ロボット博士養成講座

ロボティクスプロフェッサーコース

二足歩行ロボット②

第4回

二足歩行ロボットのプログラミング①

講師用

Human ヒューマンアカデミー ジュニア STE"AMスクール

ロボット博士養成講座 ロボティクスプロフェッサーコース

2024年2月授業分

目　　次

0.　二足歩行ロボットのプログラミング①

0.0.　「二足歩行ロボットのプログラミング①」でやること

0.1.　必要なもの

1.　超音波距離センサーの性能

1.0.　超音波距離センサーの仕様

1.1.　超音波距離センサーの動作テスト

1.2.　超音波距離センサーの問題をプログラムで解決しよう

1.3.　超音波距離センサーをライブラリで動かそう

2.　複数のデバイスを統合させよう

2.0.　超音波距離センサーとマトリクス LED を同時に動かそう

2.1.　超音波距離センサーの計測結果をマトリクス LED に表示させる

2.2.　スピーカーの機能を追加する

2.3.　コントローラーの機能を追加する

3.　まとめ

○　授業開始にあたって

授業のはじめは、着席させ、大きな声であいさつしてから始めます。

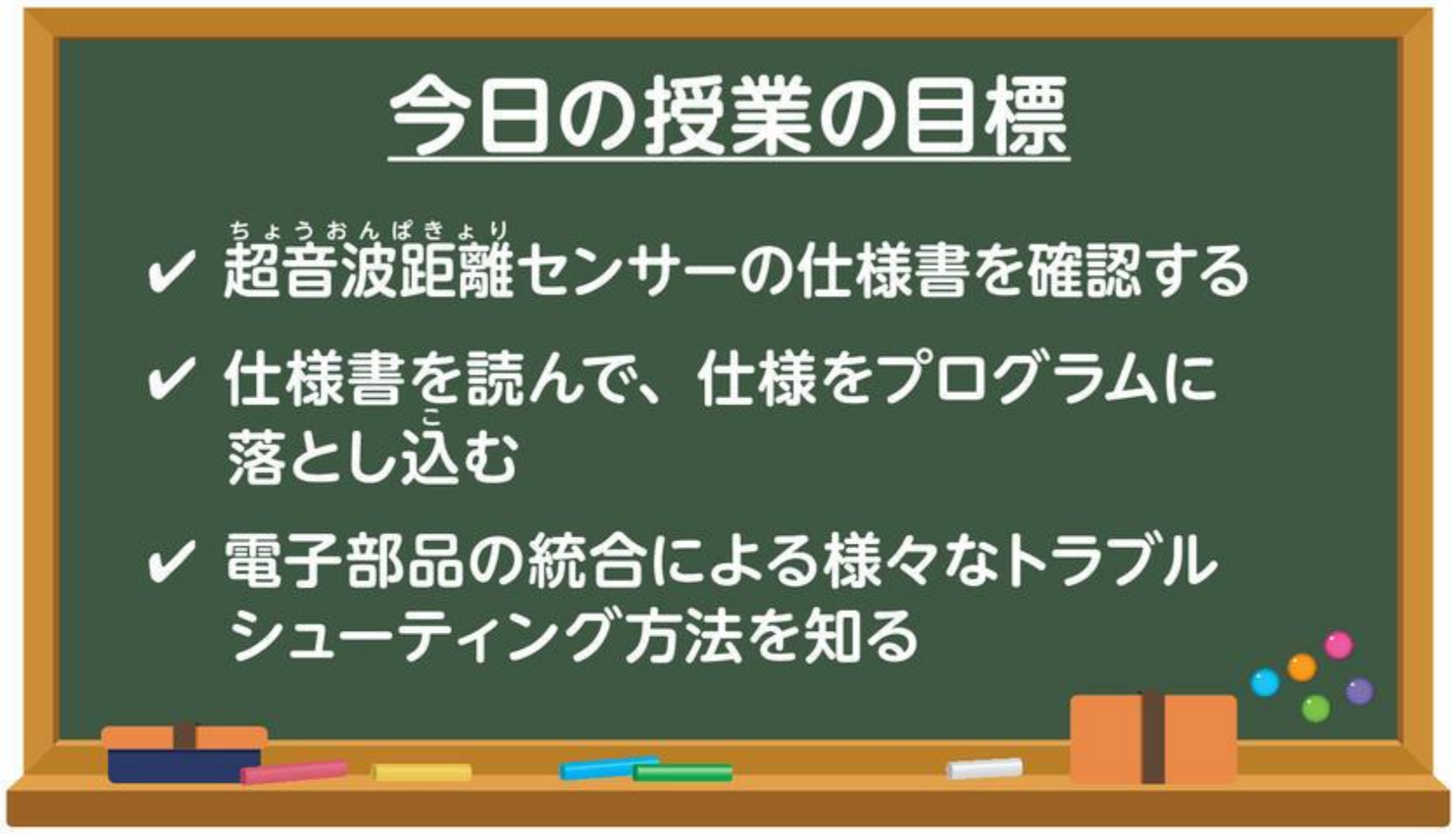
○　今回の目標をパネルで用意するか、黒板に予め書いておきます。

（授業の目標を明確化することは大変重要なことですので、生徒によく理解させます）

目安時間は授業時間 120 分のうち、休憩 10 分程度取ることを想定しています。
生徒の進捗状況により、休憩時間などを調整して授業を行ってください。

0.　二足歩行ロボットのプログラミング①（目安５分）

0.0.　「二足歩行ロボットのプログラミング①」でやること



みなさんはこれまでに、センサーやLEDなど、様々な電子機器の使い方を学んできました。
しかし、複数の電子部品を同時に接続して使用する場合、さまざまな制約が出てきます。ですが、できるだけ多くの電子部品を接続して多機能なものをつくりたいのが人間の心情ですよね。
ということで、今回はできるだけ多くの機能を使うときに考慮するポイントや、問題を回避する方法（トラブルシューティング）について学習していきましょう。また、さまざまな機能を使うためには個々の部品の仕様（性能や精度、構造、使用方法など）を知り、性能や使用方法に応じたプログラムにする必要があるので、その点についても身につけましょう。

機能をツメコミすぎてもウゴカナイノダー！

0.1.　必要なもの

第３回で完成した「二足歩行ロボット」と、以下のパーツを準備しておきましょう。

コントローラー　1	AC アダプター　1
	

図0-0　必要なもの

1．　超音波距離センサーの性能（目安６０分）

1.0.　超音波距離センサーの仕様

今回は超音波距離センサーについてくわしく見ていきます。
これまでにも何回も扱ってきたセンサーですが、皆さんももうロボプロ3年目のベテランですので、その仕組みやライブラリ・プログラムの裏側までしっかりとつかんでいきましょう。

