講師用

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

六脚(ろっきゃく)ロボット③

（第５回/第６回テキスト）

必ず、生徒に授業日と自分の名前を記入させるようご指導をお願いいたします。

第(だい)５回授業日(かいじゅぎょうび)　2024年　　月　　日

第(だい)６回授業日(かいじゅぎょうび)　2024年　　月　　日

名(な)前(まえ)

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9月授業分

3B3K_24.09

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

六脚(ろっきゃく)ロボット③

第5回

六脚(ろっきゃく)ロボットの自律化(じりつか)

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。

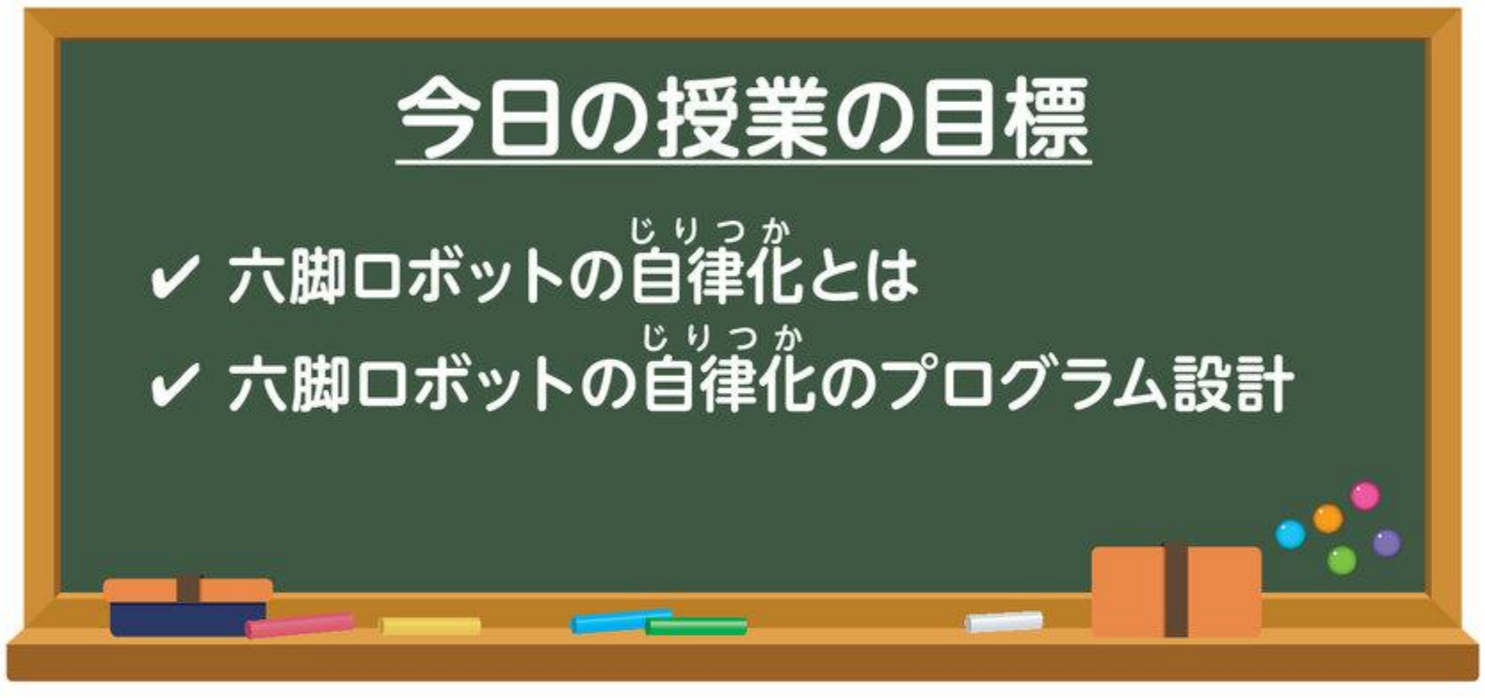
○　今回の目標をパネルで用意するか、黒板に予め書いておきます。

（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。

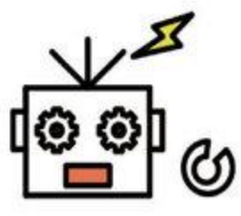
0.　六脚ロボットの自律化（目安10分）

0.0.　「六脚ロボットの自律化」でやること



今回の授業は、六脚ロボットの自律制御です。
前回、六脚ロボットをコントローラーで自在に操縦できるようにしました。それはそれで面白いのですが、これでは、ただのラジコンでしかありません。さらに進化させて、ロボット自身に判断して行動してもらいたいですよね。という訳で今回は、取り付けられたタッチセンサーの状態によってロボットに自律的に行動選択させる方法を学んでいきましょう。
ところで、「じりつ」という言葉には、**自立**と**自律**の表現があります。この２つはどう違うのでしょうか？ 自立ロボットは、電源や制御装置など必要な機能が全て備わっているロボットです。一方、自律ロボットは何らかの規律に従って自分自身で判断し行動できるロボットを意味します。
前回までのコントローラーで操縦する六脚ロボットは自立ロボットでしたが、自律ロボットではありませんでした。今回は、これまでの「自立型六脚ロボット」を、「自律型六脚ロボット」へと発展させましょう。

