11.2　ロボット実験２：Zumo と Processing の通信

本実験では，Zumo ロボットと Processing の通信を行うプログラミングを行う．この実験を行うことで，Zumo ロボットの開発に必要なプログラミング方法と，ロボット開発の考え方，Zumo ロボット開発中にロボットや環境の監視・デバッグを行うために必要な技術を習得できる．

本実験の達成目標：

- Zumo ロボットの開発環境に慣れる．
- Zumo ロボット内での処理時間を考慮したシステム開発を行う．
- XBee 無線通信を介してシリアルポート通信を行い Zumo ロボットを操作する．（11.1 でも習得済み）
- Processing により Zumo ロボットとの通信プログラミングを行う．
- 複数台の Zumo ロボットと 1 台の PC との通信を行い，制御・状況把握を行う．

11.2.1　ロボット内の処理時間

Zumo ロボットに限らず，一般的にロボット開発に用いられるロボットには，たくさんの制御対象となる機器が搭載されている．例として，外界の情報を得るための各種センサー（Zumo では，加速度センサ，地磁気センサ，カラーセンサ，ユーザボタンなど）や，動力源（Zumo では，LED，ブザー，モータなど）などがある．ロボットを動作させるためには，これらの機器を連携動作させる必要がある．

これまでのシステム実験での実習において，上記の各機器の使用方法や動作の仕方については学習を行ってきており，更に，前回のシステム実験においては，Zumo ロボットを用いて，これらの連携動作についての実習を行った．今回の実習では，これらの連携動作について実習を行う．

ここで，処理時間について考えるために，前回の実験のプログラム 4 （図 11.2.1）の一部を確認してみよう．

```
1  void loop() {
2      digitalWrite(trig, HIGH);   //10μs のパルスを超音波センサの Trig ピンに出力
3      delayMicroseconds(10);
4      digitalWrite(trig, LOW);
5
6      interval = pulseIn(echo, HIGH, 23068);  //Echo 信号が HIGH である時間(μs)を計測
7      distance = 340 * interval / 10000 / 2;  //距離(cm)に変換
8      Serial.println(distance);               //距離をシリアルモニタに出力
```

図 11.2.1　前回 11-1 のプログラム 4 の一部

このプログラムでは，超音波センサから超音波を出力し，跳ね返ってきた超音波を受信して，その間の時間から対象物との距離を計測している．そして，その結果をシリアル通信を通して出力（プリント）している．ここで，各処理にかかる時間をイメージしてみよう．

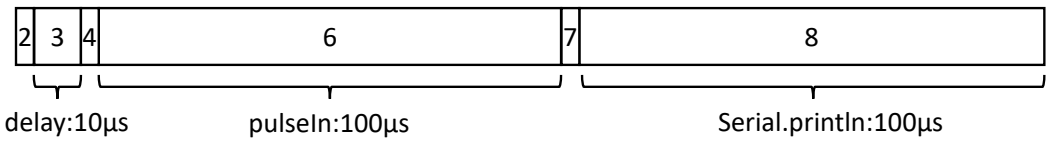

図 11.2.2　処理時間のイメージ図

図 11.2.2 に，このプログラムでの 1 ループ分の処理時間のイメージを図示している．図中の番号は，プログラムの行番号と対応している．ここで注目すべきは，超音波センサを参照してから結果が返ってくるまでの時間と，シリアル通信を用いて結果を出力する時間が長くかかっている点である．センサの参照や，シリアル通信は，他の処理に比べて時間がかかる場合が多い．図の例の場合には，プログラム全体（２行目から８行目まで）で，210 マイクロ秒程度で終わっており，比較的高速に処理が完了していることになる．このプログラムをそのままループ処理させれば，計算上では，秒間 4,761 回もの処理を行うことができるということになる．

しかし，図の例は，各処理が比較的短時間で終わった例であり，各処理が短時間では終わらない場合も考えられる．例えば，プログラム６行目の「pulseIn」は，超音波センサと物体との距離に応じて応答までにかかる時間が変化し，物体が遠くなるほど，応答時間が遅くなる．

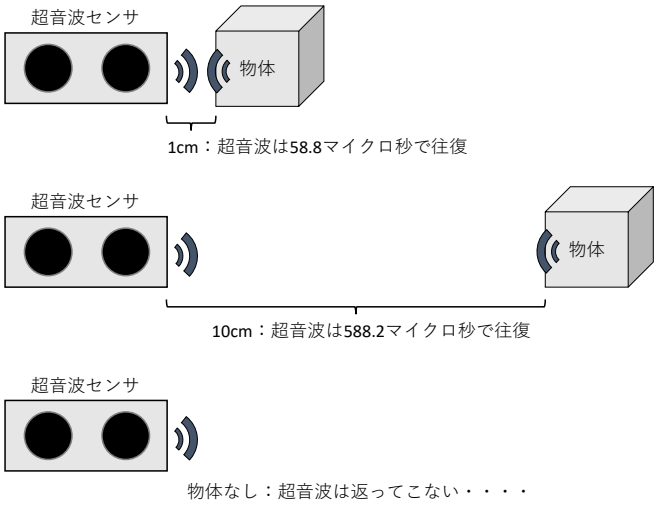

図 11.2.3　超音波センサと物体との位置と pulseIn 応答時間の関係

図 11.2.3 に示すように，例えば，物体との距離が 1cm の場合には，応答までの時間が 58.8 マイクロ秒かかり，10cm の場合には，その 10 倍の 588.2 マイクロ秒がかかることになる．物体までの距離がとても遠い（もしくは物体がない）場合は，その時間が無限に長くなるが，そのようなことを避けるため，pulseIn ではタイムアウト時間が設定できるようになっており，今回のプログラムの例では，23,068 マイクロ秒間応答がなければ，タイムアウトする設定になっている（設定しなければ，1 秒間でタイムアウトする）．つまり，物体が検出できない状況では，pulseIn 関数部分だけで最大 23,068 マイクロ秒かかるということである．この場合，プログラム全体では，23,178 マイクロ秒程度かかることになり，このプログラムをループ処理させる場合には，計算上では，秒間 43 回の処理しか行えないことになる．このプログラムを用いて，秒間 60 フレームで動作・監視するシステムを考えていた場合，処理時間が足りないため，処理落ちが発生することになってしまう．逆に考えれば，60 フレームで処理を行いたい場合には，ループ 1 回の処理を，16,666 マイクロ秒以内に終わらせる必要があり，そのようにプログラムを構築する必要がある．

上記例では，超音波センサを例に示したが，各種センサは応答時間がそれぞれ異なることに注意する必要がある．Zumo では，地磁気センサ，加速度センサは比較的応答時間が短く，カラーセンサや超音波センサは応答時間が長い．

また，上記例ではプログラム４の２，４，７行目にあたる計算処理や CPU からセンサへのアクセス（digitalWrite）の時間については，十分短いものとして扱ったが，これらの処理においても，その種類や規模によって処理時間が大きく異なってくることに注意が必要である．

11.2.2　処理の遅延とその悪影響

前節では，ロボット内の各機器において，処理・制御時間がかかることを説明した．実際にロボットプログラミングを行う上で，このことを意識することは非常に重要なことである．ここでは，処理・制御時間を考慮せずにプログラムを行うと，どのような問題が発生するかについて説明する．

システム実験においてよく起こる事例としては，各機器での処理時間を考慮せず，プログラムの内容を理解しないまま，プログラムの整理整頓をせず，これまでに実習で用いたプログラムを何でもかんでもくっつけて，対象となる目的（タスク）に不必要な処理もプログラムに含めてしまう例である．図 11.2.4 にその例を示す．

```
1  void loop() {
2
3      distance = getSonicSensorDistance()     // 超音波センサの距離計測
4      color = getRGBColor()                   // カラーセンサの値を取得
5      compass.read();                         // 地磁気センサの値を取得
6
7      Serial.println(distance);               // 距離をシリアルモニタに出力
8      Serial.println(color.R);                // カラーセンサ（赤）をシリアルモニタに出力
9      Serial.print(compass.m.x);              // 地磁気センサの値をシリアルモニタに出力
10
11     // 以下，メイン処理
12     . . .
```