1年目でも触れましたが、超音波距離センサーは反響定位（エコーロケーション）を利用します。
反響定位とは、何かにぶつかってはね返ってきた音を検知し、ものの位置や向きを探ることです。

超音波距離センサーには4本のピンがついていますが、電源用として使われるのはそのうち2本だけです。残る2本は「トリガー」と「エコー」とよばれる、信号の入力・出力に使われるピンです。

図1-0　超音波距離センサーのピン

講　超音波距離センサーは、ご購入の時期によっては基盤の色が緑色のものもございますが、性能はほぼ変わりません。

超音波距離センサーの仕事は大きくまとめると「マイコンの合図があったら超音波を発する」「反射した超音波をキャッチするまでの時間をはかり、結果の信号をマイコンに伝える」の2つです。
「トリガー」「エコー」の2つのピンは、それぞれ次のような仕事を行っています。

POINT

トリガー：マイコンから合図を受け取り、超音波を発する。
エコー：超音波が戻ってきたら、マイコン側に合図を送る。

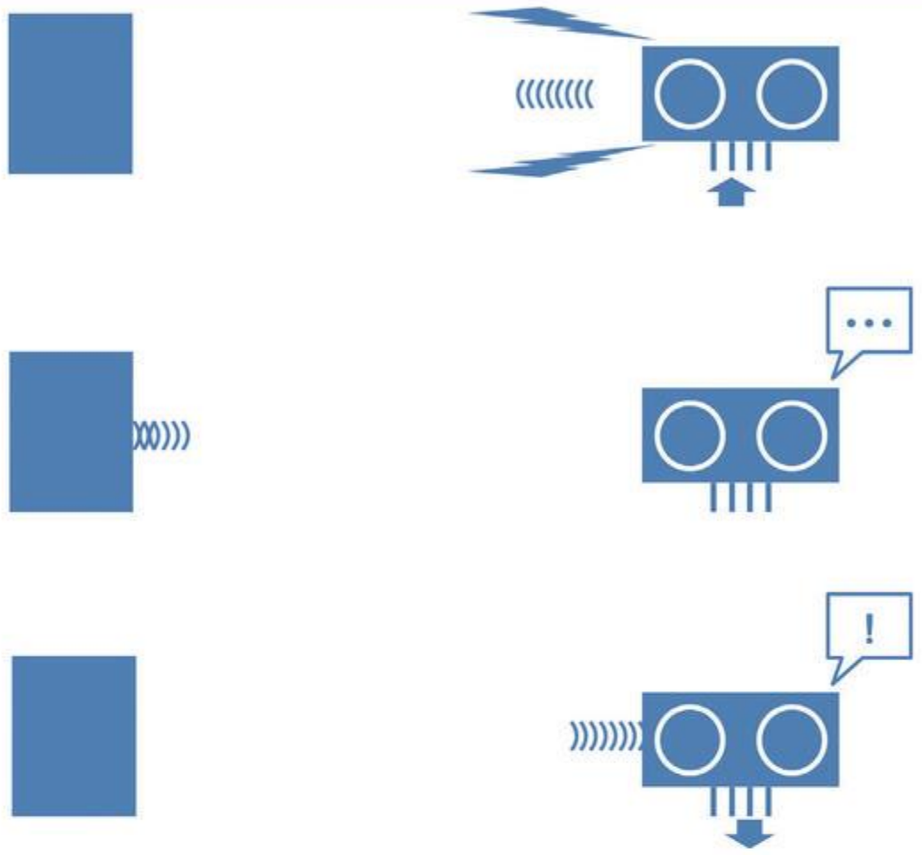


図1-1　トリガーとエコーの役割

トリガーが受け取る合図、エコーが送る合図、ともに「パルス」を用います。サーボモーターの仕組みにも登場していた、短時間だけ決まった電圧で放たれる電気信号ですね。
トリガーは、マイコンから10マイクロ秒のパルスを受け取ったら合図と見なし、超音波を発射します。エコーは、超音波が戻ってくるまでにかかった時間に応じて、マイコンに送るパルスの幅を調節します（つまり、障害物が遠いときと近いときとで、送るパルスの幅が変わります）。
マイコンは、エコーから送られてきたパルスを見れば、超音波が往復するのにかかった時間を知ることができるわけです。

…ここまで読んで、「あれ？」と思った人は、よい読解力を持っています！実は、超音波距離センサーは単体では「距離」ではなく「時間」を検知するだけの機器に過ぎないのです。
では、なぜ今まで距離をはかってこられたのでしょうか。
これは、「時間」をもとに「距離」を算出するようプログラムにかいておき、マイコンがプログラム通りに計算を行っていたからです。つまり、超音波距離センサーはマイコンの手伝いがあって初めて「距離」をはかることができるというわけです。

時間をヒントに距離を求める、というのは、実は皆さんも算数（や数学）の授業で何度もやっていますよね。
今回は、算数（や数学）の問題ではなかなか登場しない単位で計算をすることになるので、プログラムに移る前に少し練習しておきましょう。

やってみよう！

センサーが超音波を発してから、障害物に反射して戻ってくるまでに6000マイクロ秒かかった。音速を340m/sとすると、センサーと障害物との距離は何cmかな？

102 cm

ヒント

音速の単位がm/s、つまり「メートル」と「秒」を基準にしているのに、時間の単位は「マイクロ秒」だし、答えの単位は「cm」だね。このままでは計算できないので、まずは「1秒あたり340m」と表している音速を、「1マイクロ秒あたり●cm」に変換してみよう！ 1mは100cm、1秒は1000000マイクロ秒だよ！
また、反響定位では音が「往復している」ことに注意しよう！

速さの単位変換が少しややこしいですね。

POINT

・340m = 34000cmなので、「1秒あたり340m」は「1秒あたり34000cm」となる。
・1秒 = 1000000マイクロ秒なので、「1秒あたり34000cm」は「1マイクロ秒あたり0.034cm」となる。

よって、340m/s = 0.034cm/μs （「μ」はマイクロを表す文字です）

これで、計算に使える「速さ」が手に入りました。あとは「距離＝速さ×時間」なので、2つの数をかけるだけです。出てくるのが往復の距離なので、片道の距離に直すため÷2もつけ足しておきましょう。

