる．一方，本節の実験では回路による増幅を使わずに，観測電圧 V_0 を複数回観測して平均値を用いることで，より高解像度で温度を観測することを試みる.

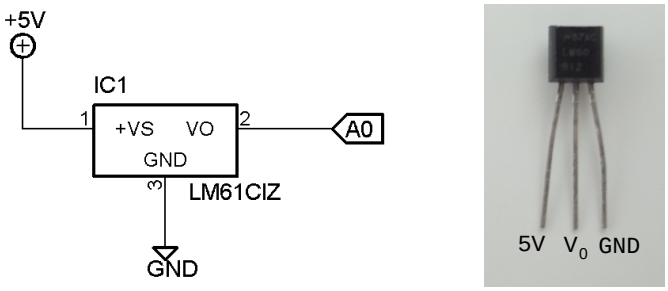

図 3.20: 温度センサ（右図）の温度で変化する電圧変化を Arduino で観測する回路図

図 3.21 に温度センサを指でつまんだ時の電圧値の時間軸グラフを示す．ただし，ウィンドウサイズは 1200 × 500，横軸はデータの表示回数（フレームレート 60 で表示），縦軸は電圧値 V_0 を 0-1023 に AD 変換した値を 170 から 190 の範囲で表示している．グラフより電圧値（を AD 変換した値）は 11 段階ぐらいで変化していることが分かる．温度センサの温度は単調に増加しているはずなので，まったく電気的なノイズや温度センサの動作に誤差のない理想的な状況で時間-電圧グラフを表示すると，階段状のグラフが得られるはずであるが，図 3.21 を見ると AD 変換後の値は連続する 2 つの整数値の間でぶれていることが分かる．おそらく電圧値 V_0 の AD 変換後の値が a（整数）とも $a+1$ とも言えない中間的な状況では等確率でどちらかの値をとり，そこから V_0 の値が増加（減少）すれば AD 変換後の値が $a+1$ (a) となる確率が増加すると考えられる．このことを利用すれば，電圧 V_0 の AD 変換後の値をある程度の数サンプルして平均化することで，より高解像度で温度を計測することができると期待される．

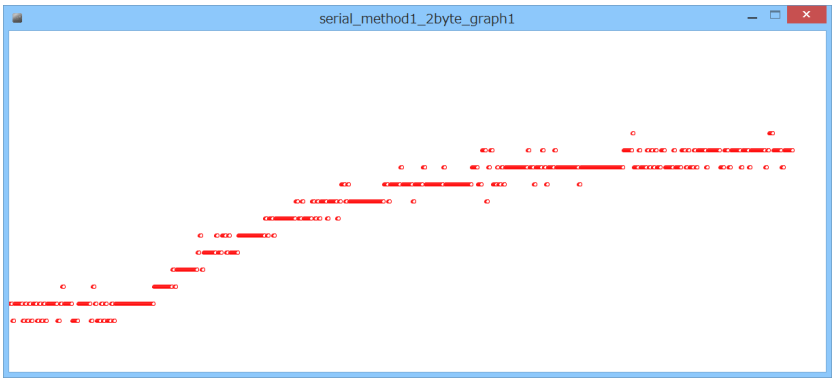

図 3.21: 温度センサの時間（データ表示回数）-電圧値（AD 変換後の値）グラフ

Processing ではある範囲にある値を別の範囲に線形変換したい場合が良くある．例えば温度 T[℃] の値を y 軸に表示させることを考える．温度範囲 $[T_{low}, T_{high}]$ が y 軸のピクセル座標 $[y_{low}, y_{high}]$ にマッピングされるように線形変換したい場合，マッピング後の y 軸のピクセル座標 y は次式で計算される．

$$y = (T - T_{low}) \times \frac{y_{high} - y_{low}}{T_{high} - T_{low}} + y_{low}$$

y の値は map() 関数を使うと次のように直観的に得られるので便利である．

```
y = map(T, T_low, T_high, y_low, y_high);
```