図 11.2.4　悪いプログラミング例

例えば，目的として「ロボットを前進させながら，超音波センサを用いて障害物が目の前に無いかどうかを確認し，物体が検出されたらロボットを回転させて避ける」という動作を実現する場合を考える．この目的を達成するために必要な機器としては，超音波センサとモータのみであり，他の機器は不必要である．しかし，図のプログラム例では，1 ループ毎に，地磁気センサの値の取得，カラーセンサの値の取得を行っており，更に，その値をシリアルモニタに出力している．これらの不必要な処理が毎回行われることで，処理が遅延していくことになる．

今回の目的のように，物体検出をしながらロボットを動かす場合，一定時間にロボットが移動する距離，またそれに対応して，一定時間に物体検出を何回行うかを考える必要がある．しかし，上記のように不必要な処理を考慮しないまま，単純にプログラムした場合は処理が追いつかず，ロボットが想定外の動きをしてしまう．

単純なプログラム計画の例（失敗へつながる例）を以下に示す．

1. ロボットは 1 秒間に 20cm の速さで動かしておこう
2. 障害物が 5cm 以内にあったら，ロボットを回転させて回避させよう
3. 毎回のループで，x < 5 となるか確認しよう
4. 1 秒間のループ回数を 10 回にして，毎秒 10 回障害物の確認をしよう
5. 1 ループ毎に時間を測って，100ms 以上経過していたら次のループに入ろう

このプログラム計画では，実際にプログラムの 1 ループ（や各処理）にどの程度の時間がかかるかが考慮されていない．他の処理に時間がかからない場合には，物体検出処理が高速に回ることになり，物体が 5cm 以内に来たことを検出でき，物体回避処理を行うことが可能である．しかし，他の処理に時間がかかる場合には，1 ループの時間が想定より長くなり，物体検出処理が適切に実行されず，ロボットはそのまま障害物に衝突してしまうことになる（図 11.2.5）．今回の例の場合には，100ms 毎に 5cm 以内に障害物が無いか確認しており，その間に想定ではロボットは 2cm しか動かないため，衝突前に障害物を確認できるようになっている．ただし，他の処理に時間がかかり，1 ループの時間が 500ms かかってしまった場合，ロボットはその間に 10cm 進んでしまい，検出範囲外（5cm 以上遠く）にあった障害物にぶつかってしまう可能性がある．

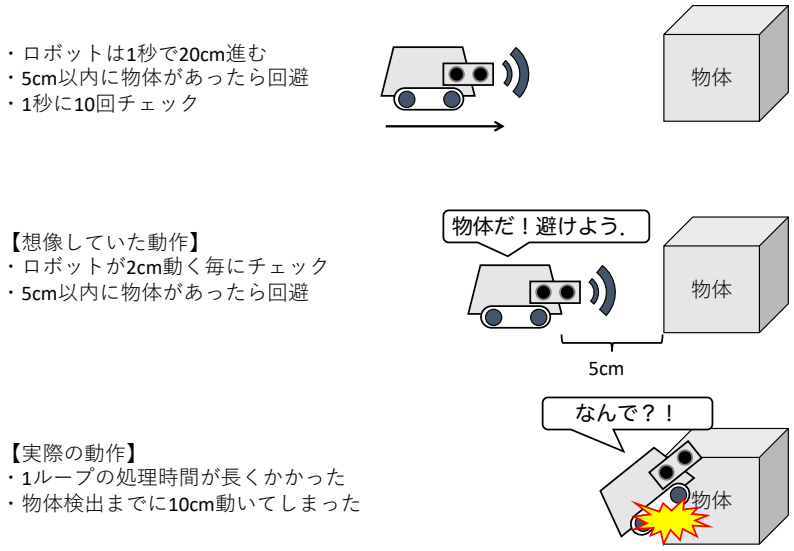

図 11.2.5　処理の遅延による悪影響例

ここまでは，プログラムに不必要な処理が入っており失敗する例を紹介したが，これ以外にも，想定外の動作になってしまう場合はたくさん存在する．例えば，上記と同じ超音波センサを用いる例では，超音波センサのタイムアウト時間を適切に設定していないために，そこで時間を取られて，それ以外の部分との整合が取れずに失敗してしまう例が考えられる．また，コンテスト用のプログラムを構築中に起きやすい問題としては，制限時間に間に合うように，ロボットの動作を早めた場合，単純にロボットの動きを早めてしまうと，その他の機器の処理・制御時間がかかることで，1 ループごとの処理時間が間に合わず，ロボットが想定外の動きをしてしまうことになる（トレースラインを大幅に外れたり，壁にぶつかったり）．

実際にロボットプログラミングを行う場合には，ここで紹介した注意事項を念頭に置いて取り組むことが重要である．

11.2.3　通信の処理時間と通信バッファ

ロボットと外部モジュールの通信や，ロボットとコンピュータを USB などを用いて有線接続する場合には，ロボットと外部との通信に「シリアル通信」が用いられる場合がある．ここでは，シリアル通信にかかる時間と，そこでの通信バッファについて説明する．

通信の処理時間

シリアル通信は，有線で信号を送受信する方式をとっており，信号そのものの遅延は少ない．しかし，通信速度が遅く，今回実験で用いる速度は，9600bps である．これは，データを 1 秒間に 9600bit を送受信できる（ストップビットやパリティビットなどは含まず，純粋にデータ部分を送受信できる速度を示す）．これは，byte に換算すると 1,200byte である．ASCII 文字は 1 文字 1byte であることから，1 秒間に 1,200 文字の送受信が可能であるということになる．逆に考えれば，1 文字を送信するのに，0.83 ミリ秒かかる計算になる（ただし，これは理論値であり，データ転送に失敗した場合などは，より時間がかかることになる）．

文字列の送受信にかかる時間（計算上の時間）の例を表 11.2.1 に示す．

表 11.2.1　シリアル通信による文字列送信にかかる時間（理論値）

文字列	時間（ミリ秒）
a	0.83
abc	2.49
10,20,30	7.47
255,255,255	9.13
r:100,g:200,b:250,gx:1000,gy:-2000	28.22

表に示すように，1 文字の送信では時間は短いものの，4 行目の 3 桁の数字をカンマで区切って送信する場合（255,255,255．RGB 値を送信する場合を想定している）は，文字列を送信するだけで 9.13 ミリ秒もかかっている．この処理だけ考えれば，繰り返し処理させても，1 秒間に 100 回以上実行できることになり，問題が無いように感じるが，他の処理にも時間がかかることを考慮すると，これでも時間がかかっており，注意が必要である．表の最後の行の例においては，rgb を示すラベルや，その他の値も同時に送信することが想定されており，文字列の送信時間は，28.22 ミリ秒になっている．これは，文字列送信だけ考えても，1 秒間に 35 回程度しか実行することができず，60 フレーム/秒でシステムを動作させることを想定している場合には，この時点で時間超過となってしまう．

Zumo ロボットで開発を行う場合には，Arduino を用いたプログラミングを行うが，Arduino でのシリアル通信では，主に以下の表 11.2.2 に示す関数を用いてデータ送信を行う．これらの詳細や，その他の関数については，基礎実験 5（2-5）のテキストを参照されたい．

表 11.2.2　Arduino でのシリアル通信用関数

関数	処理内容
Serial.write()	バイナリデータを出力
Serial.print()	文字列データを出力
Serial.println()	文字列データを出力（末尾に改行コードを挿入）