6000×0.034÷2＝102

よって、障害物との距離は102cmであることがわかります。

音速である0.034も、往復を片道に直すための÷2も、はかった時間に関係なくずっと一定の値です。よって、「障害物までの距離＝超音波距離センサーがはかった時間×0.034÷2」となりますね。
ただ、プログラムに式をかくとき、なるべくシンプルにしたいので以下のように直しています。
「障害物までの距離＝超音波距離センサーがはかった時間÷58」

「×0.034÷2」の部分を「÷58」に直しました。若干の違いはありますが、どちらの式もかなり近い値の答えが出てきます。
今回は小数点以下の細かな数値まであまり気にしても仕方ないこと、プログラムをなるべくシンプルにまとめたいことから、このような省略が行われています。

1.1. 超音波距離センサーの動作テスト

一通り仕様を理解したところで、超音波距離センサーの原理を頭に置きながら動作を確認してみましょう。二足歩行ロボットのマイコンボードにUSBケーブルを接続しパソコンと通信状態にしてください。準備ができたら、次のプログラムを実行してシリアルモニターを開いてから通信速度を9600baudに設定してください。

プログラムの書き込み

RoboticsProfessorCourse3 > BiRobot4 > USSTest0

実行結果：超音波距離センサーに手をかざし、近づけたり遠ざけたりするとシリアルモニターにセンサーとの距離の値がcmで表示される。

では、プログラムを見てみましょう。
超音波距離センサーは、トリガーに合図のパルスを送るときには OUTPUT （出力）、エコーから合図のパルスを受け取るときには INPUT （入力）として使う必要があります。
まずはトリガーにパルスを発信する部分です。

プログラム「USSTest0」より抜粋

```
pinMode(US2, OUTPUT);                    // US2をTrigger用の出力に設定

digitalWrite(US2, LOW);                  // Trigger Lowレベルにする
delayMicroseconds(10);                   // 10マイクロ秒待つ

digitalWrite(US2, HIGH);                 // Trigger Highにする
delayMicroseconds(10);                   // 10マイクロ秒待つ

digitalWrite(US2, LOW);                  // Trigger Lowレベルにする
```

続いて、エコーから信号を受け取る部分です。

プログラム「USSTest0」より抜粋

```
pinMode(US2, INPUT);                     // US2をEcho用の入力に設定

uSec = pulseIn(US2, HIGH, 58 * 200);     // US2がHighになっている時間を計測する
```

ピンを INPUT モードに変更していますね。pulseIn(US2, HIGH, 58 * 200); という関数で、エコーから送られてきたパルスの幅（信号が HIGH だった時間）をチェックしています。

なお、58 * 200 の部分は、パルス幅の検知を行う最大の時間を表しています。この時間（58×200＝11600マイクロ秒）を過ぎると、まだパルスが送られてきていたとしても検知を強制終了し、それ以上は計測されません（タイムアウトと言います）。

あとは、チェックしたパルス幅、つまり時間（マイクロ秒）を距離（cm）に変換し、シリアルモニターに表示するだけです。

プログラム「USSTest0」より抜粋

```
Serial.print("USS: ");
if (uSec == 0){
    Serial.print(">200");
}
else {
    Serial.print(uSec / 58);
}
Serial.println("[cm]");
```

先ほどの解説通り、はかった時間÷58で距離を求めています（黄色の部分）。

では、もう1度プログラム「USSTest0」を実行してシリアルモニターで状態を確認してみましょう。

プログラムの書き込み

RoboticsProfessorCourse3 > BiRobot4 > USSTest0

さきほどは、超音波距離センサーの近くで手をかざしてみましたが、今度は遠くの壁などに向けてみてください。結果、近くの障害物に対してはある程度の精度で距離が表示されると思いますが、遠くのものは検出せず正しい値が出てこないと思います。

製品の仕様書通りにプログラムを作成したのに、得られる結果が正しくないことは、複数の機能を組み合わせていくと発生する問題です。こうした問題をいかに解決していくかがロボット開発では非常に大切です。原因は、US2 端子はマトリクスLEDシールド経由で接続しているため、マトリクスLEDの仕組みが関係しています。

マトリクスLEDは8×8（64個）のLEDを点灯か消灯させて、さまざまな表示をさせますが、実は複数のLEDを同時に点灯させることができないため実際は高速で点灯、消灯させています。
つまり、電気回路のON・OFFを高速で切り替えているわけですが、回路のON・OFFを切り替えると、電圧が変化しきるまでのごく短い時間だけ、周りの電気の流れを邪魔（じゃま）するような波（ノイズ）が発生します。これをスイッチングノイズとよびます。

今回のプログラムはマトリクスLEDを点灯させていませんが、マトリクスLEDは接続されているだけでも内部的にはランダムにLEDを点灯させようとしているため、スイッチングノイズがくり返し発生していることになります。これが、センサーのエコーから受け取るはずのパルス信号を邪魔（じゃま）してしまっているというわけです。