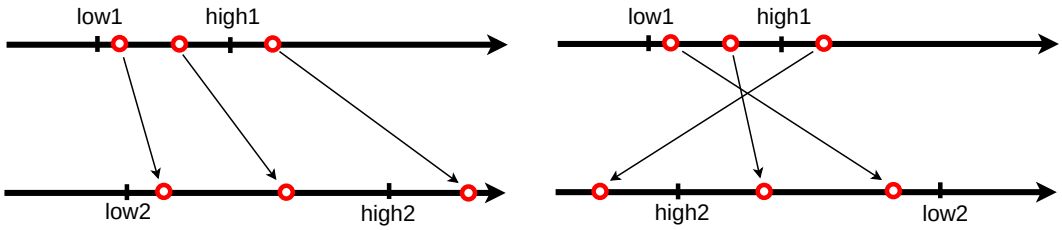

図 3.22: map() 関数による値の変換．右の例では $high2 < low2$ であることに注意．

map() 関数の仕様は以下の通りである．map() 関数によって値が変換される概念図 3.22 に示す．

map() 関数：
書式：map(value, low1, high1, low2, high2)
引数：整数または浮動小数
機能：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（範囲外の値でも OK）
戻り値：(float) 線形マッピング後の値

map() 関数の使用例：

```
int a = 120;
float b;
b = map(a, 100, 200, 400, 800)
printlin(b); // 480.00 が表示
```

3.2.4 温度センサによる温度計測の可視化（実習）

＜演習 3.2.3 ＞ 温度センサによる温度計測：基本フレームワーク

まずは温度センサで観測した温度変化をデータの平均化を考えずに時間-電圧（AD 変換後の値）グラフとして可視化してみよう．可視化実験（１）でも，光センサーの出力に対して時間 - 電圧（AD 変換後の値）グラフを作成したが，その時はデータの受信回数を疑似的に時間とみなしていた．今回は Arduino でデータを観測した時の時刻も送信することにする．今は温度センサーで読み取った電圧（AD 変換後の値）は温度に変換せずにそのまま表示することにする．

基本手順：

1. 図 3.20 に示す回路を Arduino とブレッドボードを用いて構成する．
2. 下記に記述されている Arduino スケッチ (serial_method1_thermo_basic) を用いて，温度センサーでセンシングした時刻（mills() 関数の値）および観測電圧 V_0 を AD 変換した値 (0-1023) を 50ms 間隔で Processing に送信する．スケッチの内容を良く見て理解すること．
3. 下記に記述されている Processing スケッチ (serial_method1_thermo_basic) を用いて，グラフを描画する．スケッチの内容を良く見て理解すること．

4. 温度センサを手で握り，出力されるグラフを確認（ほとんど y 方向の変化なし）.
5. println() で受信データ time（時刻）と val（AD 変換後の値）を表示せよ．val のおおよその最小値と最大値を確認し，最小値と最大値の範囲がグラフの表示範囲（y 軸方向）の 70%程度を占めるように y 軸の表示範囲値を調整せよ.
6. Processing スケッチのパラメーター period の値を適当に変更して，想定どうりに描画されるか確認せよ.

Arduino スケッチ: serial_method1_thermo_basic

```
int sensorValue0;
unsigned long int timePrev, timeNow;

void setup()
{
  Serial.begin(9600); // シリアル通信を 9600bps で初期化
  timePrev = millis(); // 経過時間の初期値
}

void loop()
{
  timeNow = millis(); // 現在の経過時間 （ms 単位）
  sensorValue0 = analogRead(0); // A0 ピンの AD 変換結果を取得 (0-1023)

  if ( timeNow - timePrev >= 50 ) { // 50ms ごとに実行
    Serial.write('H');
    Serial.write(timeNow >> 24);  // 1byte 目
    Serial.write(timeNow >>16);   // 2byte 目
    Serial.write(timeNow >> 8);   // 3byte 目
    Serial.write(timeNow & 255);  // 4byte 目
    Serial.write(sensorValue0 >> 8);    // 上位 1byte
    Serial.write(sensorValue0 & 255);   // 下位 1byte
    timePrev = timeNow; // 1 ステップ前の経過時間を更新
  }
}
```

