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

倒立振子ロボット②

第4回

姿勢検出シールドを使って ロボットを動かす

講師用

目　　次

○　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。

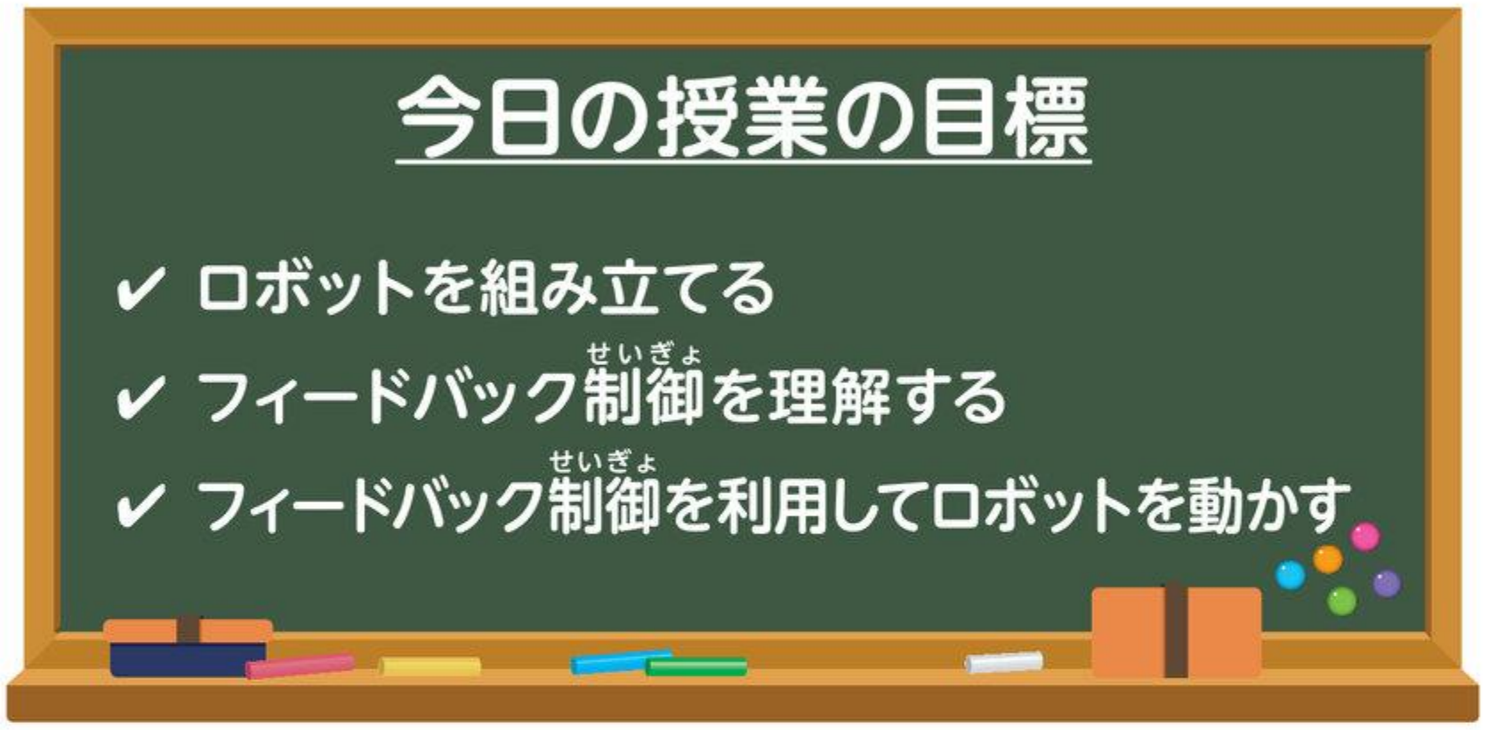
○　今回の目標をパネルで用意するか、黒板に予め書いておきます。

（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）

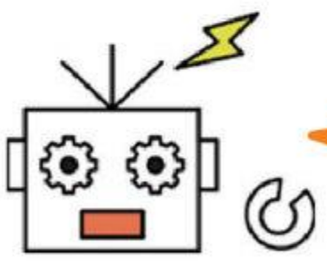
目安時間は授業時間 120 分のうち、休憩 10 分程度取ることを想定しています。
生徒の進捗状況により、休憩時間などを調整して授業を行ってください。

0. 姿勢検出シールドを使ってロボットを動かす（目安５分）

0.0. 「姿勢検出シールドを使ってロボットを動かす」でやること



これまで「加速度」「角速度」といった姿勢検出シールドのセンサーの使い方を学んできましたね。
今回は、いよいよ姿勢検出シールドをロボットに組み込んで動かします！　動かすロボットはおなじみのオムニホイールロボットです。
そして「フィードバック」というロボットには欠かせない技術を学んで「倒立振子ロボット」へのステップとします。「フィードバック」とは、出力に関する目標の値と実際の値とを比べて、自動的に２つの値が一致するように制御することです。実は私たちが日常的に使っているエアコンや冷蔵庫などでも広く活用されています。
センサーが取得した情報を使って動くロボットの特長を学びながら、フィードバック技術を使ったロボットを体験しましょう。