この問題については、この後解決していきましょう。

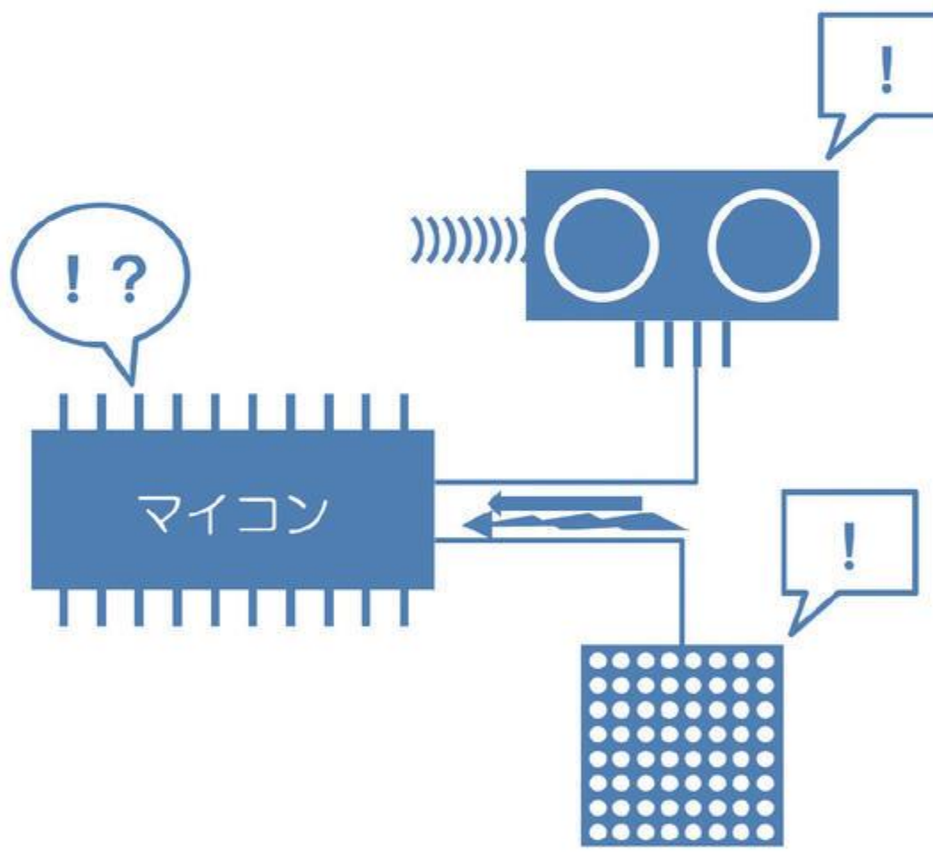


図1-2　マトリクスLEDのスイッチングノイズに邪魔されるエコー信号（イメージ）

ステップアップ

この問題を解決するとして、どんな対策が考えられるかな？
「自分に実現できるか」は抜きにして、いろいろな案を考えてみよう！

1.2. 超音波距離センサーの問題をプログラムで解決しよう

ここからが今回の本題となる「トラブルシューティング」です。
冒頭でもお話しましたが、１つの電子部品を単独で動かすときは何も問題なくても、複数の電子部品を組み合わせるときはさまざまな制約や問題が発生することがあります。
こんなときは、問題の根源を探り解決する「トラブルシューティング」がとても大切です。
トラブルシューティングをするには、あてずっぽうや推理ではなく、さきほどまでに学習したような電子部品の仕様をしっかりと知り、論理的に分析することが必要です。では実践していきましょう。

そもそもこの問題の原因は「マトリクスLEDを連続してON・OFFすることで発生するスイッチングノイズ」でしたね。ということは、スイッチングノイズを抑えるような対策が必要になります。具体的には、次の2つのような対策が代表的です。

POINT

対策①：ノイズがエコー信号を邪魔しないように回路そのものを直す。
対策②：マトリクスLEDの連続ON・OFFをプログラム側で抑制する。

対策①は非常にシンプルな考え方で、物理的に問題をシャットアウトしよう、というものです。問題部分に直接手を加えるので効果的ですが、パーツや基板を分解し、素子や配線そのものを変更することになるため、手間もお金もかかります。

一方、対策②はプログラムをかきかえるだけで済みます。もちろんプログラムだけでは修正できない問題もありますが、もしプログラム修正だけで済ませられれば手間もそれほどかからず、お金にいたってはまったくかかりません（電気代や、会社の開発現場であればプログラマーの人件費などはかかりますが…）。

このように、問題をプログラム側から修正できるような状況であれば、それで済ませるというのは一般的な電子機器製品の開発現場などでもよくあることです。

今回の問題も、プログラムの修正で解決してみましょう。

チャレンジ課題

プログラム「USSTest0」をかきかえ、スイッチングノイズの発生を抑え、より距離の測定精度が高まるようにしてみよう！

ヒント

マトリクス LED は、消灯の命令を受けない限りはランダムに LED を点灯させようとするよ。

いかがでしょうか。
はじめにマトリクスLEDに消灯命令を出しておけば、その後はON・OFFをくり返されることもないためスイッチングノイズの発生を抑えられます。

具体的には以下のようなプログラムになります。

プログラムの書き込み

RoboticsProfessorCourse3 > BiRobot4 > USSTest1

プログラム「USSTest1」より抜粋（ばっすい）

```
#include <RPlib.h>          //ロボプロシールドを使うためのライブラリ(設定ファイル)
#include <Matrix.h>         //マトリクスLEDを使うためのライブラリ(設定ファイル)
#include <Sprite.h>

Matrix myMatrix = Matrix(11, 13, 1);



void setup(){
    Serial.begin(9600);   //シリアル通信を使うときのオマジナイ
    myMatrix.clear();
}
```

マトリクスLEDに命令を出す際には、マトリクスLED用のライブラリファイル「Matrix.h」と「Sprite.h」が必要になります。

あとは、clear()命令を出すことでLEDを消灯させておくだけです。今回はloop()内でマトリクスLEDに命令を出すことが無いので、消灯命令はsetup()内で1回出しておくだけで済みます。

1.3. 超音波距離センサーをライブラリで動かそう

さて、先ほどマトリクスLEDに命令を出すため、マトリクスLED用のライブラリファイルを呼び出していましたね。
しかし、「USSTest1」はそもそも超音波距離センサーをメインで使用するプログラムです。であれば、超音波距離センサー用のライブラリも活用した方が、よりプログラムをシンプルにまとめられると思いませんか。

超音波距離センサー用のライブラリファイルは「NewPing.h」から呼び出します。実際にこのライブラリを呼び出したプログラムを開いてみましょう。

プログラムの書き込み

RoboticsProfessorCourse3 > BiRobot4 > USSTest2

プログラム「USSTest2」より抜粋

```
#include <RPlib.h>      //ロボプロシールドを使うためのライブラリ(設定ファイル)
#include <NewPing.h>    //超音波距離センサーを使うためのライブラリ(設定ファイル)
#include <Matrix.h>     //マトリクスLEDを使うためのライブラリ(設定ファイル)
#include <Sprite.h>

NewPing mySonar(US2, US2, 200);
Matrix myMatrix = Matrix(11, 13, 1);
```

まずはライブラリファイルを取り込み、mySonar()関数で使う超音波距離センサーの宣言をします。ちなみに、()内の引数はトリガーのピン、エコーのピン、最大測定距離（cm）が入ります。今回は往復200cm（片道100cm）までの測定ができるわけですね。

超音波が戻ってくるまでの時間測定も、ライブラリ内の関数を使えば簡単に命令できます。

プログラム「USSTest2」より抜粋

```
void loop(){
    unsigned int uSec;

    uSec = mySonar.ping();          // 超音波距離センサーを使って障害物に超音波
                                       が当たって
                                    // 戻ってくるまでの時間を計測する。
```