スケッチの概要：

- loop() 関数では常時 A0 ポートから電圧値 (0-5 V) を AD 変換した値 (0-1023) を読み込む.
- loop() 関数では常時現在の経過時間（timeNow）を計測し，前回通信を行った経過時間（timePrev）との差が 50 (ms) を越えたら再度データ送信.
- timeNow は unsigned long int 型（4 バイト）のデータなので，4 バイトのデータを 4 つに分割して送信していることに注意.

Processing スケッチ: serial_method1_thermo_basic

```
import processing.serial.*;
Serial port;
int val;
int x, y;
int high, low;
int byte1, byte2, byte3, byte4;
int time, time_min, time_max;
int period;

void setup()
{
  size(1200, 500);
  port = new Serial(this, "/dev/ttyACM0", 9600);

  period = 20000;   // 横軸の範囲は 20,000ms 間隔
  time_min = 0;     // 横軸の範囲の初期値
  time_max = period;// 横軸の範囲の初期値

  x = 0; y = 0;
  time = 0;
  background(255);
  frameRate(60);
}

void draw()
{
  if ( time > time_max ) { // グラフの再描画
    time_min += period; // 横軸の範囲の更新
    time_max += period; // 横軸の範囲の更新
    background(255);
  }

  x = (int)map( time, time_min, time_max, 0, width ); // x 座標値
  y = (int)map( val, 0, 1024, height, 0 );            // y 座標値
  stroke(255, 0, 0);
  ellipse(x, y, 5, 5);
}

void serialEvent(Serial p) {
  if ( p.available() >= 7 ) {
    if ( p.read() == 'H' ) {
      byte1 = p.read();
      byte2 = p.read();
      byte3 = p.read();
      byte4 = p.read();
      time = (byte1 << 24 ) + (byte2 << 16 ) + (byte3 << 8 ) + byte4; // 4 バイトデータ

      high = p.read();
      low = p.read();
      val = (high << 8 ) + low; // 2 バイトデータ
      p.clear();
    }
  }
}
```

スケッチの概要：

- 横軸は時刻 (time) を表示．時刻の表示範囲は [time_min, time_max] である．ただし，time が表示範囲を超えた場合は表示範囲を更新し，再度描画を行う．
- period は横軸の左端から右端の間の時間間隔（ミリ秒単位）
- time は Arduino から送信されてくる unsigned long int 型データ（4 バイト）を受け取る．Processing には unsigned 型（符号なしデータ型）が無いので，int 型で受け取る．Processing の int 型は 4 バイト長なので long int 型にする必要はない．

＜課題 3.2.1＞ 温度センサによる温度計測：複数サンプルの平均化

演習 3.2.3 では，温度センサから出力される電圧値（AD 変換した値）を 50ms 間隔でそのまま送信した．本課題では各 50ms 間隔の間に出来るだけ多くの電圧値（AD 変換した値）を観測し，その平均値を送信する．

基本手順：

1. 演習 3.2.3 で作成した Arduino スケッチ (serial_method1_thermo_basic) と Processing スケッチ (serial_method1_thermo_basic) を編集して作成すると良い．作成した Arduino スケッチを (serial_method1_thermo_average)，Processing スケッチを (serial_method1_thermo_average) という名前で保存せよ．以下にヒントを記載しておく．
   - 50ms ごとに電圧値（AD 変換した値）の平均値を計算するためには，loop() 関数内で A0 ポートから電圧 V_0 を読み込みたびに，AD 変換された値を足しこんでいく必要がある．足しこんだ値を保持するための変数は long int 型（4 バイトの int 型 : -2147483648〜2147483647 の範囲の整数）で宣言する．その理由は，足しこんだ値が int 型の範囲（-32768〜32767）を超えてしまう可能性があるためである．
   - 当然，各 50ms 間隔の間に何回データを足しこんだかカウントするための変数が必要である．
   - int 型変数 a を int 型変数 b で割った値を float 型変数 f に代入するには `f = (float)a/(float)b` とすれば良い．`f = a/b` とすると，小数点以下が切り捨てられる．これは平均値の計算で必要．
   - float 型変数の値をシリアル通信で送受信する方法は 3.2.2 で説明している．この課題では少数点第 2 位まで送信すれば十分だろう．
2. 電圧（AD 変換された値）データの最小値と最大値がウィンドウの縦幅の 7 割程度を占めるように調整せよ．