なお，本実験で用いられる XBee は無線通信であり，XBee 同士は無線で通信を行っているものの，その外側（ロボットや，PC）からは，あたかも有線で接続されたシリアル通信であるかのように使うことができるものである．しかし，実際には XBee 同士は無線通信を行っているため，そこで発生する遅延やバッファ処理についても，考慮する必要がある．

通信バッファ

通信を行う際，データは直接相手のメモリに格納されるわけではなく，実際には，データはバッファに格納される．バッファは，送信側，受信側の両方が持っており，データは各バッファを順に運ばれていく（図 11.2.6）．

Arduino UNO のリシアル通信用バッファは，デフォルト設定では 64byte に設定されている．また，XBee のバッファサイズは 16byte 程度である．バッファサイズを超えて一度に送信をしようとすると，バッファオーバーフロー（バッファ溢れ）が起こり，データが消失してしまう．受信側でデータが喪失して受信できない場合には，通信の問題だけではなく，バッファオーバーフローが起こっていないか，送信側のプログラムを確認する必要がある．

XBee では，通信が正常に行われている場合には，XBee ドングル（PC 側）の LED（RX）が 1 秒間に数回点滅する（図 11.2.7）．しかし，通信用のバッファがあふれるなどして，通信がうまく行われていない場合には，この LED が点滅しなくなるため，XBee 周りのエラーが起こっているかどうかは，この LED を確認する必要がある．

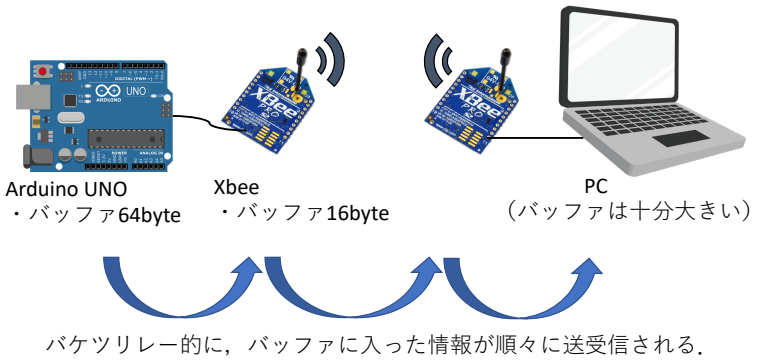

図 11.2.6　バッファの容量とデータの伝わり方

図 11.2.7　XBee の通信状態表示 LED

11.2.4　外部監視・処理サーバ（今回は Processing）の処理時間

本実験で用いられる環境のように，処理サーバ側が高速な計算機（PC など）の場合には，外部監視・処理サーバ側については，あまり処理時間を考慮する必要は無いが，それでも無計画にプログラムをしてしまうと，不必要な処理や不適切な処理を行ってしまい，想定の動作を行えない場合があるため，注意してプログラムをする必要がある．

11.2.5　実験に使用する機器

本実験に使用する機器は，前回（11-1：ロボット実験１）の実験で使用した機器と同じである．機器の詳細や，Zumo ロボットに搭載されている各種センサ，またその位置などは，前回の資料を参照されたい．

1. Zumo ロボット（単３電池４本必要）
2. Arduino UNO
3. XBee シールド
4. XBee ドングル
5. USB ケーブル（Arduino へのプログラム書き込み用）

上記の機器１～３を組み合わせて，１台のロボットにした状態が，図 11.2.9 である．

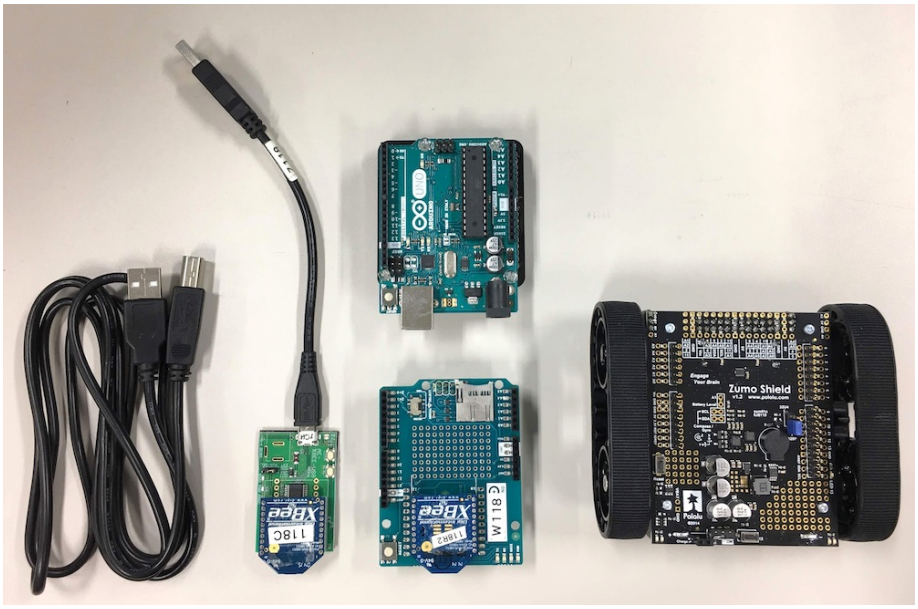

図 11.2.8　本実験で使用する機器

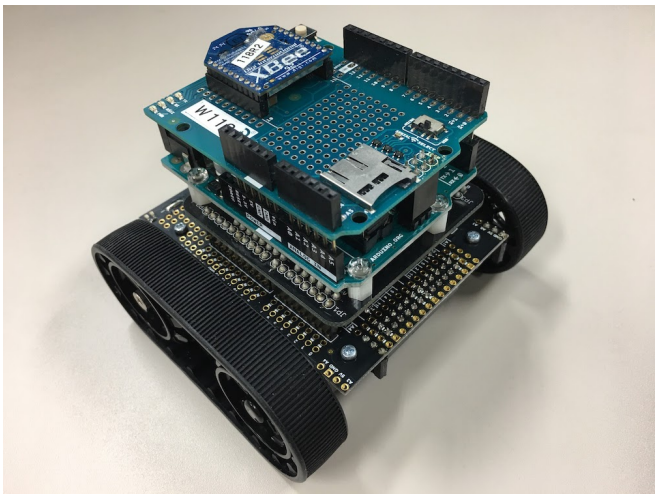

図 11.2.9　組み上げられた Zumo ロボット

11.2.6　Zumo ロボット（Arduino）へのプログラムの書き込みの際の注意

前回（11-1:ロボット実験 1 ）のテキストにおいても掲載されていたが，プログラムを書き込む際の注意事項を再掲しておく．

1. スイッチの確認
   (a) ロボット電源：OFF
   (b) XBee シールドの SERIAL SELECT スイッチ：USB 側
2. ロボット（Arduiono）と PC を USB ケーブルで接続する
3. Arduino IDE にて「検証」→「マイコンボードに書き込む」

4. USB ケーブルを抜く
5. ロボットの電源を入れる（ON にする）
6. 動作確認後，ロボットの電源を切る（OFF にする）

【Arduino UNO への書き込みに失敗する場合】

Arduino UNO へプログラムが書き込めなかった場合，以下をそれぞれ試すこと．

- Arduino IDE の設定を見直す（ボード，シリアルポート）
- Arduino IDE を再起動する
- USB ケーブルを刺し直す
- USB ポートを変更する
- USB ケーブルを変更する

11.2.7　演習

ここからは，実際に Zumo ロボットを用いた演習を進めていく．

< 演習 11.2.1 > Zumo ロボット単体の動作確認

システム開発を行う場合には，故障・バグ要素を早期に発見・切り分けるために，部分ごとの動作確認はしっかり行う必要がある．特に，ロボット開発はテストに時間的コストが大きくかかるので，動作確認は必須の作業である．この演習では，ロボット単体の動作確認を行い，データが書き込めるか，問題なく動作するかの確認を行う．