目標と現状のギャップをとらえて、そこをどう埋めていくかが課題解決のカギなのダ！
ロボットだって、人間だって同じだネ！！

0.1. 必要なもの

前回使った姿勢検出(しせいけんしゅつ)ユニットの他に、以下のパーツを準備しておきましょう。

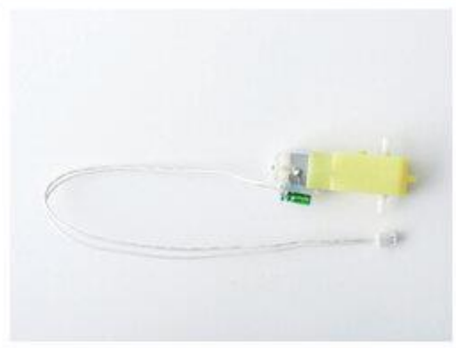
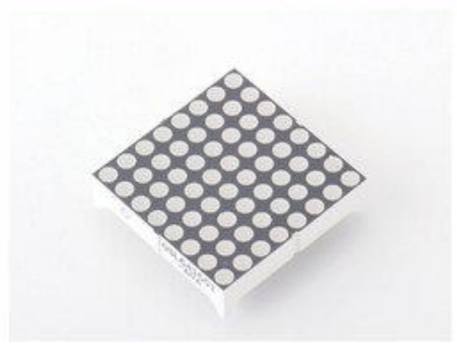
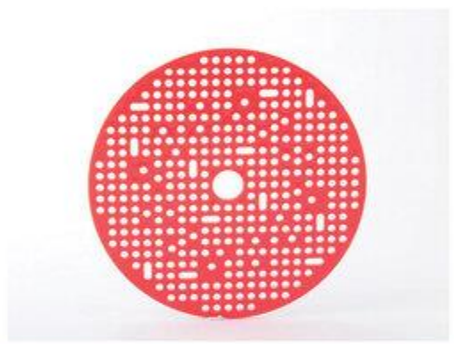
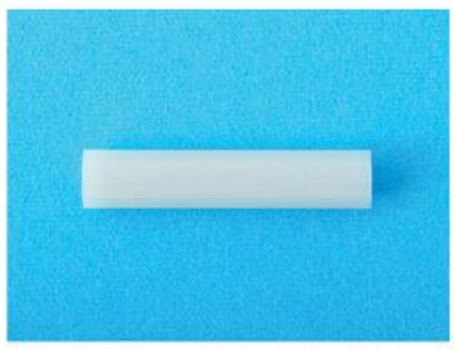
ラジオペンチ 1	ドライバー 1	USB ケーブル 1	ギアドモーター 3
			
タッチセンサー 1	**モーター L 字ステイ 3**	**マトリクス LED 1**	**オムニホイール 3**
			
M2.6L20 タッピングネジ (A) 3	**赤円形ボード 1**	**白円形ボード 1**	**M3 ナット 19**
			
M3L5 ネジ 7	**M3L8 ネジ 11**	**M3L25 ネジ 6**	**8mm 角スペーサー 4**
			
30mm 角スペーサー 3	**7 セグメント LED 1**		
			

図 0-0　必要なもの

1．ロボットの組み立てと動作確認（目安 40 分）

1.0.　組み立てと配線

図 1-0 が今回つくるロボットの完成イメージです。前回使った姿勢検出ユニットを、オムニホイールロボットに合体させたら完成となります。これから手順に沿って組み立てていきましょう。

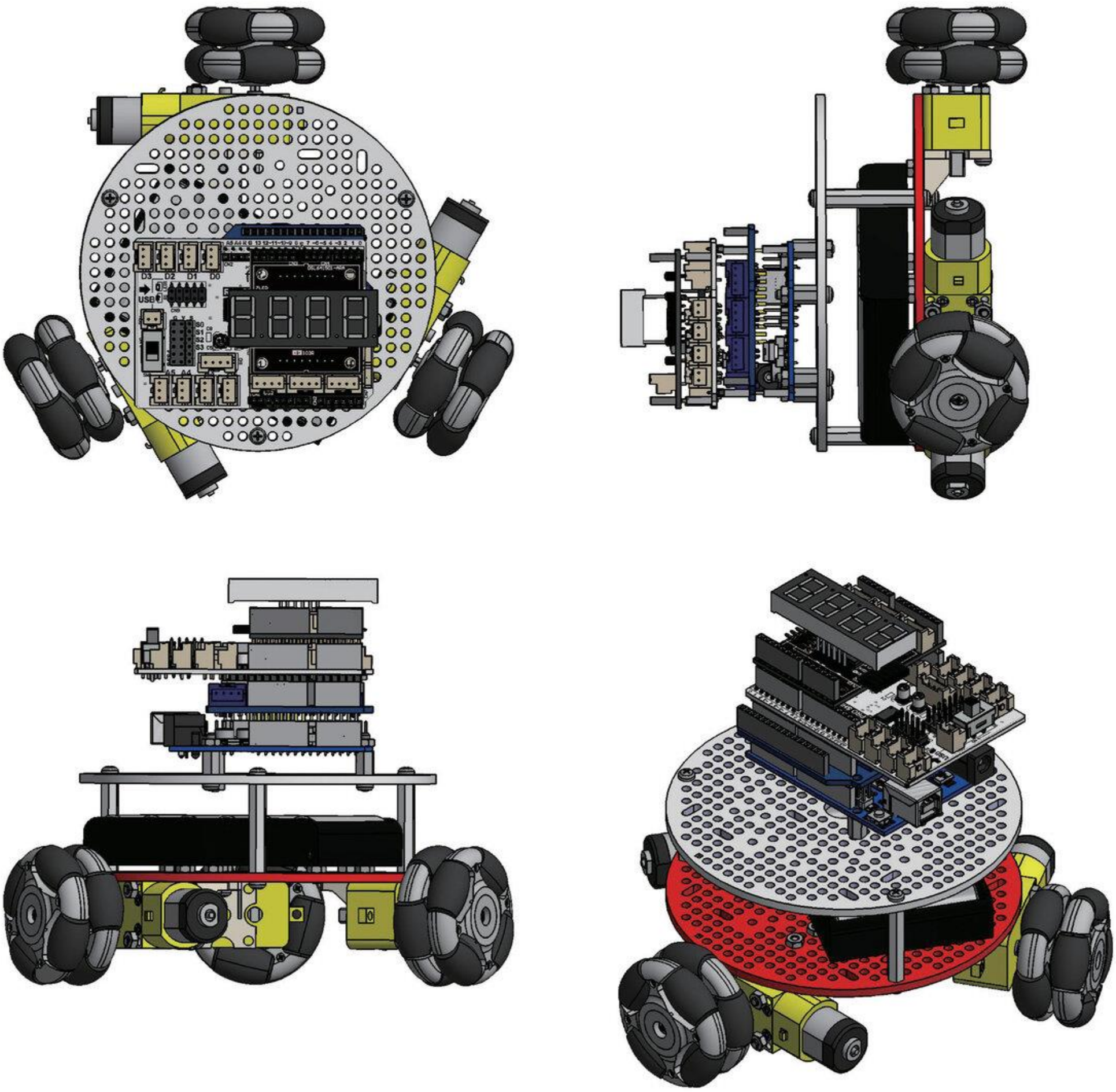


図 1-0　ロボットの完成イメージ

＜組み立て手順①＞

まずはオムニホイールロボットを組み立てましょう。オムニホイールロボットの第１回のテキストも参照しましょう。前回使った姿勢検出ユニットからマイコンボードのみ外してから始めましょう。