＜課題 3.2.2＞ 温度センサによる温度計測：時間-温度グラフの完成

課題 3.2.1 では図 3.20 に示される回路の温度センサからから出力される電圧（AD 変換した値）の平均値を時間軸グラフとして可視化した．本課題では y 軸の表示を温度単位とし，さらに温度のメモリを付けて表示してみよう．Processing の具体的な仕様は以下の通りとする．作成した Processing スケッチを (serial_method1_thermo) という名前で保存せよ．Arduino スケッチ (serial_method1_thermo_average) の変更はない．

- y 軸の表示を℃にする．
- 表示温度の最小値と最大値を t_min, t_max としてスケッチ内で指定する．
- 1 ℃ごとに横線を引き，各横線に対応する温度を文字で表示する（以下で説明する text() 関数を使用）．※ 温度目盛り表示用の関数を作成するのが良い．
- その他の仕様は課題 3.2.1 と同様．

ウィンドウ上に文字を出力するには text() 関数と textSize() 関数を用いる．これらの関数の仕様は以下の通り．

text() 関数：
書式：text(data, x, y)
引数：data は (String, char, int, float), x と y は (int または float)
機能：data が表す文字または文字列をピクセル座標 (x,y) の位置に描画．
戻り値：なし

textSize() 関数：
書式：textSize(a)
引数：a は int または float
機能：テキストサイズを a に設定
戻り値：なし

文字描画のスケッチ例：※このスケッチをコピーして基本モードで描画してみよ．

```
size(200,200);
textSize(30);
fill(0);
text("Word", 100, 100 );
int n = 40;
text(n, 100, 150);
noFill();
```

※ 文字の色は fill() 関数で設定する（図形の塗りとして色を指定）．文字の描画後は noFill() を実行しないと，そのあとの描画に影響してしまう．

＜課題 3.2.3 ＞ 温度センサによる温度計測：グラフ配置

これまでの課題ではグラフをウィンドウ全体に表示していたが，実際にはウィンドウの一部にグラフを表示した場合も多いであろう．課題 3.2.2 で作成した Processing スケッチを修正して，次の仕様で可視化せよ．作成した Processing スケッチを (serial_method1_thermo_partial) という名前で保存せよ．Arduino スケッチ（serial_method1_average）の変更はない．

- ウィンドウサイズ：1200 × 500
- ウィンドウ内での表示範囲（ピクセル座標系）は図 3.23 に示すように，4 つのパラメーターを導入して x 軸については [x_min, x_max]，y 軸については [y_min, y_max] と表す．具体的な数値は (x_min, x_max)=(100, 600)，(y_min, y_max)=(100, 250) とする．