「トリガーにパルスを送って超音波を発信させ、戻ってきた超音波を検知したらかかった時間を計測する」という一連の動作が、ping();という関数1つだけで済みます。このping();という関数はライブラリファイル「NewPing」に搭載されているので、まずライブラリの呼び出しが必要だったわけです。

関数 ping(); は時間を計測するところまでですが、距離を算出するところまで自動で行ってくれる関数も、同じライブラリに搭載されています。

ライブラリファイル「NewPing.cpp」より抜粋

```
unsigned long NewPing::ping_cm(unsigned int max_cm_distance) {
      unsigned long echoTime = NewPing::ping(max_cm_distance);
#if ROUNDING_ENABLED == false
      return (echoTime / US_ROUNDTRIP_CM);
#else
      return NewPingConvert(echoTime, US_ROUNDTRIP_CM);
#endif
}
```

距離を算出する関数 ping_cm() の動作を記しています。まず関数 ping(); で時間を計測したあと、変数 echoTime に計測した時間を入れ、それを US_ROUNDTRIP_CM という数値で割っているだけです。

US_ROUNDTRIP_CM は #define により「57」と定義されています。先ほど58で割っていたのとほぼ同じですね（音速は気圧や気温で変化するので、割る数も必ず58というわけではありません）。

普段のプログラムとはちょっとちがったかき方ですが、「なんとなくこんな意味かな？」という程度でもよいので、読んでおきましょう。

ステップアップ

プログラム「USSTest2」を、関数 ping_cm() を活用してさらにシンプルにまとめてみよう！

講　解答例は以下のプログラムです。
RoboticsProfessorCourse > BiRobot4 > USSTest3

ここまで、「USSTest0」のノイズ問題を「USSTest1」で解決し、さらに「Newpingライブラリ」を利用することでプログラム「USSTest2」のように見やすくすることができました。

最後の「ステップアップ」ではさらにプログラムを整理し、「USSTest1」と比較するとプログラム全体で10行程度減らすことができました。

2.　複数のデバイスを統合させよう（目安40分）

さて、これで二足歩行ロボットに超音波距離センサーの動作を追加できました。マトリクスLEDとの同時使用で不具合が出ましたが、これもプログラムで修正できましたね。
ロボットというのはこのように、たくさんの機能、部品を同時に動作させることが多く、そのため複数の機能が干渉しあって不具合が出るというのもよくあることです。
今回は残った時間で以下のように機能を追加していき、発生する不具合を一つ一つ解消していきましょう。

POINT

① マトリクスLEDでハートのアニメーションが再生できるようにする。
② アニメーションにあわせてスピーカーから効果音が出るようにする。
③ コントローラーによる操作ができるようにする。

2.0.　超音波距離センサーとマトリクスLEDを同時に動かそう

まず超音波距離センサーとマトリクスLEDを同時に使えるようにしましょう。
さきほどのプログラムでは、ノイズ問題の解消のために超音波距離センサーで障害物までの距離を計測している間は、マトリクスLEDをすべて消灯にしなければいけませんでした。このままでは、マトリクスLEDの機能は封印されたままなので、再びプログラムに手を加えていきましょう。
とりあえず、前回のマトリクスLEDにハートを表示させるプログラム「DisplayTest0」と、超音波距離センサーのプログラム「USSTest3」を合体させたプログラムを見てみましょう。
次のプログラムを開いてください。

プログラムの書き込み

RoboticsProfessorCourse3 > BiRobot4 > USSDispTest0

プログラム「USSDispTest0」より抜粋（ばっすい）

```
void loop(){
    unsigned long distance;

    myMatrix.clear();                          // マトリクスLEDの表示を消す

    distance = mySonar.ping_cm(200);           // 超音波距離センサーを使って
                                               障害物までの距離を計測する。

    switch (spriteIndex){
    case 0:
        myMatrix.write(0, 0, spdata_0);        // 大ハート
        spriteIndex++;
        break;

    case 1:
        myMatrix.write(0, 0, spdata_1);        // 中ハート
        spriteIndex++;
        break;

    case 2:
        myMatrix.write(0, 0, spdata_2);        // 小ハート
        spriteIndex++;
        break;

    case 3:
        myMatrix.write(0, 0, spdata_1);        // 中ハート
        spriteIndex = 0;
        break;
    }

    Serial.print("USS: ");
    if (distance == 0){
        Serial.print(">200");
    }
    else {
        Serial.print(distance);
    }
    Serial.println("[cm]");

    delay(250);
}
```

プログラム自体は、本当に「USSTest3」にただ「DisplayTest0」のアニメーションの設定（赤枠の部分など）を追加しただけです。どちらのプログラムも単体ではきちんと動いていたのに、合体させると動作が不安定になりますね。
これは、複数の機能が、マイコンの同じピンを使おうとしているために起こる不具合です。