なお、今回は無線受信モジュールは取り付けません。

図 1-1　オムニホイールロボットのモーターなどの取り付け位置

＜組み立て手順②＞

各シールドなどをマイコンボードに接続します。下から、姿勢検出(しせいけんしゅつ)シールド、ロボプロシールド、マトリクス LED シールド、7 セグメント LED の順になります。接続を終えたら、各ピンがしっかりとソケットに入っているかもチェックしましょう。

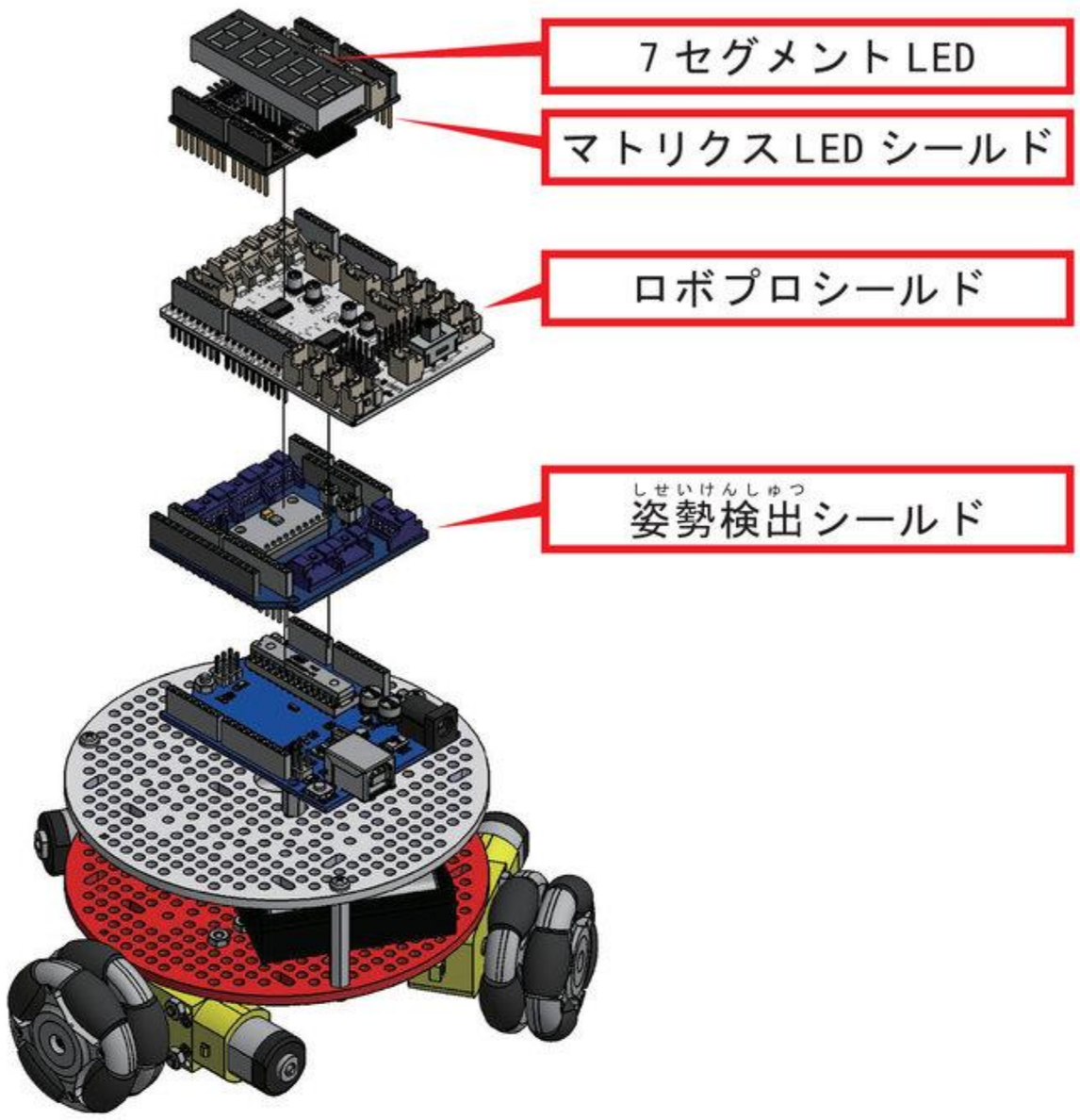


図 1-2　各シールドなどの接続

＜組み立て手順③＞

ギアドモーターとタッチセンサーのコネクターをロボプロシールドに接続します。ギアドモーターは**図 1-3** を参考に接続しましょう。また、タッチセンサーは D1 に接続しましょう。なお、タッチセンサーは、プログラム書き込み後にロボットが急に動き出すのを防ぐ目的で、スタートボタンとして使います。そのためネジなどでどこかに固定する必要はありません。