図0-0　自立と自律



今日から自律ダ！

0.1. 必要なもの

前回に引き続き、六脚ロボット本体とコントローラーを使用します。
プログラムを実行して動かす前に、タッチセンサーに取り付けた針金の形を修正し、関節のネジの緩みを直しておきましょう。

USB ケーブル 1	コントローラー 1	AC アダプター 1
		

図0-1　必要なもの

講

電池残量によってはロボットが正しい動作を行わないことがあります。
そのような場合は、ACアダプターを接続してください。

1.　ロボットの行動の自律化（目安90分）

1.0.　ロボットのモード切り替え

自律的に動くロボットにするにあたり、最初に考えるのはロボットを停止させる方法です。自律的に動かそうとすると、人間が予測しない動作をすることもあるので、電源をOFFにする以外に停止させる操作を加える必要があります。また、電源をONにした直後に動き出してしまうロボットは危険ですので、電源をONにした直後の起動時は、「操縦モード」にしておいて、コントローラーのボタン操作をしないときは動かないようにして、コントローラーの START ボタンを押すことで、「自律モード」に切り替わるようにします。
「自律モード」ではタッチセンサーが障害物に当たったら自動的に移動方向が変わるように設定していますが、コントローラーの SELECT ボタンを押すと、「操縦モード」になりコントローラーで操作できるような切り替え機能にしていきましょう。

「操縦モード」のときは、赤枠のボタン操作です。
「自律モード」のときは、タッチセンサーの状態によって移動方向が変わります。

図1-0　自律モードを追加した操作

前回使ったプログラム「RemoteWalk2」のフローチャート（**図1-1**）では、20ミリ秒ごとのタイマー割り込みで、コントローラーのボタン状態を読み取り、ロボットの前進・旋回・停止といった行動を決める歩行モードの mode_flag を切り替え、加えて、メインループでは、歩行モードに応じて脚の動きを20ミリ秒周期で更新していくように設定していました。今回のロボットの自律モードでは、コントローラーのボタン操作ではなく、2つのタッチセンサーの状態によって切り替えていくことで、六脚ロボットに自律行動をさせます。

これには、タイマー割り込みの処理部分を修正する必要があります。(第4回のフローチャートも参考にしましょう。)