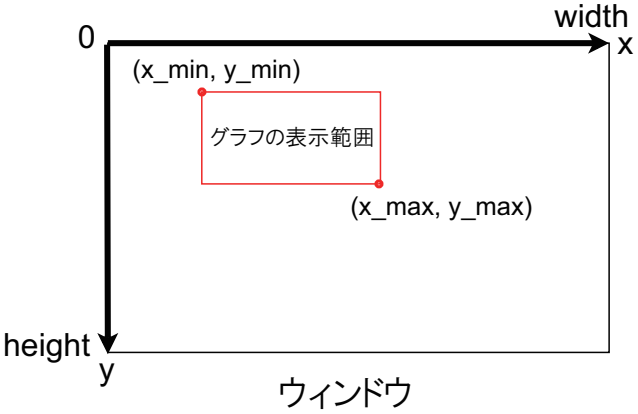

図 3.23: グラフをウィンドウの一部分に表示

3.2.5 発展課題

＜発展課題 3.2.1＞ 複数グラフの表示

課題 3.2.3 ではウィンドウに一部分に 1 つのグラフを表示した．この課題では温度センサと光センサでデータを観測し，Processing では 1 つのウィンドウ内で 2 つのグラフを表示させる．以下の仕様を満たす Arduino スケッチと Processing スケッチを作成してグラフを描画せよ．

- 課題 3.2.3 と同様に図 3.20 に示した回路上の温度センサから得られる情報を時間-温度グラフとして可視化する．
- さらに，図 3.24 に示した回路上の光センサから得られる情報を時間-電圧（AD 変換した値）グラフとして可視化する．図では A0 ポートの電圧を観測しているが，A1 ポートから観測するものとする．
- 2 つのグラフを 1 つのウィンドウ上の適当な位置に配置して表示．

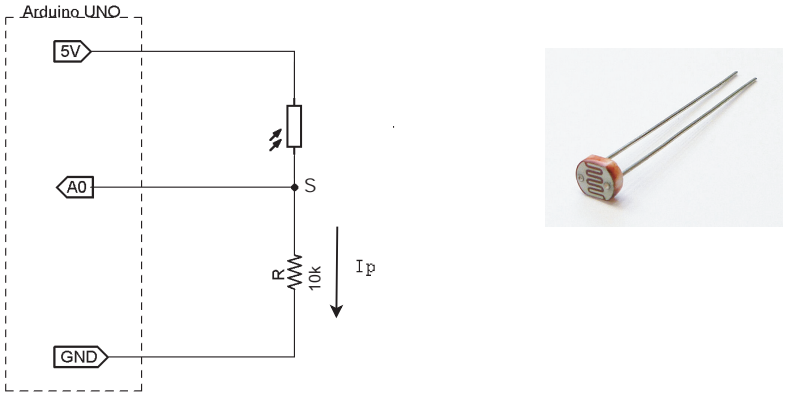

図 3.24: CdS 光センサ（右図）にあたる光の量で変化する電圧変化を Arduino で観測する回路図

＜発展課題 3.2.2＞ リサージュ図形の描画

直角な 2 方向の単振動を組み合わせたときに描かれる図形をリサジュー図形という．すなわち，次式

$$\begin{aligned} x &= \cos(\omega_x t) \\ y &= \cos(\omega_y t + \delta) \end{aligned}$$

で t を動かした時に得られる点 (x, y) の軌跡をリサージュ図形と呼ぶ．リサージュ図形は ω_x と ω_y の比と，位相差 δ の値によってさまざまな形状をとる．図 3.25 に典型的な形状を示す．以下の仕様でリサージュ図形を描く Processing スケッチを作成せよ．出力される図形の参考図を図 3.26 に示す．Arduino スケッチは serial_method1 を用いれば良い．

- 演習 3.2.2 では図 3.17 に示す回路の 2 つの半固定抵抗を回して変化する 2 点 S0 と S1 の電圧値（を AD 変換した値）を Arduino で観測し，Processing で楕円の横幅，縦幅として可視化した．2 つの電圧値（を AD 変換した値）を a，b とする．
- $\omega_x = 1$ として，a が 0 から 1023 まで変化した時 ω_y が 1 から 4 まで 1024 段階に変化．
- b が 0 から 1023 まで変化した時 δ が 0 から 2π まで 1024 段階に変化．
- ウィンドウサイズは適当で良いが，ウィンドウの壁に接しない用にリサージュ図形を表示．
- t を少しずつずらしながら線で結んで描画．t の刻み幅や，どこまで t を進めるかは各自で考えて設定すること（1 周期が非常に長くなる場合もある）．
- さらに余裕があれば以下も考慮せよ．リサージュ図形は ω_x と ω_y がある程度簡単な整数比にならないと，非常に線が入り組んだ図形になってしまう（1 周期が非常に長くなる）．a を 1 から 4 までの数値に変換した値に対し，$\frac{n}{m}$ $(1 \leq m \leq 30, m \leq n)$ の中から最も値が近い $\frac{n}{m}$ の値で a で置き換えることで，ω_x と ω_y の比をある程度単純な整数比になるように補正することができる．