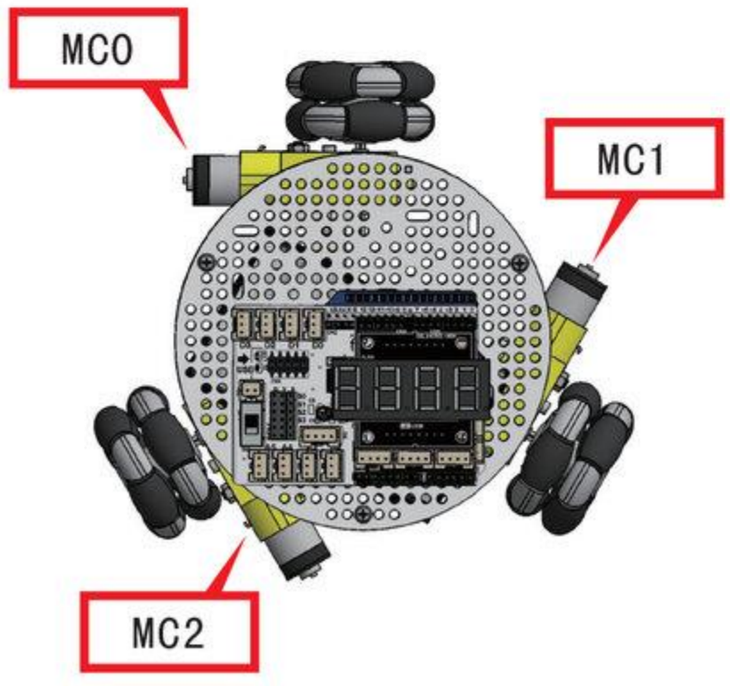


図 1-3　ギアドモーターの接続

1.1.　動作確認

１）移動方向の確認

ロボットの移動方向を確認します。
以下のプログラムを実行しましょう。

プログラムの書き込み
RoboticsProfessorCourse2 ＞ Inverted 4 ＞ OmniDriveTest

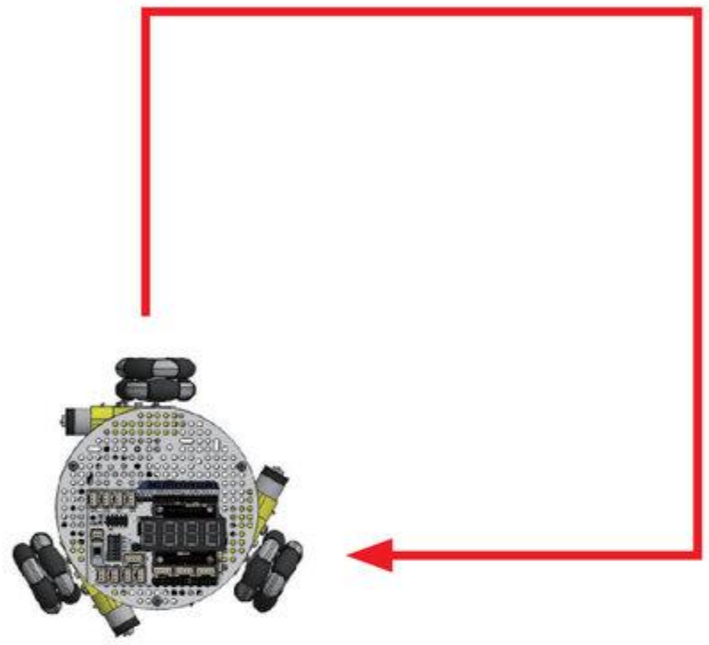


図 1-4　プログラム「OmniDriveTest」の動作イメージ

実行結果：**図 1-4** のように右回りに四角の軌跡（きせき）を描くように動く。四角の軌跡（きせき）を描いた後は、その場で回転し続ける。

まっすぐ動かなかったり、正しい動きができなかったりする場合は、モーターの配線が間違っていないか、オムニホイールのネジをきつくしめ過ぎていないか、などを確認しましょう。

さらに、以下の黄色の部分に調整値を入れて、より精度を高めましょう。なお、まっすぐ動く場合は、調整値の変更（へんこう）は不要です。

プログラム「OmniDriveTest」より抜粋（ばっすい）

```
RPomniDirect omniBot(1.0f,1.0f,1.0f,40.0f);
```

2）センサーの動作確認

続いてセンサーの動作を改めて確認しておきましょう。
以下のプログラムを実行しましょう。

プログラムの書き込み
RoboticsProfessorCourse2 > Inverted4 > AngCtrlOmniTemp

実行結果：はじめは 7 セグメント LED の周りを光がまわる（センサーが初期値を探しています）。動かさずにしばらく待ち、数値が表示されてから z 軸を中心に回転させると、7 セグメント LED の数値が変化する。初期位置から時計回りでマイナスの角度値、反時計回りでプラスの角度値が表示される。

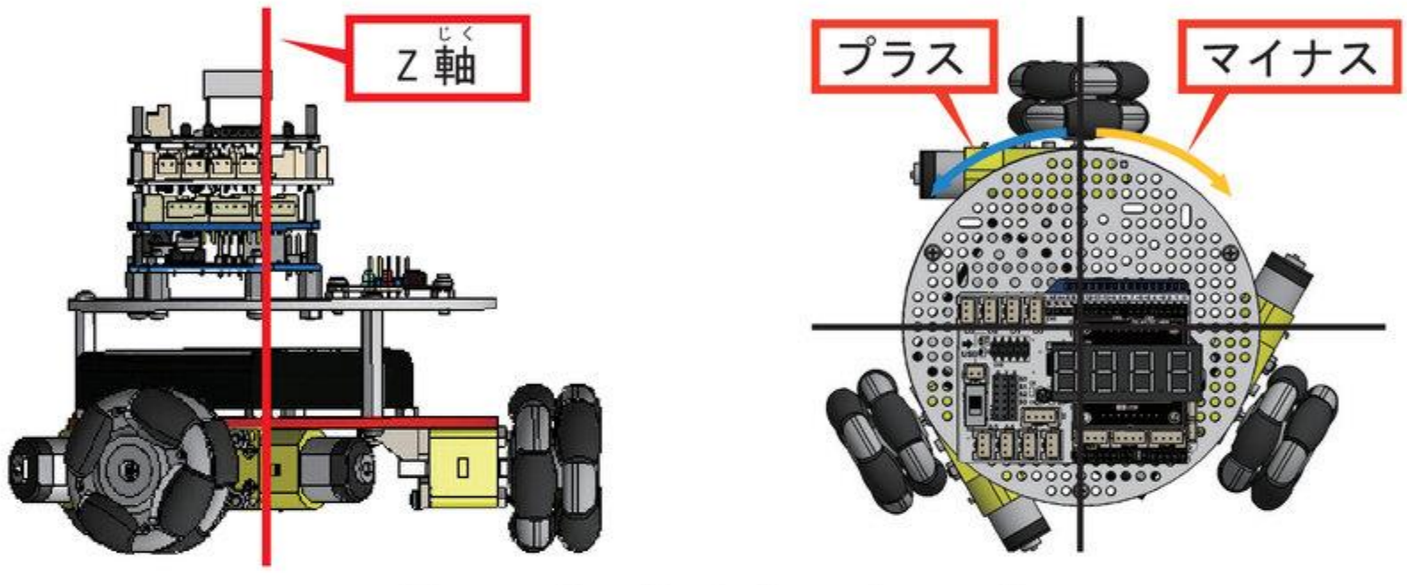


図 1-5　プログラム「AngularLED」

動作確認ができたら、次に進みましょう。ここでうまく動かないと、この後のプログラムも正しく動かなくなるので、しっかりと確認をしておきましょう。

2. フィードバック制御（目安 15 分）

2.0. フィードバック制御とは

1）フィードバック制御

突然ですが、地面にかかれた直線上を、自転車で走るときのことを考えてみましょう。
線から外れてしまったときは、左右どちらに逸れてしまったかを判断して、線に戻れるようにハンドルを操作しますね。