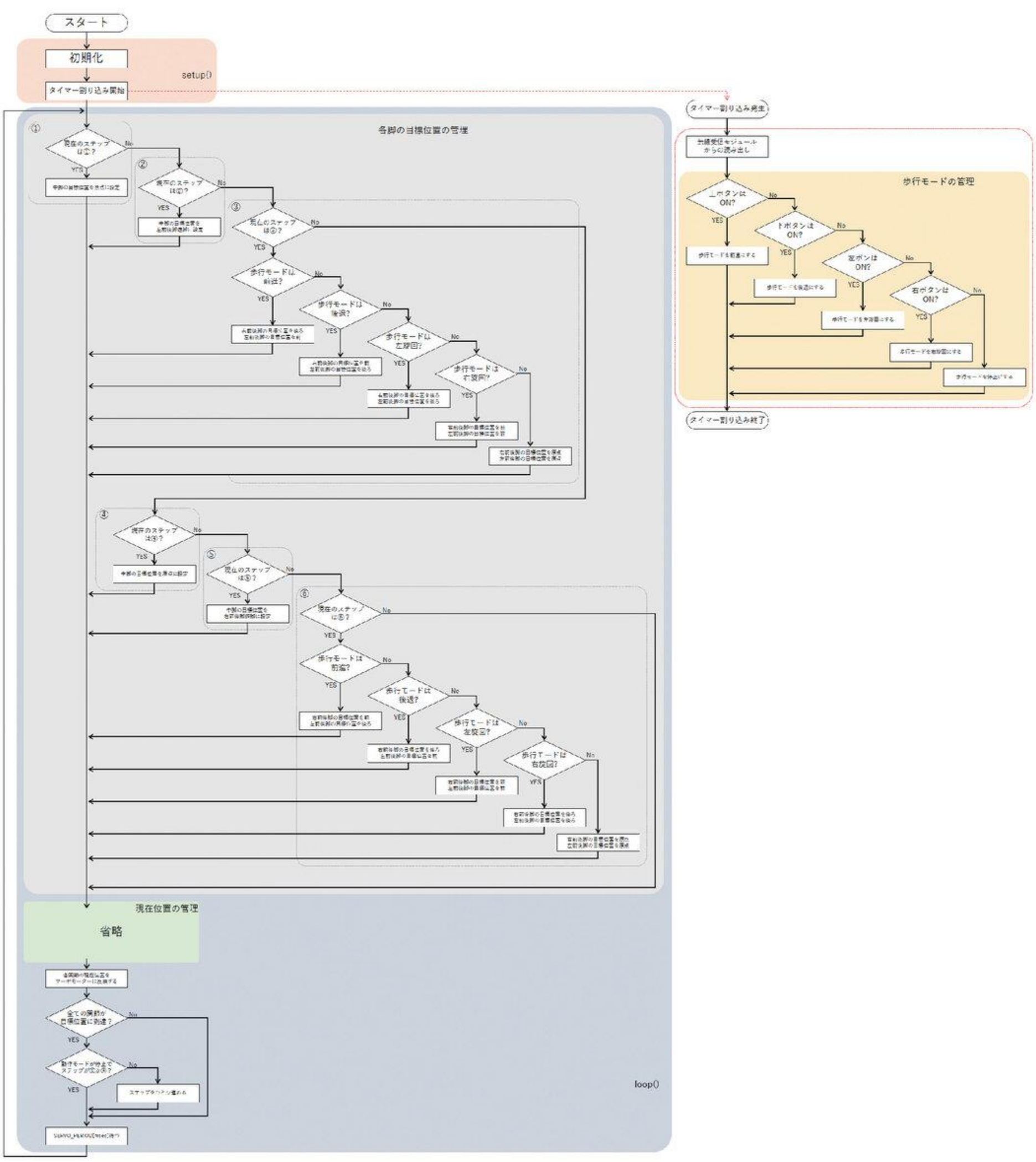


図1-1　プログラム「RemoteWalk2」のフローチャート

講　プログラム「RemoteWalk2」のフローチャートの拡大図が巻末にあります。

1.1. ロボットの自律化に必要な処理

では、プログラムを追加・変更するにあたって、追加で必要になる機能は何かを考えていきましょう。
まず、今回のプログラムではコントローラーで操縦する**「操縦モード」**と、ロボットがタッチセンサーの情報をもとに自律行動する**「自律モード」**の２つのモードを用意します。そして、このモードの切り替えによって、タイマー割り込みの処理が変わるので、自分が今どっちのモードにいるのかを、覚えておく必要があります。
そこで、次のような管理用のフラグを用意します（**操縦モード**と**自律モード**の切り替え）。

```
#define    CONTROL_REMOTE        0          // 操縦モード
#define    CONTROL_AUTO          1          // 自律モード

int    mode_flag;                           // 歩行モードの管理用
int    step_flag;                           // 動作ステップの管理用
int    control_flag;                        // 動作モード（操縦・自律）の管理用
```