（補足）Processing では

```
float omega;
float x = cos(omega);
float y = sin(omega);
```

のように三角関数が使用できる．引数 `omega` の単位はラジアン（$2\pi = 360°$）である．また，円周率は定数 `PI` で定義されている．

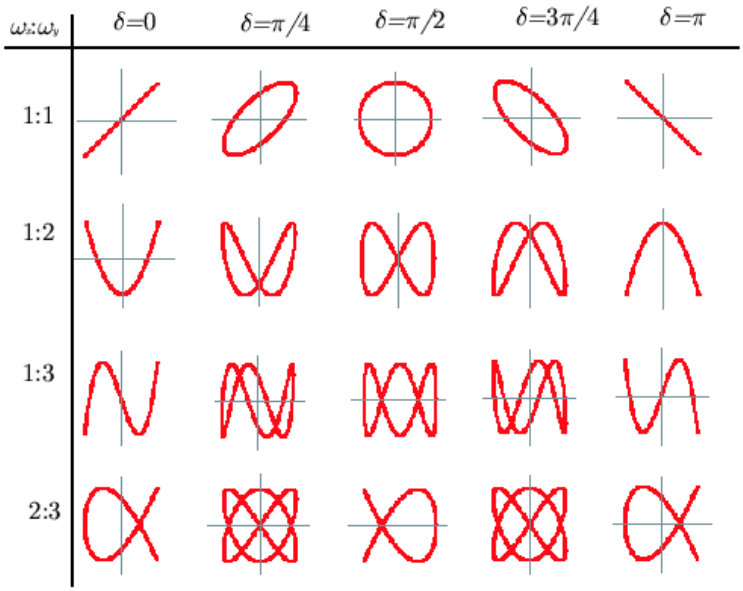

図 3.25: 典型的なリサージュ図形の例（http://www.ne.jp/asahi/tokyo/nkgw/index.html より転載）

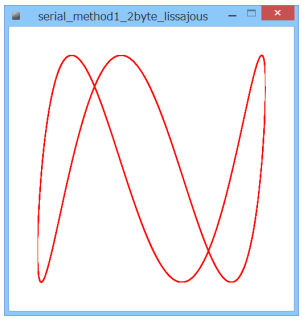

図 3.26: Processing で描画したリサージュ図形の例

【レポート 3.2　（2019 年 5 月 31 日出題，2019 年 6 月 7 日 (12:50) 締切）】

※ スケッチには詳細なコメントを記述すること．コメントの無いものは採点しない．

レポート 3.2.1 温度センサによる温度計測：基本フレームワーク

演習 3.2.3 で表示した時間-電圧（AD 変換後の値）グラフのスナップショットを報告せよ．

レポート 3.2.2 温度センサによる温度計測：複数サンプルの平均化

課題 3.2.1 で作成した Arduino スケッチ (serial_method1_thermo_average) と Processing スケッチ (serial_method1_thermo_average) を報告せよ．表示した時間-電圧（AD 変換後の値）グラフのスナップショットを報告せよ．サンプルの平均化を行った場合と行わなかった場合の違いについて考察せよ．

レポート 3.2.3 温度センサによる温度計測：時間-温度グラフの完成

課題 3.2.2 で作成した Processing スケッチ (serial_method1_thermo) を報告せよ．表示した時間-温度グラフのスナップショットを報告せよ．

レポート 3.2.4 温度センサによる温度計測：グラフ配置

課題 3.2.3 で作成した Processing スケッチ (serial_method1_thermo_partial) を報告せよ．表示した時間-温度グラフのスナップショットを報告せよ．