以下のプログラム 1（図 11.2.10：前回テキストの演習 11.1.3 のプログラム 3 の改変）の内容を確認し，Zumo ロボット（Arduino UNO）への書き込みを行い，ロボットを動作させる．

```
#include <Pushbutton.h>               // ボタンライブラリの読み込み
#include <ZumoBuzzer.h>               // ブザーライブラリの読み込み

const int buzzerPin = 3;              // ブザーは 3 番ピン
const int ledPin = 13;                // LED は 13 番ピン

Pushbutton button(ZUMO_BUTTON);       // Pushbutton クラスのインスタンスを生成
ZumoBuzzer buzzer;                    // ZumoBuzzer クラスのインスタンスを生成

void setup() {
  pinMode(ledPin, OUTPUT);            // 13 番ピンを出力モードに設定
  button.waitForButton();             // ユーザボタンが押されるまで待機
  buzzer.play("L16 cdegreg4");        // ブザーにて音楽を鳴らす
  delay(500);                         // 500ms待つ
}

void loop() {
  if (button.isPressed() == true) {   // ボタンが押されている状態なら
    digitalWrite(ledPin, HIGH);       // LED を点灯
    buzzer.play(">>a32");             // 音を鳴らす
  } else {                            // ボタンが押されていなかったら
    digitalWrite(ledPin, LOW);        // LED を消灯
  }
}
```

図 11.2.10　プログラム 1 ：LED 点灯，サウンド再生のテスト

このプログラムでは，ZumoBuzzer ライブラリを用いてメロディーを再生する機能を紹介している．ロボットの動作としては，ロボット起動後，ユーザボタンを押すとメロディーが再生され，その後は，ユーザボタンを押している間，ブザー音を再生しながら本体 LED を点灯させる．以下に，本演習のロボット動作テストの手順を示す．

1. スイッチの確認
   (a) ロボット電源：OFF
   (b) XBee シールドの SERIAL SELECT スイッチ：USB 側
2. ロボット（Arduiono）と PC を USB ケーブルで接続する
3.「ツール」→「シリアルポート」が，Arduino UNO のポートになっていることを確認
4. Arduino IDE にて「検証」→「マイコンボードに書き込む」
5. USB ケーブルを抜く
6. ロボットの電源を入れる（ON にする）
   (a) ユーザボタンを 1 回押す：メロディーが流れる
   (b) ユーザボタンを押す：押している間，ブザー音再生，LED 点灯
7. 動作確認後，ロボットの電源を切る（OFF にする）

＜ 演習 11.2.2 ＞ シリアル通信の動作確認

次に，シリアル通信の動作確認を行う．本実験で用いるロボットと PC とのシリアル通信には，XBee を用いる．Zumo と XBee の接続には，XBee シールドを用い，PC と XBee の接続には，USB ドングルと USB ケーブルを用いる．本演習で用いるプログラムを図 11.2.11 に示す．

```
1  int cnt = 0;                              // 送信回数の確認用カウンタ
2
3  void setup()
4  {
5    Serial.begin(9600);                     // シリアル通信の初期化
6  }
7
8  void loop()
9  {
10   String data = "count:" + String(cnt); // 送信回数送信用の文を作成
11   Serial.println(data);                 // シリアル通信により文字列送信（改行コード有り）
12
13   cnt += 1;                             // 送信回数のカウンタをインクリメント
14   delay(50);                            // 50ms待つ
15 }
```

図 11.2.11　プログラム２：シリアル通信のテスト

以下に，本演習の手順を示す．

1. スイッチの確認
   (a) ロボット電源：OFF
   (b) XBee シールドの SERIAL SELECT スイッチ：USB 側
2. ロボット（Arduiono）と PC を USB ケーブルで接続する
3.「ツール」→「シリアルポート」が，Arduino UNO のポートになっていることを確認
4. Arduino IDE にて「検証」→「マイコンボードに書き込む」
5. USB ケーブルを抜く
6. XBee ドングルを PC に USB 接続する
7. シリアル通信の内容を受信・表示するプログラムを起動する
   - （この演習の末尾に，シリアル通信を受信・表示する方法を紹介してある）
8. ロボットの XBee シールドの SERIAL SELECT スイッチを MICRO 側に切り替える
9. ロボットの電源を入れる（ON にする）
   (a) PC のシリアル通信の受信・表示プログラムに受信内容が表示される
10. 動作確認後，ロボットの電源を切る（OFF にする）

シリアル通信の結果を受信して確認するためには，いくつか方法があるが，ここでは，２つの方法を紹介する．

【Arduino IDE によるシリアル通信の確認】

Arduino IDE を用いてシリアル通信の内容を確認するためには，「ツール」内に含まれる「シリアルモニタ」を用いる（図 11.2.12）．

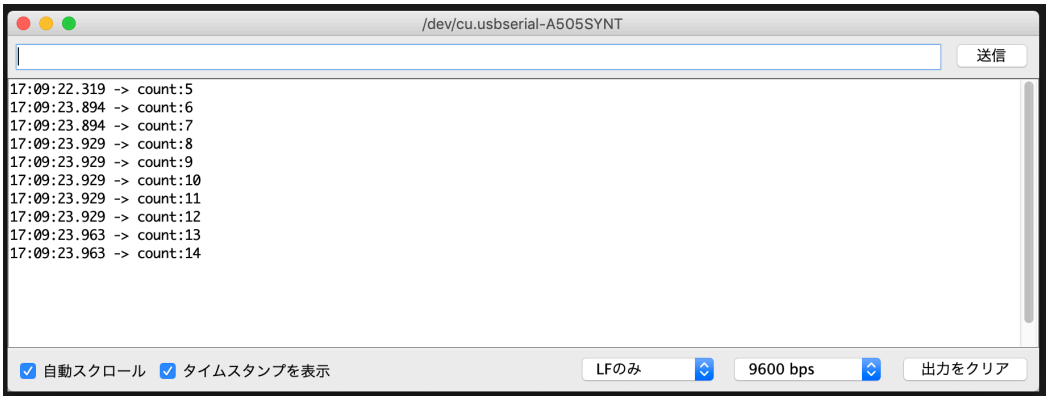

図 11.2.12　Arduino IDE によるシリアル通信の確認（シリアルモニタ）

使い方は，以下の通りである．

1. Arduino IDE を起動する
2.「ツール」→「シリアルポート」から，XBee が接続されたシリアルポートを選択する
3.「ツール」→「シリアルモニタ」を選択して起動する

【screen によるシリアル通信の確認】

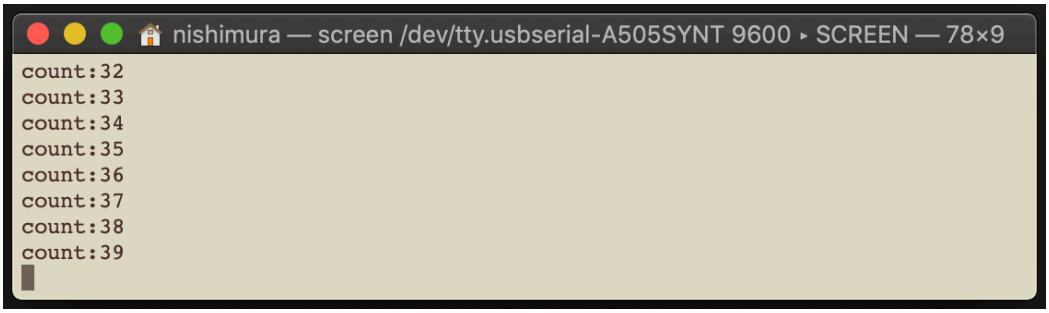

図 11.2.13　screen によるシリアル通信の確認

Linux 環境や，Mac 環境において，screen というプログラムが導入されていれば，これを用いてシリアル通信の内容を確認することが可能である．ターミナルで以下のように入力することで，シリアルポートの受信状態となり，内容が表示される（図 11.2.13）．引数の 1 番目がシリアルポートの指定である（各自の環境に合わせて書き換えること）．２番目が通信速度の指定である．