図 2-0　自転車で線をなぞる

このように、目標値と現在の値を常に検知して、目標値に近づくような処理を行い続ける制御を「フィードバック制御」といいます。
エアコンの暖房機能なども、もし設定温度より室温が低ければその差に応じた強さで温風を出し、設定温度より室温が高ければ送風を停止します。これも立派なフィードバック制御です。
ちなみに、フィードバック制御では、目標値と現在の値とのずれを「偏差」とよびます。22 度の設定なのに室温が 20 度であれば、偏差は 2 度（正確にはマイナス 2 度）です。

さて、自転車の例でいえば「線からどれだけ外れているか」が偏差にあたります。この偏差が大きいほど、つまり線から大きく外れているほど、線に戻るためのハンドル操作量も増加します。
外れた距離が 2 倍に増えれば、当然ハンドル操作も 2 倍に増えます。

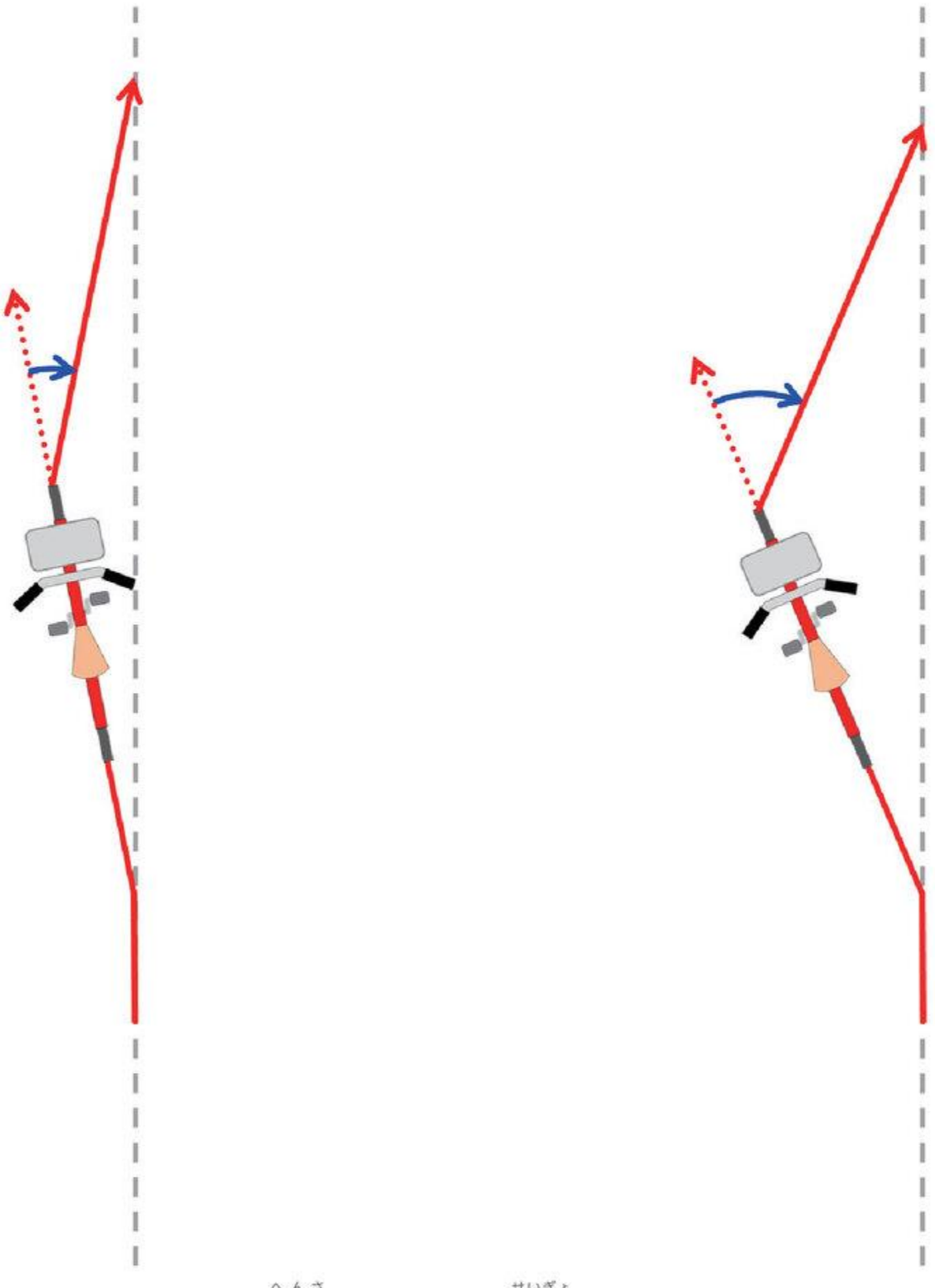


図 2-1　偏差が大きいと、制御量も大きくなる

つまり、「偏差が 2 倍、3 倍、…になると、必要な制御量（出力）も 2 倍、3 倍、…になる」ということです。どこかで聞いたことがある言いかたですね。そう、比例です。
このように、偏差と比例した出力を行うフィードバック制御を特に「比例制御」とよびます。
また「Proportion（比例)」の頭文字をとって「P 制御」とよぶ場合もあります。

2）フィードバック制御とゲイン

さて、たったいま学んだ比例制御ですが、もう少し理解を深めましょう。
たとえば、目標との偏差が 10、20、30、…と増えたとき、機械 A は出力値が 10、20、30、…と変化し、機械 B は出力値が 20、40、60、…と変化するとしましょう。この場合、比例制御を行っていると言えるのは機械 A と機械 B、どちらでしょうか？

表 2-0　偏差と出力値の関係

偏差	10	20	30	…
機械 A の出力	10	20	30	…
機械 B の出力	20	40	60	…

出力値こそ違いますが、どちらも偏差が 2 倍、3 倍、…になったとき出力も 2 倍、3 倍、…になっていることには変わりないですね。よって、正解は「どちらも比例制御を行っている」です。

では、機械 A と B は何がちがうのでしょうか？

機械 A は、出力値と偏差の値が等しくなるように決められています。つまり機械 A では「出力値＝１×偏差」の関係になっているということです。それに対して、機械 B では「出力値＝２×偏差」の関係です。算数や数学で比例を学んだ時、「比例定数」という言葉が登場しましたね。機械 A と B はどちらも比例制御ではあるものの、比例定数が異なるのです。

このように、比例制御では偏差に何らかの数をかけて出力値を決めることが多いです。このかける値のことを「ゲイン」と言います。今回は比例制御におけるゲインなので、特に「比例ゲイン」とよびます。