発展課題レポート 3.2.1 複数グラフの表示

発展課題 3.2.1 で作成したスケッチを報告せよ．表示したグラフのスナップショットを報告せよ．

発展課題レポート 3.2.2 リサージュ図形の描画

発展課題 3.2.2 で作成したスケッチを報告せよ．また，表示したリサージュ図形の中で面白いと思う図形のスナップショットを 3 つ報告せよ．

可視化実験（1）（2）で作成したスケッチの内容を以下にまとめておく（発展課題は除く）.

Arduino スケッチ：

- serial_method1_1byte: 1 つの電圧値をアナログポートから読み込んで 0-255 の値に AD 変換し，1 バイトのバイナリデータとしてシリアルポートに送信（通信方式 1）.
- serial_method2_1byte: 1 つの電圧値をアナログポートから読み込んで 0-255 の値に AD 変換し，1 バイトのバイナリデータとしてシリアルポートに送信（通信方式 2）.
- serial_method1_1byte_multi: 2 つの電圧値をアナログポートから読み込んで 0-255 の値に AD 変換し，それぞれ 1 バイトのバイナリデータとしてシリアルポートに送信（通信方式 1）.
- serial_method1: 2 つの電圧値をアナログポートから読み込んで 0-1023 の値に AD 変換し，それぞれ 2 バイトのバイナリデータとしてシリアルポートに送信（通信方式 1）.
- serial_method1_thermo_basic: 1 つの電圧値をアナログポートから読み込んで 0-1023 の値に AD 変換し，データ取得時刻（4 バイトデータ）と AD 変換した値（2 バイトデータ）をシリアルポートに送信（通信方式 1）.
- serial_method1_thermo_average: 1 つの電圧値をアナログポートから読み込んで 0-1023 の値に AD 変換し，データ取得時刻（4 バイトデータ）と AD 変換した値の一定期間内の平均値（2 バイトデータ）をシリアルポートに送信（通信方式 1）.

Processing スケッチ：

- serial_method1_1byte_circle: 1 つの値 (1 バイト) を受信（通信方式 1）. 円の直径として可視化.
- serial_method2_1byte_circle: 1 つの値 (1 バイト) を受信（通信方式 2）. 円の直径として可視化.
- serial_method1_1byte_graph1: 1 つの値 (1 バイト) を受信（通信方式 1）. 時間（データ取得回数）-電圧（AD 変換後の値）グラフを表示
- serial_method1_1byte_graph2: 1 つの値 (1 バイト) を受信（通信方式 1）. 時間（データ取得回数）-電圧（AD 変換後の値）グラフ（折れ線）を表示
- serial_method1_1byte_meter: 1 つの値 (1 バイト) を受信（通信方式 1）. 電圧（AD 変換後の値）のメーター表示
- serial_method1_1byte_ellipse: 2 つの値 (各 1 バイト) を受信（通信方式 1）. 楕円の横幅，縦幅として可視化.
- serial_method1_ellipse: 2 つの値 (各 2 バイト) を受信（通信方式 1）. 楕円の横幅，縦幅として可視化.
- serial_method1_thermo_basic: 時刻 (4 バイト）と電圧（AD 変換後の値，2 バイト）を受信（通信方式 1），時間-電圧（AD 変換後の値）グラフを表示
- serial_method1_thermo_average: 時刻 (4 バイト）と電圧の平均値（AD 変換後の値，2 バイト）を受信（通信方式 1），時間-電圧（AD 変換後の値）グラフを表示
- serial_method1_thermo: 時刻 (4 バイト）と電圧の平均値（AD 変換後の値，2 バイト）を受信（通信方式 1），時間-温度グラフを表示.
- serial_method1_thermo_partial: 時刻 (4 バイト）と電圧の平均値（AD 変換後の値，2 バイト）を受信（通信方式 1），時間-温度グラフをウィンドウの一部分に表示.