% screen /dev/tty/usbserial-A505SYNT 9600

なお，このプログラムを閉じる際には，Ctrl+a → Ctrl+k と押せば良い．その後，「Really kill this window [y/n]」などと画面に出た場合には，y を押せばプログラムが終了する．

< 演習 11.2.3 > 地磁気センサの利用とシリアル送受信

この演習では，ロボットのユーザボタンを押すことで，地磁気センサの値を取得し，その値を文字列としてシリアル送信し，PC で受信して値の確認を行う．本演習で用いるプログラムを図 11.2.14 に示す．

```
#include <ZumoMotors.h>           // Zumo用モータ制御ライブラリの読み込み
#include <Pushbutton.h>           // Zumo用ユーザボタンライブラリの読み込み
#include <Wire.h>                 // I2C/TWI 通信デバイスの制御用ライブラリの読み込み
#include <LSM303.h>               // 加速度，地磁気センサ用ライブラリの読み込み

int cnt = 0;                      // 送信回数の確認用カウンタ

#define CRB_REG_M_2_5GAUSS 0x60   // CRB_REG_M の値 ： 地磁気センサーのスケールを +/-2.5 ガウスに設定
#define CRA_REG_M_220HZ    0x1C   // CRA_REG_M の値 ： 地磁気センサのアップデートレートを 220 Hz に設定

Pushbutton button(ZUMO_BUTTON);
LSM303 compass;

void setup()
{
  Serial.begin(9600);             // シリアル通信の初期化
  Wire.begin();                   // Wireライブラリの初期化と，I2Cバスとの接続

  compass.init();                 // LSM303 の初期化
  compass.enableDefault();        // 加速度センサ・地磁気センサ を利用可能にする
  compass.writeReg(LSM303::CRB_REG_M, CRB_REG_M_2_5GAUSS); // 地磁気センサーのスケールを +/-2.5 ガウスに設定
  compass.writeReg(LSM303::CRA_REG_M, CRA_REG_M_220HZ);    // 地磁気センサのアップデートレートを 220 Hz に設定

  button.waitForButton();         // ユーザボタンが押されるまで待つ
  Serial.println("START");        // 文字列をシリアル送信
}

void loop()
{
  button.waitForButton();         // ユーザボタンが押されるまで待つ

  compass.read();                 // 地磁気センサの値を読む
  Serial.print(compass.m.x);      // 地磁気センサから得られた値(x)をシリアル送信
  Serial.print(",");              // 区切り文字（,）をシリアル送信
  Serial.print(compass.m.y);      // 地磁気センサから得られた値(y)をシリアル送信
  Serial.print(",");              // 区切り文字（,）をシリアル送信

  Serial.println(String(cnt));    // 送信回数カウンタをシリアル送信
  cnt += 1;                       // 送信回数カウンタをインクリメント
}
```

図 11.2.14　プログラム 3：地磁気センサの値を取得し，シリアル送信する

このプログラムでは，地磁気センサの使い方（キャリブレーションなどは無く，単純にセンサの値を取得するのみ）を紹介している．プログラムの動作としては，ロボット起動後，ユーザボタンを 1 度押すと，シリアル通信で「START」と出力し，その後は，ユーザボタンを押すたびに，地磁気センサの値を取得し，シリアル通信で送信する．動作確認の際には，ロボットの筐体をいろいろな向きに向けて，ユーザボタンを押し，値が変わることを確かめること．

以下に，本演習の手順を示す．

1. スイッチの確認
   (a) ロボット電源：OFF
   (b) XBee シールドの SERIAL SELECT スイッチ：USB 側
2. ロボット（Arduiono）と PC を USB ケーブルで接続する
3.「ツール」→「シリアルポート」が，Arduino UNO のポートになっていることを確認
4. Arduino IDE にて「検証」→「マイコンボードに書き込む」
5. USB ケーブルを抜く
6. XBee ドングルを PC に USB 接続する
7. シリアル通信の内容を受信・表示するプログラムを起動する

8. ロボットの XBee シールドの SERIAL SELECT スイッチを MICRO 側に切り替える
9. ロボットの電源を入れる（ON にする）
   (a) ユーザボタンを押すと，PC のシリアル通信の受信・表示プログラムに「START」と表示される
   (b) ２度目以降は，ユーザボタンを押すたびに，地磁気センサの値を取得し，シリアル送信する．
10. 動作確認後，ロボットの電源を切る（OFF にする）

< 演習 11.2.4 > プログラム中の経過時間の計測

この演習では，プログラム中の特定の範囲の処理時間の計測を行う．本演習で用いるプログラムを図 11.2.15 に示す．

```
#include <ZumoMotors.h>           // Zumo用モータ制御ライブラリの読み込み
#include <Pushbutton.h>           // Zumo用ユーザボタンライブラリの読み込み
#include <Wire.h>                 // I2C/TWI 通信デバイスの制御用ライブラリの読み込み
#include <LSM303.h>               // 加速度，地磁気センサ用ライブラリの読み込み

unsigned long int timeStart = 0;  // 時間計測用変数
unsigned long int timeEnd = 0;    // 時間計測用変数

int cnt = 0;                      // 送信回数の確認用カウンタ

#define CRB_REG_M_2_5GAUSS 0x60   // CRB_REG_M の値 ： 地磁気センサーのスケールを +/-2.5 ガウスに設定
#define CRA_REG_M_220HZ    0x1C   // CRA_REG_M の値 ： 地磁気センサのアップデートレートを 220 Hz に設定

Pushbutton button(ZUMO_BUTTON);
LSM303 compass;

void setup()
{
  Serial.begin(9600);             // シリアル通信の初期化
  Wire.begin();                   // Wireライブラリの初期化と，I2Cバスとの接続

  compass.init();                 // LSM303 の初期化
  compass.enableDefault();        // 加速度センサ・地磁気センサ を利用可能にする
  compass.writeReg(LSM303::CRB_REG_M, CRB_REG_M_2_5GAUSS); // 地磁気センサーのスケールを +/-2.5 ガウスに設定
  compass.writeReg(LSM303::CRA_REG_M, CRA_REG_M_220HZ);    // 地磁気センサのアップデートレートを 220 Hz に設定

  button.waitForButton();         // ユーザボタンが押されるまで待つ
  Serial.println("START");        // 文字列をシリアル送信
}

void loop()
{
  button.waitForButton();         // ユーザボタンが押されるまで待つ

  timeStart = millis();           // 現在の時間取得
  compass.read();                 // 地磁気センサの値を読む
  Serial.print(compass.m.x);      // 地磁気センサから得られた値(x)をシリアル送信
  Serial.print(",");              // 区切り文字（,）をシリアル送信
  Serial.print(compass.m.y);      // 地磁気センサから得られた値(y)をシリアル送信
  Serial.println();               // 改行コードをシリアル送信
  timeEnd = millis();             // 現在の時間取得

  delay(50);                      // 150ms待つ
  Serial.println(timeEnd-timeStart);    // 経過時間をシリアル送信

}
```

図 11.2.15　プログラム４：プログラム中の時間計測

プログラム４は，プログラム３の地磁気センサを用いるものとほとんど同様であるが，数行の追加を行った（6，7，３５，４１行目など）．Arduino では，表 11.2.3 に示す時間計測用関数が用意されている．

プログラム４では，地磁気センサを読み，その結果をシリアル送信するまでの間の時間を計測している．経過時間は，再度シリアル送信を行い，確認できるようにしている．プログラム４では，millis() を用いてミリ秒で計測を行っているが，millis() 部分を micros() に変更し，結果がどのように変わるかも確認すること．