この動作モードの管理用のフラグは、コントローラーの START または、SELECT が押された時に切り替えを行い、押されない時は、直前の状態を継続します。
タイマー割り込み中は、動作モードの管理フラグが、**「操縦モード」**のときは、コントローラーの十字ボタンや L2 、 R2 ボタンの操作によって歩行モードの管理フラグを切り替えたり、ツノの目標位置を変更したりする従来の操縦の処理を行います。一方、**「自律モード」**の場合は、２つのタッチセンサーの状態によって歩行モードの管理フラグを切り替えます。
処理の流れは**図1-2**を参考にしてください。

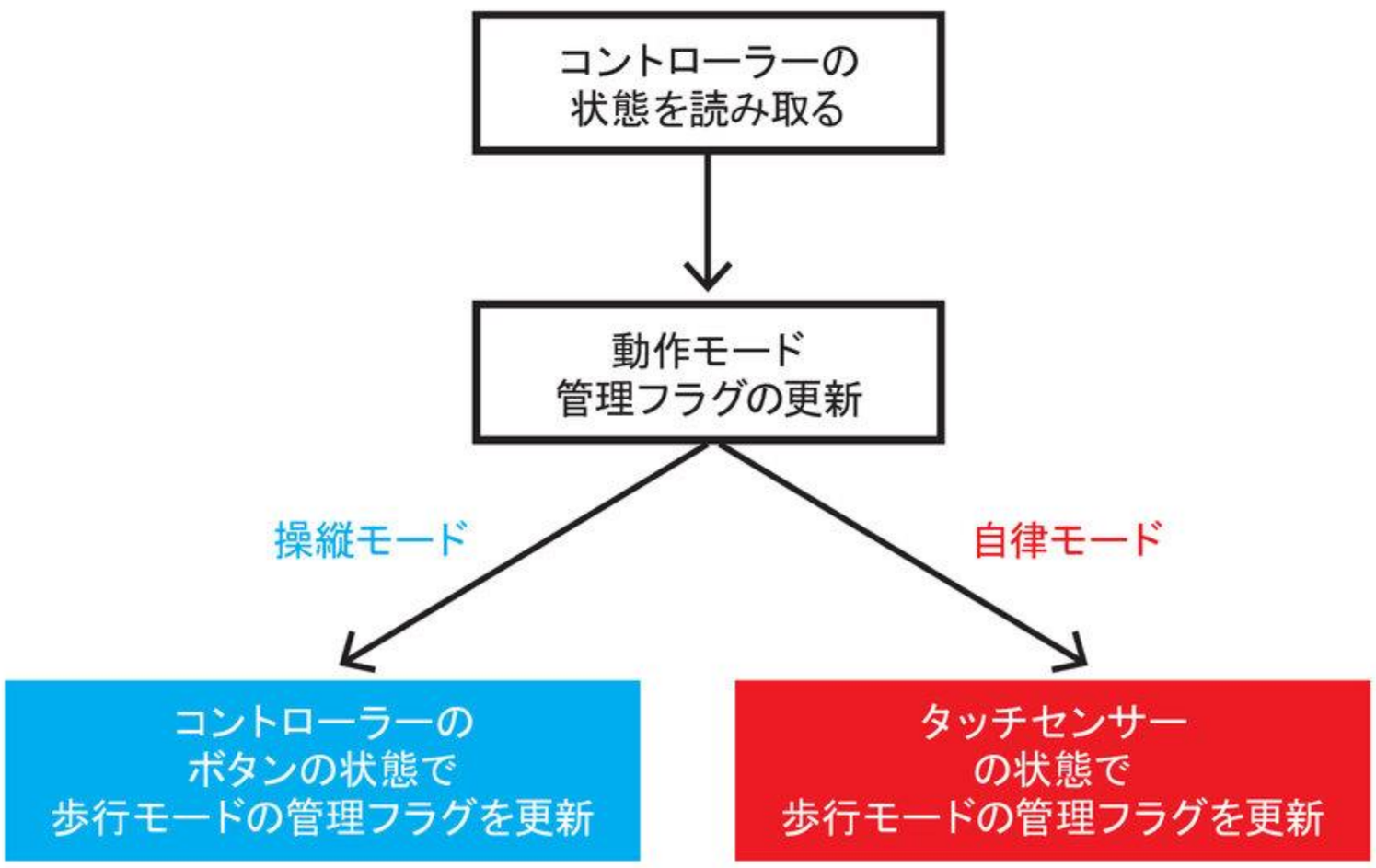


図1-2　タイマー割り込み処理の流れ

次に、**図1-2**の流れに沿って作ったプログラムを説明していきます。

1）コントローラーの状態を読み取る

このプログラムは以下になります。

```
void  gamepad_update(){
    ps2x.read_gamepad();      // コントローラーから値を読み取る
```

2）動作モードの管理フラグの更新

1）でコントローラーの状態を読み取ったので、次に START ボタンと SELECT ボタンが押されているかどうかを確認して、モードを切り替える処理を行います。このプログラムは以下になります。

```
if (ps2x.Button(PSB_START)     == HIGH) { // もしスタートボタンがおされたら
    control_flag = CONTROL_AUTO;           // 自律モードにする
}
else if (ps2x.Button(PSB_SELECT)     == HIGH) {
// もしセレクトボタンがおされたら
    control_flag = CONTROL_REMOTE;         // 操縦モードにする
}
```

3）コントローラーのボタンの状態で動作モードの管理フラグを更新

この部分は、従来の処理をそのまま使います。
この条件分岐の場合、すでに**「自律モード」**のときに START が押されると、動作モードの管理フラグを**「自律モード」**にする処理が実行されますが、もともと**「自律モード」**になっているので変化はありません。