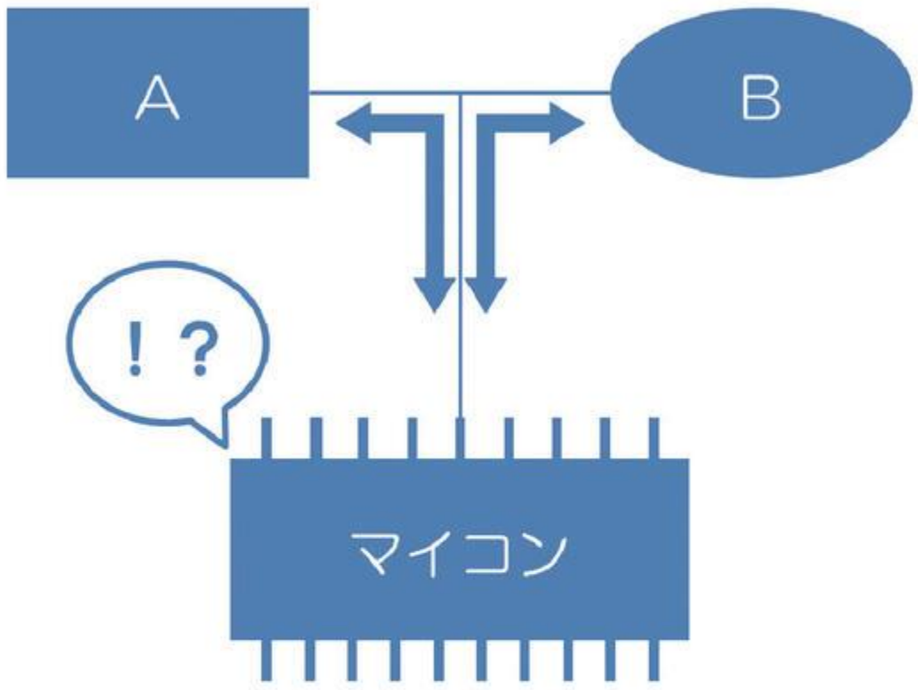


図2-0　ピンの競合

マイコンボードは基本的に「●番ピンから信号を出す」「▲番ピンに届いた信号を読み取る」といった処理（しょり）を行うので、同じピンの取り合いが発生すると「Aに送りたかった信号がBに届いてしまった！」「Bからの信号を待ってるのに、Aから届いたためちゃくちゃな信号を検知してしまった！」「Aに信号を送るためのピンなのに、なぜかBから信号が送られてきた！」といった予期しない状態になってしまいます。
各種シールドのコネクタは、それぞれマイコンボードの決まったピンにつながっています。それぞれの使用ピンを読み取り、どの機能でピンの取り合いが発生しているのか確認してみましょう。

やってみよう！

巻末の「コネクタ分配表」および「ピンの競合表」をチェックして、今回の不具合がどの機能の、何番ピンの取り合いから発生しているのか確認してみよう！

競合を起こしている機能：　シリアルモニター　と　マトリクスLED

取り合っているピン：　P1

わかりましたか？
「ピンの競合表」のほうが見やすいと思いますが、シリアル通信（シリアルモニターを表示するための機能）とマトリクスLEDが、ともにP1を使用するため取り合いが起こっていますね。

取り合いを防ぐための一番手っ取り早い対策は「どちらかの機能をオフにする」ことです。マトリクスLEDは今後アニメーションの表示に必要なので、今回はシリアル通信をオフにしましょう。

```
//Serial.begin(9600);                    // シリアル通信を9600bpsで初期化する
```

実行結果：マトリクスLEDに、♡のアニメーションが正常に表示される。
上のようにスラッシュを2本「//」入れるとシリアル通信の初期化部分を無効にすることができます。結果、マトリクスLEDには♡のアニメーションが正常に表示されました。

ただこれでは、シリアル通信が使えなくなり、超音波距離センサーが障害物までの距離を計測した結果がわかりません。次では、超音波距離センサーの計測結果をマトリクスLEDに表示をさせる方法を考えましょう。

講　巻末のコネクタ分配表を活用すると、授業で使用するマイコンに対し、どの電子部品とどの電子部品が同時に使用できないか（競合するか）を確認することができます。

2.1. 超音波距離センサーの計測結果をマトリクスLEDに表示させる

ここでは、マトリクスLEDの♡のアニメーションの表示はそのままに、空いた場所に超音波距離センサーの計測結果を表示させるバーグラフ表示（インジケーター）をさせてみましょう。

♡のアニメーション表示は、マトリクスLEDの一番下の列を空けてあることを利用して、その部分に超音波距離センサーで計測した障害物までの距離を**図2-1**のようにバーグラフ表示をさせます。

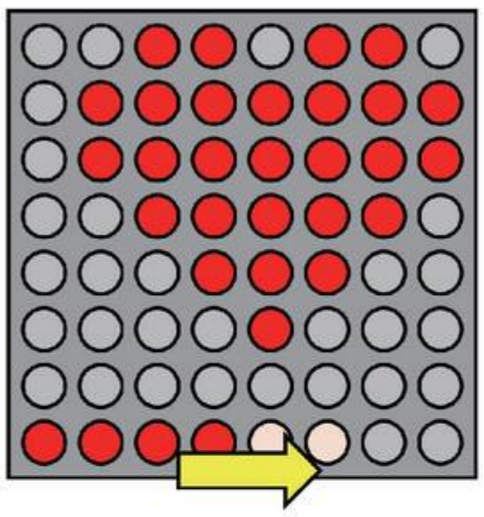


図 2-1　障害物のバーグラフの表示

では、バーグラフ表示の処理を加えた次のプログラムを実行しましょう。

プログラムの書き込み

RoboticsProfessorCourse3 > BiRobot4 > USSDispTest1

実行結果：マトリクスLEDにバーグラフが表示されて、超音波距離センサーで計測した距離に応じて変化する。シリアルモニターには何も表示されなくなる。

なお、バーグラフ表示のプログラムを♡マーク表示の処理より前にかいてしまうと、データが上がきされてバーグラフ表示が消えてしまうので、プログラムの順番を間違えないように注意するのがポイントです。
では、次のページでプログラムの中身を見てみましょう。

プログラム「USSDispTest1」より抜粋（ばっすい）

```
if (distance == 0){
    num = 8;
}
else {
    num = ( distance * 8 ) / 100;
}

for (i = 0; i < 8; i++){
    myMatrix.write(i, 0, num <= i);
}
```

おなじみfor文が登場しています。(0,0) から(7,0)まで、横一列のLEDに点灯命令を出していますね。
ただし、障害物（しょうがいぶつ）までの距離（きょり）の変数 distance の値に応じて、0~8まで変化する変数 num をつくりました。 num の値に応じて、点灯させるLEDと消灯させるLEDが変化します。 num <= i の部分は、この条件式が成り立つときは点灯（HIGH)、成り立たないときは消灯（LOW）の命令をLEDに送ることを意味しています。

2.2. スピーカーの機能を追加する

ここでは、さらにスピーカーの機能を追加していきましょう。
さきほどと同じようにプログラム「USSDispTest1」に単純にスピーカーの機能を使うためのプログラムをかき加えていきます。以下のプログラムを開いて見てみましょう。

プログラムの書き込み

RoboticsProfessorCourse3 > BiRobot4 > USSDispToneTest0

まずは、次のようにスピーカーを使用するための「Toneライブラリ」と、スピーカーを接続するポートをロボプロシールドの D3端子 にする初期設定をかき加えています。