以下に，本演習の手順を示す．

表 11.2.3　Arduino での時間計測用関数

関数	処理内容
millis()	プログラム起動後から現在までの時間をミリ秒で返す（型：unsigned long）
micros()	プログラム起動後から現在までの時間をマイクロ秒で返す（型：unsigned long）

1. スイッチの確認
   (a) ロボット電源：OFF
   (b) XBee シールドの SERIAL SELECT スイッチ：USB 側
2. ロボット（Arduiono）と PC を USB ケーブルで接続する
3.「ツール」→「シリアルポート」が，Arduino UNO のポートになっていることを確認
4. Arduino IDE にて「検証」→「マイコンボードに書き込む」
5. USB ケーブルを抜く
6. XBee ドングルを PC に USB 接続する
7. シリアル通信の内容を受信・表示するプログラムを起動する
8. ロボットの XBee シールドの SERIAL SELECT スイッチを MICRO 側に切り替える
9. ロボットの電源を入れる（ON にする）
   (a) ユーザボタンを押すと，PC のシリアル通信の受信・表示プログラムに「START」と表示される
   (b) ２度目以降は，ユーザボタンを押すたびに，地磁気センサの値と，経過時間がシリアル送信される
10. 動作確認後，ロボットの電源を切る（OFF にする）

< 演習 11.2.5 > Processing との通信

この演習では，Processing と Zumo ロボットとの通信を行う．Zumo ロボットの地磁気センサの値を Processing にシリアル送信して，その値をテキストで表示する．Zumo 側のプログラムは，演習 11.2.3 で用いたプログラム 3 を再度用いる．

本演習での，結果をテキストで表示する Processing のプログラムを図 11.2.16 に示す．

プログラムの１０行目にて，シリアル通信のポートを指定している（/dev/cu.usbserial-A505SYNT）が，ここは各自の環境に合わせて書き換えること．また，プログラム中では，シリアル通信中に，データが途中で途切れて送受信された場合に対処するために，改行コードまでデータをバッファする処理が行われている（１２行目）．更に，データを読み込む際にも，改行コードまでをまとめて読み込むようにしている（２９行目）．

以下に，本演習の手順を示す．

1. スイッチの確認
   (a) ロボット電源：OFF
   (b) XBee シールドの SERIAL SELECT スイッチ：USB 側
2. ロボット（Arduiono）と PC を USB ケーブルで接続する
3.「ツール」→「シリアルポート」が，Arduino UNO のポートになっていることを確認
4. Arduino IDE にて「検証」→「マイコンボードに書き込む」
5. USB ケーブルを抜く
6. XBee ドングルを PC に USB 接続する
7. Processing を起動し，プログラムを実行する（テキスト表示用ウィンドウが表示される）
8. ロボットの XBee シールドの SERIAL SELECT スイッチを MICRO 側に切り替える

```
1  import processing.serial.*;
2
3  Serial port1;
4
5  String myString1 = null;
6  int LF = 10;                                                    // LF（Linefeed）のアスキーコード
7
8  void setup() {
9    size(600, 200);                                               //幅 600px, 高さ 200px のウインドウを生成
10   port1 = new Serial(this, "/dev/cu.usbserial-A505SYNT", 9600); //Serial クラスのインスタンスを生成
11   port1.clear();
12   port1.bufferUntil(0x0d);                                      // LF = 0x0d までバッファ
13 }
14
15 void draw() {
16   background(0);                                                 //背景色を黒に
17   textSize(30);                                                  //文字の大きさを 30 に
18   textAlign(CENTER,CENTER);                                      //文字の配置をウインドウの中心に
19
20   if(myString1 != null){
21     text(myString1, 200, 100);
22   }
23
24 }
25
26 // シリアルポートにデータが到着するたびに呼び出される割り込み関数
27 void serialEvent(Serial p) {
28   if (p == port1 && (p.available() > 0)) {
29     myString1 = port1.readStringUntil(LF);                       //文字データの最後まで読み込み
30     myString1 = port1.readString();
31     if(myString1 != null){                                       //文字が入ってたら
32       myString1 = trim(myString1);                               //行末の改行 '¥n' を削除
33     }
34   }
35
36 }
```

図 11.2.16　プログラム 5 ：Processing によるテキスト表示

9. ロボットの電源を入れる（ON にする）
   (a) ユーザボタンを押すと，Processing の画面に「START」と表示される
   (b) ２度目以降は，ユーザボタンを押すたびに，地磁気センサの値が表示される
10. 動作確認後，ロボットの電源を切る（OFF にする）
11. Processing の「停止」ボタンを押し，プログラムを停止させる

< 演習 11.2.6 > Processing によるグラフ描画

この演習では，Processing によるグラフィカルな結果表示を行う．Zumo ロボット側のプログラムを図 11.2.17 に示す．このプログラムでは，Zumo ロボットから地磁気センサの値を連続的に送る．ロボットを起動し，ユーザボタンを 1 度押すと，それ以降，連続的に地磁気センサの値をシリアル送信する．

Processing 側のプログラムを図 11.2.18 に示す．このプログラムでは，シリアル送信された値（カンマで区切られた３つの値）を受信して，その値をグラフィカルに表示する．本プログラムは，当初カラーセンサ表示を想定していたため，「8-1，8-2 センサ実験３」にて構築した RGB 表示の画面構成になっている．ロボットから取得する値は地磁気センサの値とカウンタであるため，これを RGB の値とみなして処理している．なお，このプログラムにおいても，前回の演習同様，プログラムの１１行目にて，シリアル通信のポートを指定している（/dev/cu.usbserial-A505SYNT）が，ここは各自の環境に合わせて書き換えること．

ロボットと Programing を起動後，ロボットを動かすと，Processing のグラフ表示が変化することが確認できるはずである．

以下に，本演習の手順を示す．

```
#include <ZumoMotors.h>           // Zumo用モータ制御ライブラリの読み込み
#include <Pushbutton.h>           // Zumo用ユーザボタンライブラリの読み込み
#include <Wire.h>                 // I2C/TWI 通信デバイスの制御用ライブラリの読み込み
#include <LSM303.h>               // 加速度，地磁気センサ用ライブラリの読み込み

int cnt = 0;                      // 送信回数の確認用カウンタ

#define CRB_REG_M_2_5GAUSS 0x60   // CRB_REG_M の値 ： 地磁気センサーのスケールを +/-2.5 ガウスに設定
#define CRA_REG_M_220HZ    0x1C   // CRA_REG_M の値 ： 地磁気センサのアップデートレートを 220 Hz に設定

Pushbutton button(ZUMO_BUTTON);
LSM303 compass;

void setup()
{
  Serial.begin(9600);             // シリアル通信の初期化
  Wire.begin();                   // Wireライブラリの初期化と，I2Cバスとの接続

  compass.init();                 // LSM303 の初期化
  compass.enableDefault();        // 加速度センサ・地磁気センサ を利用可能にする
  compass.writeReg(LSM303::CRB_REG_M, CRB_REG_M_2_5GAUSS); // 地磁気センサーのスケールを +/-2.5 ガウスに設定
  compass.writeReg(LSM303::CRA_REG_M, CRA_REG_M_220HZ);    // 地磁気センサのアップデートレートを 220 Hz に設定

  button.waitForButton();         // ユーザボタンが押されるまで待つ
}

void loop()
{
  compass.read();                 // 地磁気センサの値を読む
  Serial.print(compass.m.x);      // 地磁気センサから得られた値(x)をシリアル送信
  Serial.print(",");              // 区切り文字（,）をシリアル送信
  Serial.print(compass.m.y);      // 地磁気センサから得られた値(y)をシリアル送信
  Serial.print(",");              // 区切り文字（,）をシリアル送信

  Serial.println(String(cnt));    // 送信回数カウンタをシリアル送信
  cnt += 1;                       // 送信回数カウンタをインクリメント
  delay(100);                     // 100ms 待つ
}
```