「操縦モード」も同じです。仮に、START と SELECT が同時に押されている場合は、先に START が押されているかどうかの条件分岐に入るため、**「自律モード」**に切り替わります。
もし、この方法を変更したい場合は、この条件式を、「一方のボタンが押されていて、なおかつ、もう一方のボタンが押されていないとき」というプログラムに変更することで変更が可能になります。

4）タッチセンサーの状態で歩行モードの管理フラグを更新

ここでは、六脚(ろっきゃく)ロボットの触角の役割をしている2つのタッチセンサーの状態によって、歩行モードの管理フラグを更新する処理(しょり)を行います。タッチセンサーは左右に配置され、ON、OFF、2つの状態があります。それぞれの状態の組み合わせは**表1-0**のように4パターンあります。

表1-0　歩行モードの切り替え

左タッチセンサー	右タッチセンサー	動作（移動方向）
OFF	OFF	前進
OFF	ON	左旋回
ON	ON	後退
ON	OFF	右旋回

やってみよう！

左右のタッチセンサーの状態により、六脚(ろっきゃく)ロボットにどのような動作をさせれば良いか考えて、**表1-0**の動作の欄に解答を書き込(こ)んでみよう。

講

まずロボットの動作の妨げになる障害物がどの方向にあるかを想定させてください。
障害物がある方向と逆の方向に移動するようにし、歩行モードをプログラムします。

1.2. ロボットの自律化プログラムのポイント

それでは、ここまで説明した処理をまとめたプログラムをつくってみましょう。

やってみよう！

第4回プログラム「RemoteWalk2」を書きかえ、タッチセンサーを利用した自立歩行制御を追加してみよう！

ヒント

タッチセンサーをチェックする条件分岐そのものは、これまでに扱ってきたif文で簡単につくれそうだね！
このif文をどこに入れるかを見抜くのが、今回のポイントだよ！

講

解答例は以下のプログラムです。
RoboticsProfessorCourse3 > HexRobot5 > HexCrawler0

自律モードの際、角が初期位置のままだとタッチセンサーより角の方が前に出ているため、センサーがうまく押されません。よって、自律モードに変更された際に角を上げるような処理を追加しています。
これは、実際にプログラムをつくってみた際にはじめて発見される問題になるはずです。こういった偶発的な問題に対応する力を身につけられるよう、適切なレベルでの声掛けをお願いいたします。