豆知識

ゲインを変数にするときは頭文字の「K」がよく使われます（ドイツ語の「Konstante（定数）」の頭文字です）。特に比例ゲインであれば「Kp」などとするのが一般的です。

コラム　フィードバック制御とシーケンス制御

フィードバック制御は特定の目標値をキープさせたいときには便利な制御方法ですが、他にも場面に応じていろいろな制御方法があります。

フィードバック制御とは違った制御方法だと、たとえば「シーケンス制御」があります。洗濯機はいったんボタンを押せば「注水→洗濯→排水→注水→すすぎ→排水→脱水」と手順通りに洗濯をしてくれますね。このように、動作の内容とその順番をあらかじめ決めておくという制御方法です。

オーブントースターなども、加熱時間を決めたらその時間だけとにかく加熱をします。これもシーケンス制御の一種ですね。
もし「パンの表面温度をはかり、もし高すぎるようなら出力を下げ、低すぎるようなら出力を上げることで、最適な温度でパンを焼き続けられる」という高性能オーブントースターがあったら、それはフィードバック制御と言えます。しかし、一般的なオーブントースターであればシーケンス制御を行うので、パンが分厚すぎて焼き加減がイマイチなときも、逆に焼けすぎて黒こげになっているとしても、とにかく最初に設定した時間だけ、同じように加熱をしてしまいます。

ちなみに、ロボプロでは「タッチセンサーが押されたら停止するロボット」をつくったことがあります。これは「センサーを使っているからフィードバック制御」と思われがちですが、タッチセンサーに目標値があったわけでも、偏差の大きさによって動きを分けているわけでもありません。「タッチセンサーが押されたら止まる」という事前に決められた動作を行っているだけなので、これはシーケンス制御といえます。

フィードバック制御は「センサーを使っている」ことではなく「偏差の大きさを見て、なるべく偏差がなくなるように出力を決める」ことが特徴です。紛らわしいですが理解しておきましょう。

３．フィードバック制御の利用（目安 45 分）

ここでは、実際に「フィードバック制御」を利用したロボットをつくってみましょう。

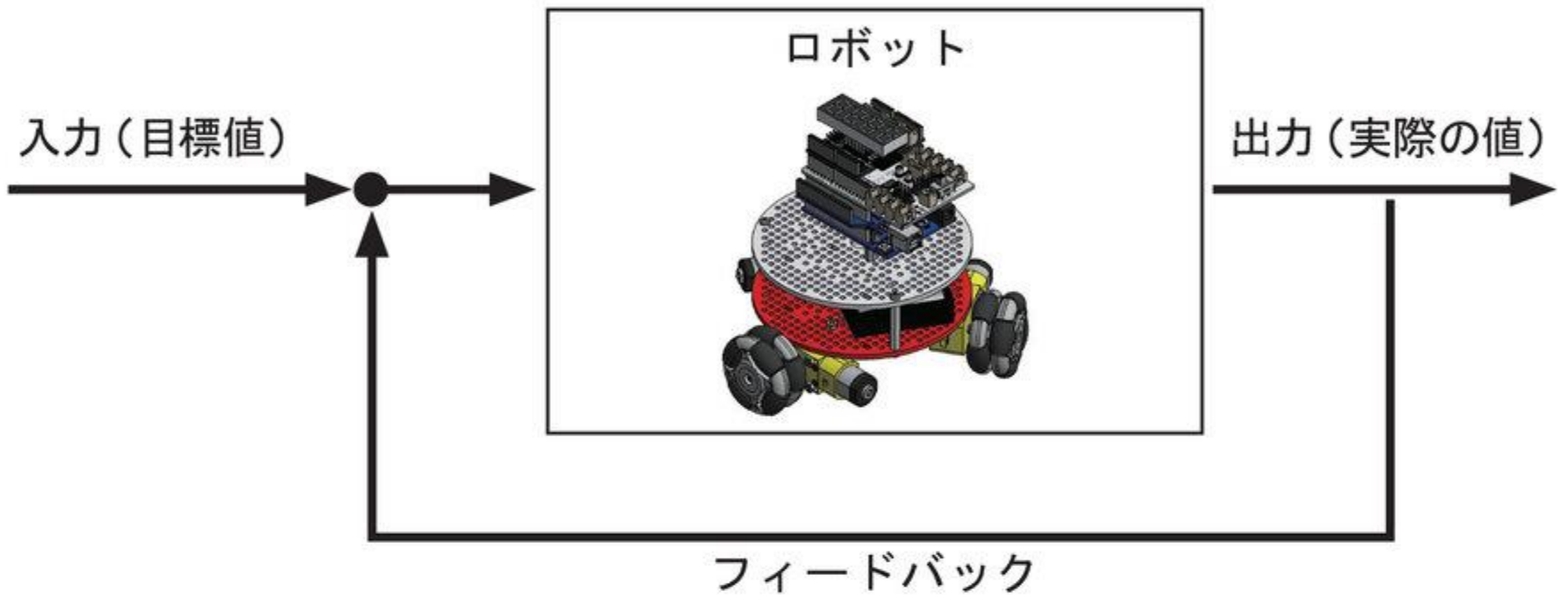


図 3-0　ロボットのフィードバック制御

3.0.　指定した角度を向くロボットを作る

１）作成プラン

今回は、あらかじめプログラム内に目標角度を書いておいて、その角度まで回転するロボットを作ります。フィードバック制御を活用して、目標角度からずれても自動で元の向きに戻れるような動きを実装してみましょう。全体的には、以下のような流れになるようプログラムに処理を加えていきます。

POINT

1．「初期化モード」と「制御モード」の２つの動作を実装する。
2．プログラムを起動したら、まず「初期化モード」として起動時の向きを計測し、その角度を０度としておく。
3．タッチセンサーを押したら「制御モード」に切り替わる。
4．「制御モード」の間は、以下の２つの処理を繰り返す。
　① いま向いている角度を計測し、目標角度との偏差を求める。
　② 求めた偏差をもとにモーターの出力値を算出し、目標角度を向くようにロボットを回転させる。