図 11.2.17　プログラム 6 ：地磁気センサの値を連続的に送る（100ms の delay）

1. スイッチの確認
   (a) ロボット電源：OFF
   (b) XBee シールドの SERIAL SELECT スイッチ：USB 側
2. ロボット（Arduiono）と PC を USB ケーブルで接続する
3.「ツール」→「シリアルポート」が，Arduino UNO のポートになっていることを確認
4. Arduino IDE にて「検証」→「マイコンボードに書き込む」
5. USB ケーブルを抜く
6. XBee ドングルを PC に USB 接続する
7. Processing を起動し，プログラムを実行する（グラフ表示用ウィンドウが表示される）
8. ロボットの XBee シールドの SERIAL SELECT スイッチを MICRO 側に切り替える
9. ロボットの電源を入れる（ON にする）
   (a) ユーザボタンを押すと，地磁気センサの値が連続的に送信される
10. 動作確認後，ロボットの電源を切る（OFF にする）
11. Processing の「停止」ボタンを押し，プログラムを停止させる

```
import processing.serial.*;

Serial port1;

String myString1 = null;
float red=0, green=0, blue=0;
int LF = 10;                                                    // LF (Linefeed) のアスキーコード

void setup() {
  size(640, 480);                                               // 幅 640px, 高さ 480px のウインドウを生成
  port1 = new Serial(this, "/dev/cu.usbserial-A505SYNT", 9600);  // Serial クラスのインスタンスを生成
  port1.clear();
  port1.bufferUntil(0x0d);                                      // LF = 0x0d までバッファ
}

void draw() {
  background(0);                                                // 背景色を黒に
  textAlign(CENTER,CENTER);                                     // 文字の配置をウインドウの中心に

  // グラフの表示．　色の値をｙ軸の向きに逆に取ることで，上向きに上下するグラフを描画している
  fill(red,0,0); rect(10,470,100,-red);                         // 赤色のグラフ
  fill(0,green,0); rect(120,470,100,-green);                    // 緑色のグラフ
  fill(0,0,blue); rect(230,470,100,-blue);                      // 青色のグラフ
  fill(red, green, blue); rect(360,110,260,260);                // RGB値を用いた四角描画

  // RGB値を，テキストで画面に表示（グラフの上）
  fill(255);
  text("red=",20,10); text(int(red),40,10);
  text("green=",130,10); text(int(green),160,10);
  text("blue=",240,10); text(int(blue),265,10);

  if(myString1 != null){
    text(myString1, 200, 100);                                  // シリアル通信で受信したテキストの表示
  }
}

// シリアルポートにデータが到着するたびに呼び出される割り込み関数
void serialEvent(Serial p) {
  if (p == port1 && (p.available() > 0)) {
    myString1 = port1.readStringUntil(LF);                      // 文字データの最後まで読み込み
    myString1 = port1.readString();
    if(myString1 != null){                                      // 文字が入ってたら
      myString1 = trim(myString1);                              // 行末の改行 '¥n' を削除

      float data[] = float(split(myString1, ','));              // カンマで区切られた値を分割
      if (data.length == 3){                                    // 受信した数値の数が３つだったら
        // キャリブレーションあり
        red = map(data[0], -5000, 5000, 0, 255);                // 値を0〜255にマッピング
        green = map(data[1], -5000, 5000, 0, 255);              // 値を0〜255にマッピング
        blue = map(data[2], -5000, 5000, 0, 255);               // 値を0〜255にマッピング
      }
    }
  }
}
```

図 11.2.18　プログラム 7：Processing によるグラフ描画

< 演習 11.2.7 > Processing の画面を分割して表示

この演習では，Processing 上で複数のロボットの情報を表示するべく，画面分割を行って，グラフ表示を行う．Zumo ロボット用のプログラムは，前回の演習で用いたもの（プログラム 6）をそのまま用いる．

Processing による画面分割表示プログラムを，図 11.2.19 に示す．このプログラムでは，画面を 4 つに分割し，1 台目のロボットの情報を左上に表示する．データの受信方法や表示方法は，前回の演習のプログラム（プログラム 7）と同様である．なお，プログラム中には，2 台の Zumo を接続して表示する際に，どの部分にプログラムを追記すればよいかのヒントを示してある（２０，４５行目）．

```
import processing.serial.*;

Serial port1;                                                    // 1台目のZumoのシリアル通信用
Serial port2;                                                    // 2台目のZumoのシリアル通信用

String myString1 = null;
float red=0, green=0, blue=0;
int LF = 10;                                                     // LF（Linefeed）のアスキーコード

int zumo_id = 0;

int graph_width =100;                                            // グラフの幅を定義

void setup() {
  size(1200, 800);                                               // 幅 1200px，高さ 800px のウインドウを生成
  port1 = new Serial(this, "/dev/cu.usbserial-A505SYNT", 9600);  // Serial クラスのインスタンスを生成
  port1.clear();
  port1.bufferUntil(0x0d);                                       // LF = 0x0d までバッファ

  // ＊＊＊　ヒント：ここでport2の「ポート指定，クリア，LFまでバッファ」を行う　＊＊＊//

  background(0);                                                 // 背景色を黒に

  fill(100, 100, 100); rect(width/2, 0, width/2, height/2);      // 右上の領域を塗りつぶす
  fill(200, 200, 200); rect(0, height/2, width/2, height/2);     // 左下の領域を塗りつぶす
  fill(150, 150, 150); rect(width/2, height/2, width/2, height/2);  // 右下の領域を塗りつぶす
}

void draw() {
  if(zumo_id == 1){                                              // データを受信したときだけ書き換える（1番目の
       Zumo)
    fill(0, 0, 0); rect(0,0, width/2,height/2);                  // 対象画面の初期化（黒く塗りつぶす）

    // グラフなどの描画
    fill(red,0,0); rect(10, height/2, 100, -red);
    fill(0,green,0); rect(120, height/2, 100,-green);
    fill(0,0,blue); rect(230,height/2, 100,-blue);

    fill(red, green, blue); rect(350,30,200,200);

    if(myString1 != null){
      text(myString1, 10, 10);                                   // シリアル通信で受信したテキストの表示
    }
  }

  // ＊＊＊　ヒント：ここに2番目のZumo用の処理を書く　＊＊＊//

}

// シリアルポートにデータが到着するたびに呼び出される割り込み関数
void serialEvent(Serial p) {
  if ((p == port1 || p == port2) && (p.available() > 0)) {       // 割り込みシリアル通信が，
       port1か，port2で，なおかつデータが入っている時
    zumo_id = 1;                                                 // データをやり取りしたロボットのIDを記憶

    if(p == port1)  zumo_id = 1;                                 // データをやり取りしたロボットのIDを記憶
    else if(p == port2) zumo_id = 2;

    myString1 = port1.readStringUntil(LF);                       //文字データの最後まで読み込み
    if(myString1 != null){                                       //文字が入ってたら
      myString1 = trim(myString1);                               //行末の改行 '¥n' を削除

      float data[] = float(split(myString1, ','));               // カンマで区切られた値を分割

      // 受信した数値の数が３つで，NaN（Not a Number）ではない時
      if (data.length == 3 && data[0] != Float.NaN && data[1] != Float.NaN && data[2] != Float.NaN){
        // キャリブレーションあり
        red = map(data[0], -5000, 1000, 0, 255);                 // 値を0〜255にマッピング
        green = map(data[1], -5000, 1000, 0, 255);               // 値を0〜255にマッピング
        blue = map(data[2], -5000, 1000, 0, 255);                // 値を0〜255にマッピング
      }
    }
  }
}
```