図1-3　自律モードを追加した操作

前のページの「やってみよう！」の解答例は以下のプログラムです。

プログラムの書き込み

RoboticsProfessorCourse3 > HexRobot5 > HexCrawler0

プログラム「HexCrawler0」のプログラムのポイントを整理していきます。

プログラム「HexCrawler0」より抜粋

```
#define    CONTROL_REMOTE          0          //操縦モード
#define    CONTROL_AUTO            1          //自律モード

int mode_flag;                                //歩行モードの管理用
int step_flag;                                //動作ステップの管理用
int control_flag;                             //動作モード(操縦・自律)の管理用

(中略)

void setup(){
    mode_flag    = MODE_STOP;
//歩行モード停止(コントローラーのボタンの状態の取得前が不定だと困るため)
    step_flag    = STEP_1;                    //ステップは１からはじめる
    control_flag = CONTROL_REMOTE;
//操縦モードではじめる(電源を入れると勝手に動き出してしまうため)
```

「自律モード」のときの処理を説明します。

左タッチセンサー digitalRead(D2) と、右タッチセンサー digitalRead(D3) の状態を読み取り、その状態にあわせて、歩行モードを切り替えます。

さきほどの**表1-0**を完成させると以下のようになります。この内容をプログラムに反映します。

表1-1　歩行モードの切り替え解答

左タッチセンサー	右タッチセンサー	動作（移動方向）
OFF	OFF	前進
OFF	ON	左旋回
ON	ON	後退
ON	OFF	右旋回

プログラム「HexCrawler0」より抜粋

```
if (control_flag == CONTROL_AUTO){
    //ツノを上げる(自律の時はツノが邪魔になるため)
    targetH = H_TARGET_POSITION_UP;

    //触角センサーの状態によって行動を決定
    if (( digitalRead(D2) == HIGH ) && ( digitalRead(D3) == HIGH )){
        //もし左右のセンサーが反応したら
        mode_flag = MODE_BACKWARD;              //歩行モードを後退にする
    }
    else if (digitalRead(D2) == HIGH){
        //もし左のセンサーが反応したら
        mode_flag = MODE_TURN_RIGHT;            //歩行モードを右旋回にする
    }
    else if (digitalRead(D3) == HIGH){
        //もし右のセンサーが反応したら
        mode_flag = MODE_TURN_LEFT;             //歩行モードを左旋回にする
    }
    else {
        //そうでなかった(左右のセンサーともに反応がなかった)ら
        mode_flag = MODE_FORWARD;               //歩行モードを前進にする
    }
}
```

POINT

ifとelseを使って、左右タッチセンサーが(==HIGH)の場合は後退し、左タッチセンサーだけが(==HIGH)の場合は右旋回し、右タッチセンサーだけが(==HIGH)の場合は左旋回し、どちらのセンサーにも反応がない場合は前進をする処理をしています。
プログラムの黄色の部分では、「自律モード」ではタッチセンサーが障害物に当たるときにツノが邪魔になるので、ツノを上げておく処理を付け加えています。