プログラム「USSDispToneTest0」より抜粋（ばっすい）

```
#include "NewPingKAI.h"
#include <RPlib.h>          //ロボプロシールドを使うためのライブラリ(設定ファイル)
#include <Matrix.h>         //マトリクスLEDを使うためのライブラリ(設定ファイル)
#include <Sprite.h>
#include "ToneKAI.h"        //スピーカーを鳴らすためのライブラリ(設定ファイル)

#include  "SpriteData.h"

NewPingKAI mySonar(US2, US2, 200);
Matrix myMatrix = Matrix(11, 13, 1);
ToneKAI myTone;
int spriteIndex;



void setup(){
    spriteIndex = 0;

    myMatrix.clear();
    myMatrix.setBrightness(1);      // マトリクスLEDの明るさを設定(0－8)

    myTone.begin(D3);               // スピーカーがD3に接続されていることを設定
}
```

include "ToneKAI.h" のように KAI が入っているヘッダファイルは、ヘッダーファイル「Tone」に独自の設定を加えるため、タブ化したことを意味します。

次に、マトリクスLEDの♡の表示にスピーカーの音を合わせるプログラムをかき加えます。
（第３回のプログラム「DisplayToneTest0」と全く同じ内容です。）

プログラム「USSDispToneTest0」より抜粋

```
    case 0:
      myMatrix.write(0, 0, spdata_0);    // 大きなハート
      myTone.play(NOTE_C7, 100);
      spriteIndex++;
      break;
    case 1:
      myMatrix.write(0, 0, spdata_1);    // 中くらいのハート
      myTone.play(NOTE_C6, 100);
      spriteIndex++;
      break;
    case 2:
      myMatrix.write(0, 0, spdata_2);    // 小さいハート
      myTone.play(NOTE_C5, 100);
      spriteIndex++;
      break;
    case 3:
      myMatrix.write(0, 0, spdata_1);    // 中くらいのハート
      myTone.play(NOTE_C6, 100);
      spriteIndex = 0;
      break;
  }
```

では、中身を見たところでプログラム「USSDispToneTest0」を実行してみましょう。
結果、プログラム画面のメッセージエリアに「コンパイル時にエラーが発生しました。」というメッセージが表示され、書き込みエラーとなります。
さきほど言ったように、「DisplayToneTest0」と全く同じようにプログラムしたのにエラーがでるのはなぜでしょうか？

先ほどは使用ピンの取り合いからエラーが起こりましたが、今回は「タイマー」という機能の取り合いが原因です。
内容が専門的すぎるので簡単にまとめますが、Arduinoでは時間に関係した命令を処理するとき「タイマー」という機能を使っています。使えるタイマーの枠が1つだと、複数の機能で別々に時間を管理するのが難しいので、プログラム時に使えるタイマーの枠は「Timer1」と「Timer2」の2つ用意されています。（「Timer0」という枠もありますが、これはArduino本体の時間管理専用の枠になっているためプログラムでは使えません）
プログラム側の問題で「どちらも同じタイマーを使おうとする」と、ピンの取り合いと同じように不具合が出ることがあるのです。

今回のケースでは、スピーカー用ライブラリ「ToneKAI」と超音波距離センサー用ライブラリ「NewPing」が、どちらもTimer2を使おうとした結果取り合いになってしまっています。よって、どちらかのライブラリを少しかきかえてあげましょう。
今回はライブラリのかき換えを行えるように、画面上部のタブからライブラリファイルを表示できるようにしてあります。「NewPingKAI.h」のタブを探して表示し、以下のようにかきかえてみましょう。

プライブラリファイル「NewPingKAI.h」より抜粋

```
#define TIMER_ENABLED true
```

▼

```
#define TIMER_ENABLED false
```

プログラムの書き込み

RoboticsProfessorCourse3 > BiRobot4 > USSDispToneTest0_lib

実行結果：エラーが解消されて、全ての機能が正常に機能する。

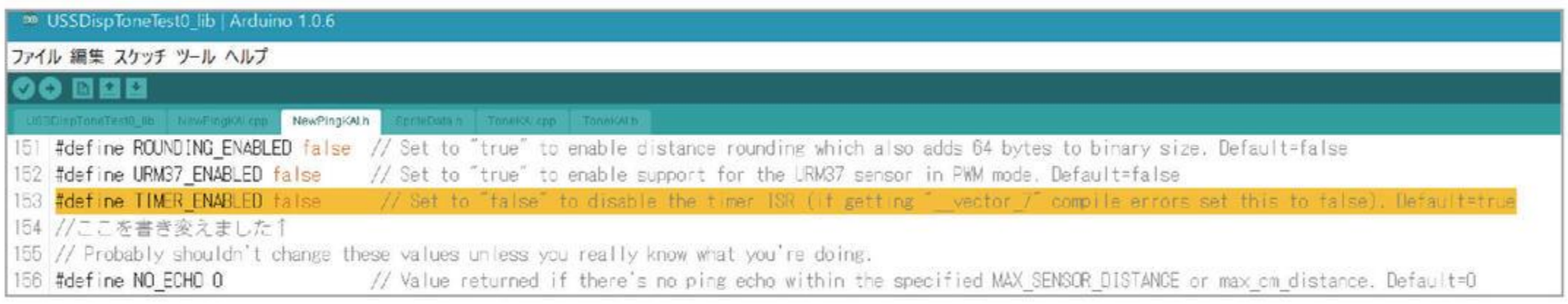


図2-2 「USSDispToneTest0_lib」の「NewPingKAI.h」

2.3.　コントローラーの機能を追加する

最後に、コントローラーの機能を追加しましょう。
ここまでに何かひとつ機能を追加するたびに次々と問題が起こって、気が滅入(めい)ってしまったかもしれませんが、ここでは問題が発生しませんので安心してください。
次のプログラムを実行しましょう。

プログラムの書き込み

RoboticsProfessorCourse3 > BiRobot4 > USSDispToneRemoteTest0_lib

実行結果：コントローラーの十字ボタンを上下左右に入れると、音階が変化する。
では、「USSDispToneTest0_lib」から追加されたプログラムを確認していきましょう。

まず、コントローラーを使用するために必要なライブラリと初期化を行います。
次の黄色のラインが追加された部分です。

プログラム「USSDispToneRemoteTest0_lib」より抜粋(ばっすい)

```
#include <RPlib.h>          // ロボプロシールドを使うためのライブラリ（設定ファイル）
#include "NewPingKAI.h"     // 超音波距離センサーを使うためのライブラリ（設定ファイル）
#include <Matrix.h>         // マトリクスLEDを使うためのライブラリ（設定ファイル）
#include <Sprite.h>
#include <Tone.h>           // スピーカーを使うためのライブラリ（設定ファイル）
#include <PS2X_lib.h>       // コントローラーを使うためライブラリ（設定ファイル）

#include  "SpriteData.h"

NewPingKAI mySonar(US2, US2, 200);      // 超音波距離センサーの初期化
Matrix  myMatrix = Matrix(11, 13, 1);   // マトリクスLEDの初期化
Tone    myTone;                         // スピーカーの初期化
PS2X    myPS2X;                         // コントローラーの初期化

(中略)

  myTone.begin(D3);                     // スピーカーがD3に接続されていることを
                                           設定

  myPS2X.config_gamepad(13,11,10,12);   // コントローラーの初期化
}
```