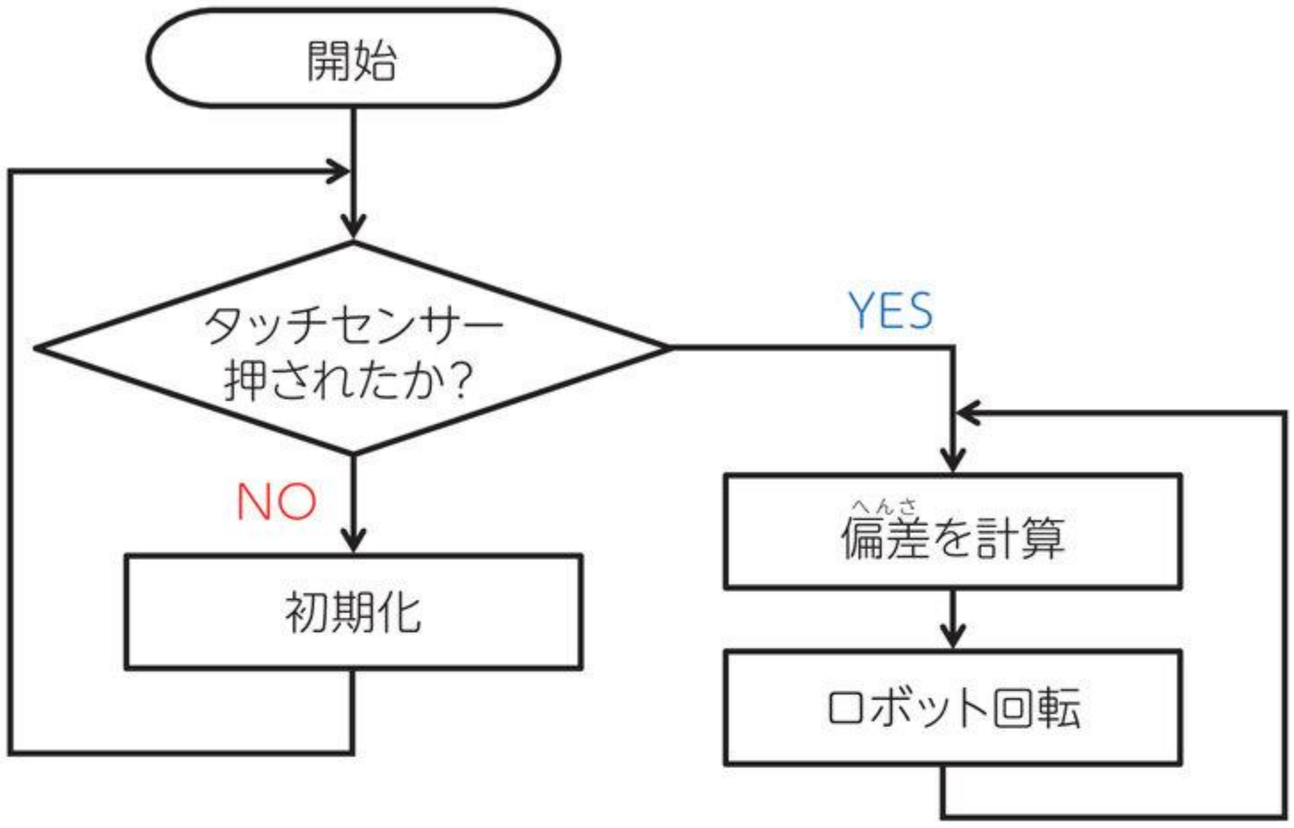


図 3-1　フィードバック制御ロボのフローチャート

フローチャートで表すと、上のような例ができますね。

まず「初期化モード」は以下のプログラムにつくってあるので、書き込んでおきましょう。

プログラムの書き込み
RoboticsProfessorCourse2 > Inverted4 > AngCtrlOmniTemp

プログラム「AngCtrlOmniTemp」より抜粋

```
accelgyro.getMotion9(&ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &mz);
    //姿勢センサーの検出
  ang += (( gz - ave ) / 16.4 / 8 ) * 0.0255;
    //+-250 range modeの変換 [deg/sec] x センシングに25.5msecかかっている
  lc.setDec(0, ang);
```

今回は Z 軸を中心とした回転速度である gz を元に角度を算出しています。
void loop()内では、gz を使って ang という変数の値を求めていますね。これが「起動時からの回転角度」を表す値です。とはいえ、今はまだ ang の値を 7 セグメント LED に表示させているだけなのでロボットは動きませんね。

あとは、「タッチセンサーが押されたら動作モードを切り替える処理」と、「フィードバック制御で回転する処理」を追加すれば完成です。
1つずつ書いていきましょう。

2）スイッチ（タッチセンサー）機能

では１つ目の機能を追加します。プログラム「AngCtrlOmniTemp」に、赤字の部分を追加しましょう。

```
void setup(){
(中略)
        lc.setLed(0, rx[i % 12], ry[i % 12], LOW);
                                          // 7セグメントLEDルーレット処理
    }
    ave /= 500;
    pinMode(D1, INPUT_PULLUP);
}
```

つけ加えた部分は以下のような命令となります。

命　　令「pinMode」
実行結果：ピンの動作を入力か出力に設定する
使 い 方：pinMode([ピン番号],[入出力])

[ピン番号 (pin)]　設定したいピンの番号（タッチセンサーであれば D0 ～ D3 または A0 ～ A5 ）を入れる
[入出力 (mode)]　INPUT 、OUTPUT 、INPUT_PULLUP のどれかを入れる

ピンの動作を設定しているわけです。
ただし設定したいピンに他の機能が接続されている場合は、使えないことがあるので注意しましょう。

続いては、実行動作の loop( ) につけ加えます。以下の黄色のラインの部分をつけ加えましょう。

```
void loop(){
    while(digitalRead(D1) == OFF){
    }
    while(1){
    accelgyro.getMotion9(&ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &mz);
        //姿勢センサーの検出
    ang += (( gz - ave ) / 16.4 / 8 ) * 0.0255;
        //+-250 range modeの変換 [deg/sec] x センシングに25.5msecかかっている
    lc.setDec(0, ang);    //z軸周りの角度のLED表示(deg)
    }
}
```

「初期化モード」のうち、起動時の角度の計測は void setup() 内ですでに終わっています。よって、void loop() 内ではタッチセンサーが押されるまでにしておく処理はもうありません。

追加したプログラムのうち最初の 2 行は「 D1 のタッチセンサーが押されるまでは、“何もしない”を繰り返す」という処理になっています。

次の行にも while が出てきますが、カッコの中が式の形になっていませんね。

while(1) という while 文は、直訳してみると「1 が成立する（true である）あいだはくり返す」となりますね。Arduino では、「0」以外の数値単体を条件式として使うと「true である」と見なされます。よって、while(1) で囲まれた部分が電源を切るまでずっとくり返されることになります。