1.3. 自律モードの改善

自律モードの動きはどうでしたか？
障害物に当たったとき、**図1-4**のような動きになってしまいませんでしたか？

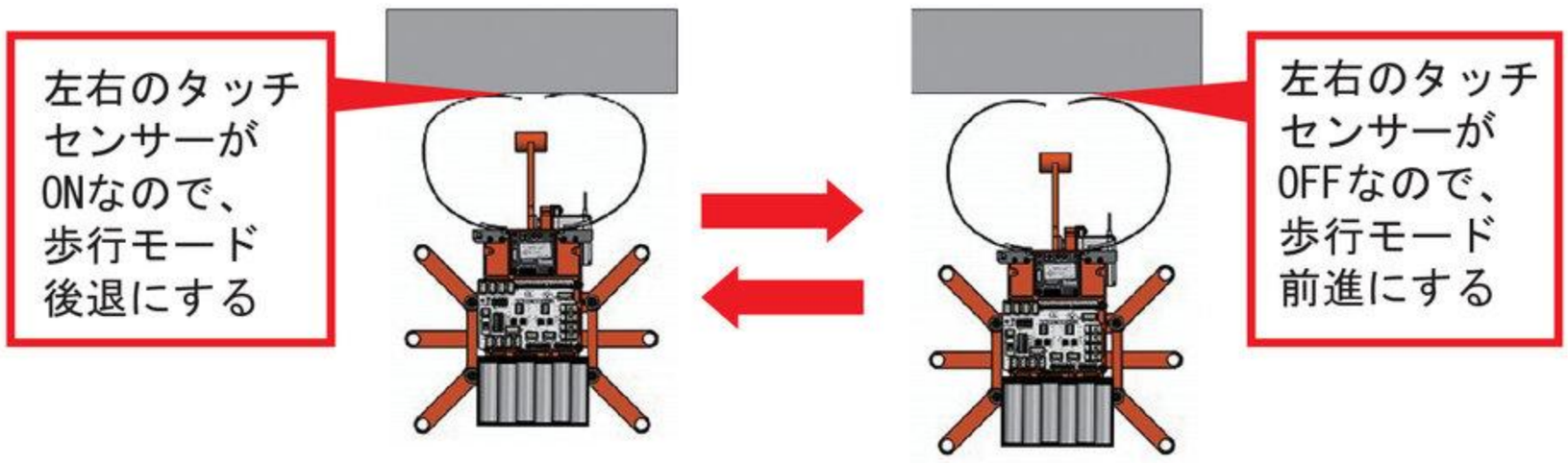


図1-4　自律モードの問題点

あまり賢くないですね。この問題を解決するため、これから自律モードのプログラムを改善していきましょう。
最終的には、障害物に当たって後退したあと、再び同じ方向に前進せず左右どちらかに避けることが必要になりますね。まずは、壁から十分離れることができるようなプログラムにしましょう。