ここで注目してほしいのは、次の部分です。
マトリクスLEDを使用するときに使われているピンとコントローラーを使用するときに使われているピンが11番と13番の2箇所で競合していると思われますが、問題は起きません。この11番ピンと13番ピンは、マイコンからマトリクスLEDとコントローラーに命令するときの信号になっていますが、両方の部品に同時に命令が渡ることはなく、競合問題を避けられるようになっています。そうした点でエラーが生じることがなかったのです。

```
Matrix  myMatrix = Matrix(11, 13, 1);      // マトリクスLEDの初期化
（中略）
myPS2X.config_gamepad(13,11,10,12);        // コントローラーの初期化
```

では実際にプログラムを実行して、コントローラーで行いたい処理(しょり)を実行してみましょう。
十字ボタンを上下左右に入れて、音階が変化したでしょうか。

３．まとめ（目安５分）

今回は、二足ロ歩行ボットの各機能のプログラムを段階的に統合(とうごう)していく手順を解説しました。
単体ずつでは問題なく動いていたプログラムも、組み合わせて使おうとすると同じ機能を取り合ったり、電気信号が干渉(かんしょう)しあったりして思わぬ不具合が起きてしまいました。これは、ロボットに限らずものづくりをしていく中で非常によくある問題です。このようなときには問題個所を切り出して解消法を考える「トラブルシューティング」が重要であることを学べましたね。

ロボットのような専門的な機械のトラブルシューティングは経験がものを言う場面も多いですが、「どんな問題なんだろう？」「なぜ起こっているんだろう？」「どうすれば解消するかな？」といったことを考える力が必要なことも確かです。こういった「問題解決」の力はロボットのことに限らず、とにかく「深く考える」ことで大きく伸びますので、皆さんも普段ロボプロや学校の授業、あるいは日常生活から学べることをきちんと身につけていってほしいと思います。

次回は、さらに手足を動かすためのサーボモーターの機能を統合し、二足「歩行」ロボの完成にさらに近づきましょう！

講

○以下の理解度を確認します。
・超音波距離センサーの仕様書を確認する
・仕様書を読んで、仕様をプログラムに落とし込む
・電子部品の統合による様々なトラブルシューティング方法を知る
○次回のテーマは「二足歩行ロボットのプログラミング②」であることを告知します。

表1）ピンの排他（はいた）表：マイコンボードピンとロボプロシールドのコネクタ分配表

			ロボプロシールド																						マトリクスLEDシールド					姿勢検出シールド					液晶ディスプレイシールド	
	マイコン使用ピン		P10	P9	P2	P3	A0	A1	A2	A3	A4	A5	P2/P3/P4	A0 A1/P5	P6/P7/P8	P9/P12/P13	P0/P1	A4/A5	P6	P7	P8	P5	P2	P3	P10/11/12/13	P0	A3	A2	P1/P11/P13	P1/P11/P13	A4/A5	P0/P9	A2/A3	P10/P12	A4/A5	P1/P9/P11/P13/A2	P11/P12/P13/A3
	マイコンピンソケット	ロボプロ表記	D0	D1	D2	D3	A0	A1	A2	A3	A4	A5	MC0	MC1	MC2	MC3	SCI	IIC	S0	S1	S2	S3	S4	S5	コントローラ	US0	US1	US2	7セグメントLED	マトリクスLED		ENC0	ENC1	ENC2	ENC3	TFT CON	TFT CON SD
ロボプロシールド	P10	D0	■													■									■												
	P9	D1		■												■																■			■			
	P2	D2			■								■																								
	P3	D3				■							■											■														
	A0	A0					■							■																								
	A1	A1						■						■																								
	A2	A2							■																			■					■				■	
	A3	A3								■																	■						■				■
	A4	A4									■							■													■						
	A5	A5										■						■													■						
	P2/P3/P4	MC0			■	■							■										■	■													
	A0/A1/P5	MC1					■	■						■								■																
	P6/P7/P8	MC2													■				■	■	■																
	P9/P12/P13	MC3		■												■									■				■	■					■	■	■
	P0/P1	SCI(シリアル通信)															■									■			■	■						■	
	A4/A5	IIC									■	■						■													■						
	P6	S0													■				■																		
	P7	S1													■					■																	
	P8	S2													■						■																
	P5	S3												■								■															
	P2	S4											■										■														
	P3	S5				■							■											■													
	P10/P11/P12/P13	コントローラ-SPI	■													■									■				☆	☆					■	■	■
マトリクスLEDシールド	P0	US0															■									■						■					
	A3	US1								■																	■										
	A2	US2							■																			■									
	P1/P11/P13	7セグメントLED-SPI														■	■								☆				■	■						■	■
	P1/P11/P13	マトリクスLED-SPI														■	■								☆				■	■						■	■
姿勢検出シールド	A4/A5										■	■						■													■						
	P0/P9	ENC0		■																							■					■					
	A2/A3	ENC1							■	■																							■				
	P10/P12	ENC2	■													■									■									■			
	A4/A5	ENC3	使用しません																																		
液晶ディスプレイシールド	P1/P9/P11/P13/A2	TFTCON		■					■							■	■								■				■	■						■	
	P11/P12/P13/A3	TFTCON-SD								■						■									■				■	■							■

※姿勢検出シールドとIIC(カラーセンサ)は絶対に同時接続しないようにしてください

☆例外的に同時に使えます

表2）ピンの競合表

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	A0	A1	A2	A3	A4	A5
US0	■																			
US1																		■		
US2																	■			
IIC																			■	■
SCI	■	■																		
A0															■					
A1																■				
A2																	■			
A3																		■		
A4																			■	
A5																				■
D0											■									
D1										■										
D2			■																	
D3				■																
S0							■													
S1								■												
S2									■											
S3						■														
S4			■																	
S5				■																
MC0			■	■	■															
MC1						■									■	■				
MC2							■	■	■											
MC3										■			■	■						
CONTROL											■	■	■	■						
MATRIX LED		■										■		■						
7 SEGLED		■										■		■						