豆知識

他のプログラミング言語だと、このルールが異なることもあります。たとえば、「0」「1」でなく「false」「true」という文字列でないと反応しないもの、「0」も「true である」とみなすものなどもあります。もし他のプログラミング言語に触れる機会があったら、こうした基本ルールをまずチェックしておきましょう。

３）目標角度の設定と偏差の算出

続いて、void setup() と void loop() の間に変数の宣言を追加します。

```
void setup(){
(中略)
}

float ang = 0;
int Ref = 0;

void loop(){
```

ここで宣言した変数 Ref が、目標角度を入れる変数です。
たとえば int Ref = 90; と変えてプログラムを実行したら、ロボットが 90 度回ったところで止まってくれるようなプログラムにしていけばよいですね。

あとは、引き続き「制御モード」のプログラムつくっていきましょう。

４．「制御モード」の間は、以下の２つの処理を繰り返す。
① いま向いている角度を計測し、目標角度との偏差を求める。
② 求めた偏差をもとにモーターの出力値を算出し、目標角度を向くようにロボットを回転させる。

いま向いている角度の計測はすでにできていましたね。変数 ang が現在角度でした。
あとは、上の赤字の部分にあたる処理だけです。

ステップアップ

プログラムに赤字部分の処理を追加して、指定した向きを向くロボットを完成させよう！

ヒント

実はこのプログラムには、すでにオムニホイールロボットを動かす命令 `omniBot.move(x, y, z);` が使えるようになっているよ！
(x, y, z); の部分にはそれぞれ左右の移動速度、前後の移動速度、回転速度が入るから、今回は 3 つ目の値だけ指定すればいいね。
偏差が大きい（＝目標角度から大きく離れている）ときほど回転速度が速くなるようにしてみよう。ただ、あまり速く回転しすぎると角度計測が上手くいかなくなるから、± 30 くらいの範囲にとどめておこう。

講

解答例は以下のプログラムです。
RoboticsProfessorCourse2 > Inverted4 > AngCtrlOmni

まず現在角度と目標角度の偏差、つまり `ang - Ref` を求める必要があります。求めた偏差を入れておくため、`Err` という変数を宣言しています。また、± 30 に制限して求めた最終的な回転速度を入れるのにも `spd` という新たな変数を使っています。

3.1. フィードバック制御のゲインを変えてみる

これで、目標角度からの偏差にあわせるように出力値を変えるフィードバック制御のプログラムが出来ました。
ところで、今つくったプログラムには「ゲイン」がありませんね。試しに「ゲイン」を追加してあれこれ変更し、動きがどう変化するか確かめてみましょう。
まずは、「ゲイン」を入れておくための変数を作ります。

```
void setup(){
(中略)
}

float ang = 0;
int Ref = 0;
int Err = 0;
int Kp = 1;

void loop(){
```

変数 Kp を、初期値１で宣言しました。これが「ゲイン」になります。
ゲインは先ほどの説明通り「比例定数」にあたります。つまり、偏差とかけ算するのに使われる値です。よって、ゲインを０にしてしまうとかけ算して求めるはずの出力まで０になってしまうため、初期値を１にしています。
では、Kp をゲインとして組み込んでみましょう。

チャレンジ課題

出力を算出するとき、変数 Kp をゲインとして使うような処理を追加してみよう！
できたらゲインの値を上下させてみて、ロボットの動きがどのように変化するか確かめてみよう！

講

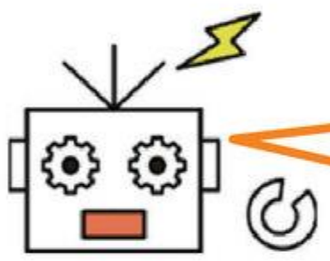
解答例は以下の通りです。

```
spd  = constrain(Kp * Err, -30, 30);
```

ゲインの説明ページに書いた通り、偏差にゲインをかけて出力を求めるようにすればよいので、偏差 Err に Kp をかけています。
Kp の値が１であれば動きは変わりませんが、Kp の値を増やせば少しの偏差でもすぐ回転速度が上がるようになります。素早く目標までたどり着けますが、勢い余って目標を通り越して小刻みに左右に振れるようになるかもしれません。ゲインを小さくすれば逆に落ち着いた動きになりますが、目標角度に近づくまえに出力不足でモーターが回転しなくなる恐れがあります。ちょうどいい値を調整させて下さい。

４．まとめ（目安５分）

今回は姿勢検出シールドを使って、フィードバック制御について学びました。ねらい通りに動くロボットになりましたか？　ジャイロセンサーなどの環境にたよるロボットを作る場合、その環境により誤差が生じたり、バッテリーの残量によって精度が落ちたりすることがあります。ただし、ここを攻略すると、ロボットが完成したときに大きな達成感があるので、今回うまくいかなかった人も再チャレンジしてみましょう。
さて、やっとこのタームのゴールである「倒立振子ロボット」を動かすために必要な知識を得ました。次回以降、組み立てから制御まで一気に進めていきます。
楽しみにしていてください！



次回は、転びそうで転ばない倒立振子ロボットの製作開始！

講

○　以下の理解度を確認します。
・ロボットを組み立てる
・フィードバック制御を理解する
・フィードバック制御を利用してロボットを動かす

○　次回のテーマは、「倒立振子ロボットの製作」であることを告知します。

≪次回必要なもの≫

次回は、今回使ったロボットの他に以下のパーツを準備しておきましょう。

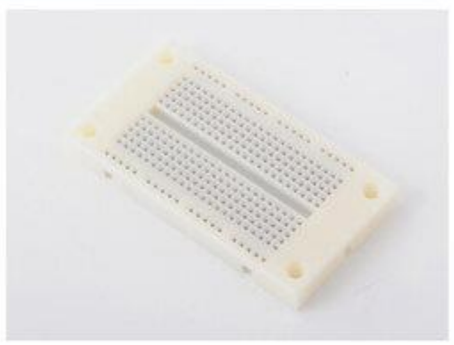
ラジオペンチ 1	ドライバー 1	USB ケーブル 1	リボンケーブル 1
			
無線受信モジュール 1	**ユニバーサルボード 1**	**301 ブレッドボード 1**	**ジャンパー線 65**
			
可変抵抗ボリューム 2	**タイヤ 2**	**M3L8 ネジ 5**	**25mm 角スペーサー 8**
			

図 4-0　次回必要なもの