図 11.2.19　プログラム８：Processing による画面分割表示

以下に，本演習の手順を示す．

1. スイッチの確認
   (a) ロボット電源：OFF
   (b) XBee シールドの SERIAL SELECT スイッチ：USB 側
2. ロボット（Arduiono）と PC を USB ケーブルで接続する
3.「ツール」→「シリアルポート」が，Arduino UNO のポートになっていることを確認
4. Arduino IDE にて「検証」→「マイコンボードに書き込む」(前回のままプログラムがロボット上に残っている場合には書き換える必要は無い)
5. USB ケーブルを抜く
6. XBee ドングルを PC に USB 接続する
7. Processing を起動し，プログラムを実行する（4 分割のグラフ表示用ウィンドウが表示される）
8. ロボットの XBee シールドの SERIAL SELECT スイッチを MICRO 側に切り替える
9. ロボットの電源を入れる（ON にする）
   (a) ユーザボタンを押すと，地磁気センサの値が連続的に送信される
10. 動作確認後，ロボットの電源を切る（OFF にする）
11. Processing の「停止」ボタンを押し，プログラムを停止させる

< 演習 11.2.8 > 複数台のロボットの接続方法

複数台の Zumo ロボットと PC の接続は，1 台のロボットの接続方法と同様で，対応する XBee 用 USB ドングルを PC の USB ポートに接続するだけで良い．2 台または 3 台の Zumo ロボットと PC を接続する場合には，USB ポートに XBee 用 USB ドングルを刺し，対応するシリアルポートを確認すること．確認方法は，これまで 1 台の Zumo ロボットで行ってきた方法と同様である．例えば，Arduino IDE の「ツール」→「シリアルポート」で確認する方法や，Terminal 上で「 ls /dev/cu.*」と入力して，一覧を表示する方法などがある．

11.2.8　課題

＜ 課題 11.2.1 ＞ シリアル通信

演習 11.2.2 でのシリアル通信のプログラム（プログラム２）では，プログラム中で 50ms の delay を行っていた．この delay を行わない場合（コメントアウトなどして削除した場合），シリアル通信の結果がどのように変化するか確認せよ．（しばらく表示させ，表示結果が変わることを確認する．）

＜ 課題 11.2.2 ＞ 時間計測

演習 11.2.4 にて，地磁気センサとシリアル通信に対して，プログラム中の経過時間の計測を行った．本課題では，Serial.write() と Serial.println() のそれぞれにかかる時間を計測し，比較をせよ．時間計測は，millis() と micros() の両方で行うこと．

＜ 課題 11.2.3 ＞ Processing と 2 台の Zumo ロボットとの通信

演習 11.2.7 にて，画面を 4 分割して 1 台の Zumo の情報のグラフ描画を行ったが，本課題では，このプログラムを拡張させ，2 台の Zumo の情報を同時に Processing 上に描画せよ．ここで，2 台の Zumo を接続して動作させると，Processing にてスムーズな描画が行われないはずである．どうしてそのような問題が起こるのか調査せよ．

＜ 課題 11.2.4 ＞ 複数台のロボットの連携動作と管理

3 台の Zumo ロボットと PC を接続し，各 Zumo ロボットと Processing をシリアル通信でつなぎ，1 台の zumo マシンが何か作業をしてそれが終わると，PC に終わったことを知らせ，2 台目の zumo マシンが続いて作業し，それが終わると 3 台目のマシンが作業するプログラムを構築せよ．例えば，3 台の Zumo を並べ，モータを制御し，それぞ順に回転させていくなどさせると見た目にもわかりやすい．また，その制御状況（各 Zumo ロボットの動いている，動いていない　など）を Processing に描画せよ．

＜ 課題 11.2.5 ＞ ロボット動作の時間変化

前回のシステム実験（ロボット実験１）の課題２のプログラムを参考に，超音波センサを利用するプログラムを参考にプログラムを改変し，動作の 1 回のループにかかる時間を計測せよ．計測結果の時間（数値）はシリアル通信で出力し，確認できるようにする（シリアル通信でテキストで確認しても良い，Processing でグラフィカルに表示しても良い）．そして，検出する物体までの距離が変化すると，1 回のループにかかる時間がどのように変化するのか調査せよ．

＜ 発展課題 11.2.6 ＞ Processing と 2 台の Zumo ロボットとの安定した通信

課題 11.2.3 にて，2 台の Zumo ロボットとの通信を行い，Processing にて表示を行ったが，動作が不安定であったはずである．この問題に対処し，安定した動作が可能なプログラムを作成せよ．

＜ 発展課題 11.2.7 ＞ ロボットの安定動作と安定通信

Zumor ロボットにて，センシングする値を 3 つ以上に増やして（例えば，地磁気センサ，加速度センサ，超音波センサ），シリアル通信を行った場合，Processing に通信結果が 正しく表示されるかどうかを確認せよ．複数のセンサから値を取得して，その結果を正しく表示させることのできるプログラムを作成せよ．また，Arduino からデータ送信する間隔を変更した場合，どうなるかについても調査せよ．

11.2.9　レポート

【レポート 11.2 （2019 年 10 月 10 日出題，2019 年 10 月 17 日 12:50 提出締切)】

< レポート 11.2.1 >

課題 11.2.1 で行ったシリアル通信プログラム中の delay を削除すると，表示内容がどのように変わったか，画面のスクリーンショットを示して解説せよ．また，結果が変わった原因を考察せよ．

< レポート 11.2.2 >

課題 11.2.2 で作成したプログラムを示せ．また，行った実行時間比較の結果を示し，結果について考察せよ（実際にプログラムにこれらの関数 (Serial.write(), Serial.println()) を利用する場面を考え，その利点と注意すべき点を示すと良い)．

< レポート 11.2.3 >

課題 11.2.3 で最初に作成したプログラムを示せ（Arduino, Processing)．Processing で描画した画面のスクリーンショットも示せ．また，発生した問題（スムーズな描画が行われなくなったこと）について述べ，考察せよ．

< レポート 11.2.4 >

課題 11.2.4 で作成したプログラム（Arduino，Processing）を報告せよ．プログラムにはコメントを詳細に記述すること．

< レポート 11.2.5 >

課題 11.2.5 で作成したプログラム（Arduino と，あれば Processing）を報告せよ．プログラムにはコメントを詳細に記述すること．通信内容を確認した際の画面のスクリーンショットを示せ（Processing を用いた場合には，描画した画面のスクリーンショットを示す)．また，検出する物体までの距離の変化と，ループにかかる時間の変化との関係をまとめ，その結果について考察せよ．

< 発展課題レポート 11.2.6 >

課題 11.2.3 にて，2 台のロボットを接続したことにより発生した問題に対処するべく作成したプログラム（Arduino, Processing の両方）を報告せよ．プログラムにはコメントを詳細に記述すること．また，プログラムについて工夫した点について報告せよ．

< 発展課題レポート 11.2.7 >

発展課題 11.2.6 で作成したプログラム（Arduino，Processing）を報告せよ．プログラムにはコメントを詳細に記述すること．複数のセンサから値を取得して，その結果を正しく表示させるプログラムをどのように実現したか，アルゴリズムを解説せよ．また，次の語句を含めた考察をせよ．「センサの応答時間，1 ループの時間，バッファ，通信速度」．

使用部品一覧

部品	個数	備考
Zumo ロボット	3	
XBee ドングル	3	
XBee シールド	3	
単３電池	12	
充電器	3	
Arduino UNO	3	
USB ケーブル	3	

参考図書

[1] pololu zumo-shield, `https://github.com/pololu/zumo-shield`

[2] pololu zumo-shield-arduino-library, `https://github.com/pololu/zumo-shield-arduino-library`

[3] Pololu Zumo Shield for Arduino User's Guide, `https://www.pololu.com/docs/0J57`