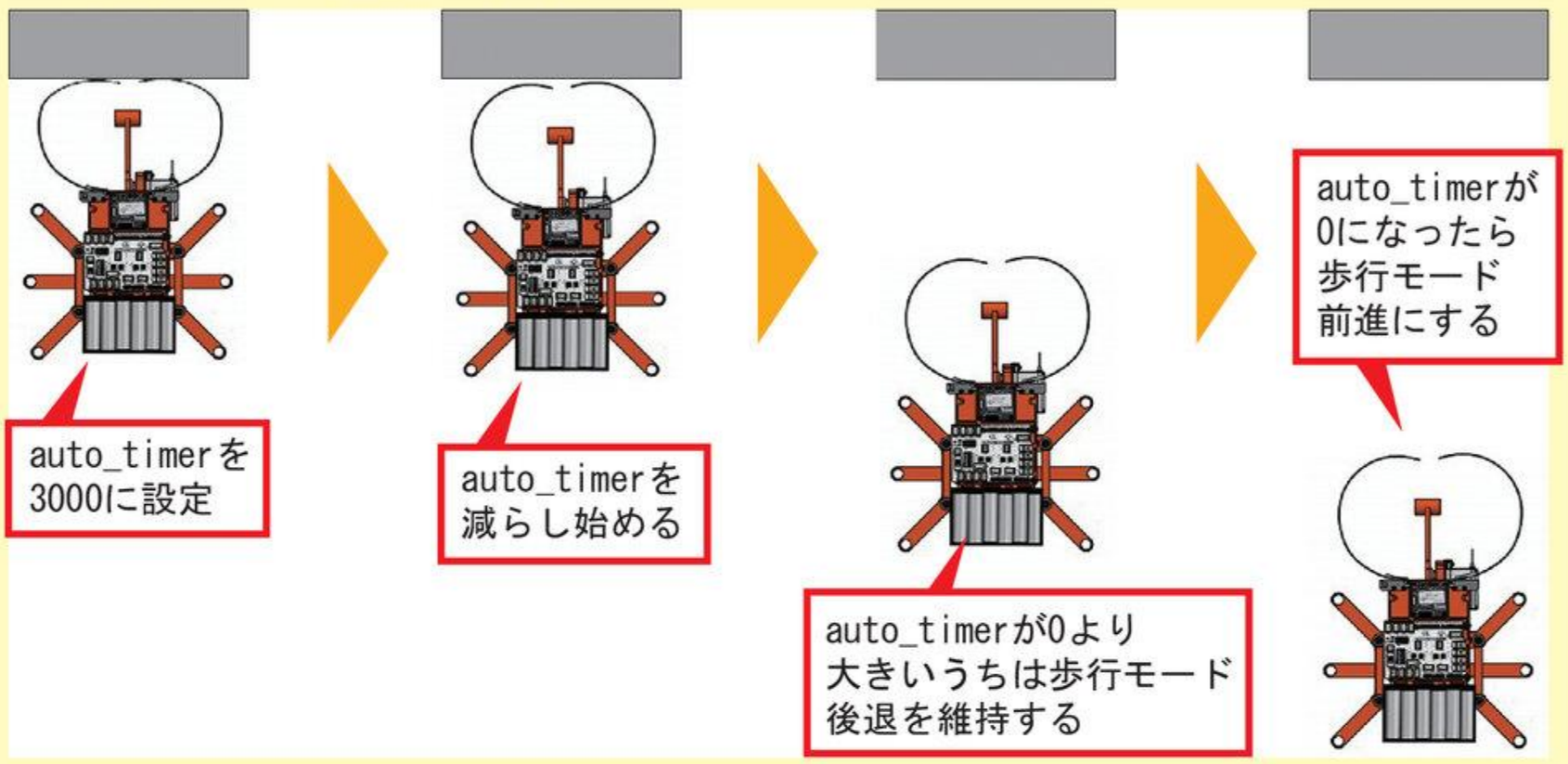
ステップアップ

プログラム「HexCrawler0」を書きかえ、「歩行モードが後退になったら、タッチセンサーがOFFになってからも3秒間は後退をし続ける」という動きにしてみよう！

ヒント

うまくできないという人は、以下の流れを参考にしてみよう。

・変数 auto_timer を宣言する。
・左右タッチセンサーがONならば auto_timer を3000にする。
・左右タッチセンサーがOFFならば auto_timer をカウントダウンするか歩行モードを前進にする。



講 タッチセンサーの状態読みとりが100秒ごとに行われるため、auto_timer のカウントダウンも100ずつ行う必要があります。

プログラムはつくれましたか？歩行モード後退のときだけでなく、右旋回、左旋回でも同様の処理を追加したのが以下のプログラムです。動作を確認してみましょう。

プログラムの書き込み

RoboticsProfessorCourse3　>　HexRobot 5　>　HexCrawler 1

1.4. 六脚ロボットに個性をもたせよう

プログラム「HexCrawler1」では、同じ値を auto_timer にセットして、どの回避行動を取る場合も同じ時間だけ動くようになっていますが、それぞれの回避行動に応じてセットする時間をかえることで、六脚ロボットの「自律モード」に個性を持たせることもできます。あまり回避行動に時間を取らないロボットや、右旋回だけ慎重に回避行動を続けるロボットなど、好みに合わせてプログラムをかえることが可能です。

チャレンジ課題

プログラム「HexCrawler1」をカスタマイズして、自分好みの六脚ロボットにしてみよう。
プログラム中で定義されているさまざまな値を変更して、動きの変化を確かめてみよう。

1.5. 自律モードのさらなる改善

さて、改良した「自律モード」でロボットを動かしてみてどうだったでしょうか？
「HexCrawler1」のプログラムでは、やはり同じ壁に当たる動作をくり返しがちです。この問題を解決するにはどうすればいいでしょうか？
今回の問題は、回避行動が一つの動作で完結し、すぐに前進動作に戻ってしまうところにあります。後退動作をしたら、タイマーがゼロになってもすぐに前進をしないで、左右どちらかの旋回動作を挟むことで、くり返し前進して壁に当たることを改善することができます。

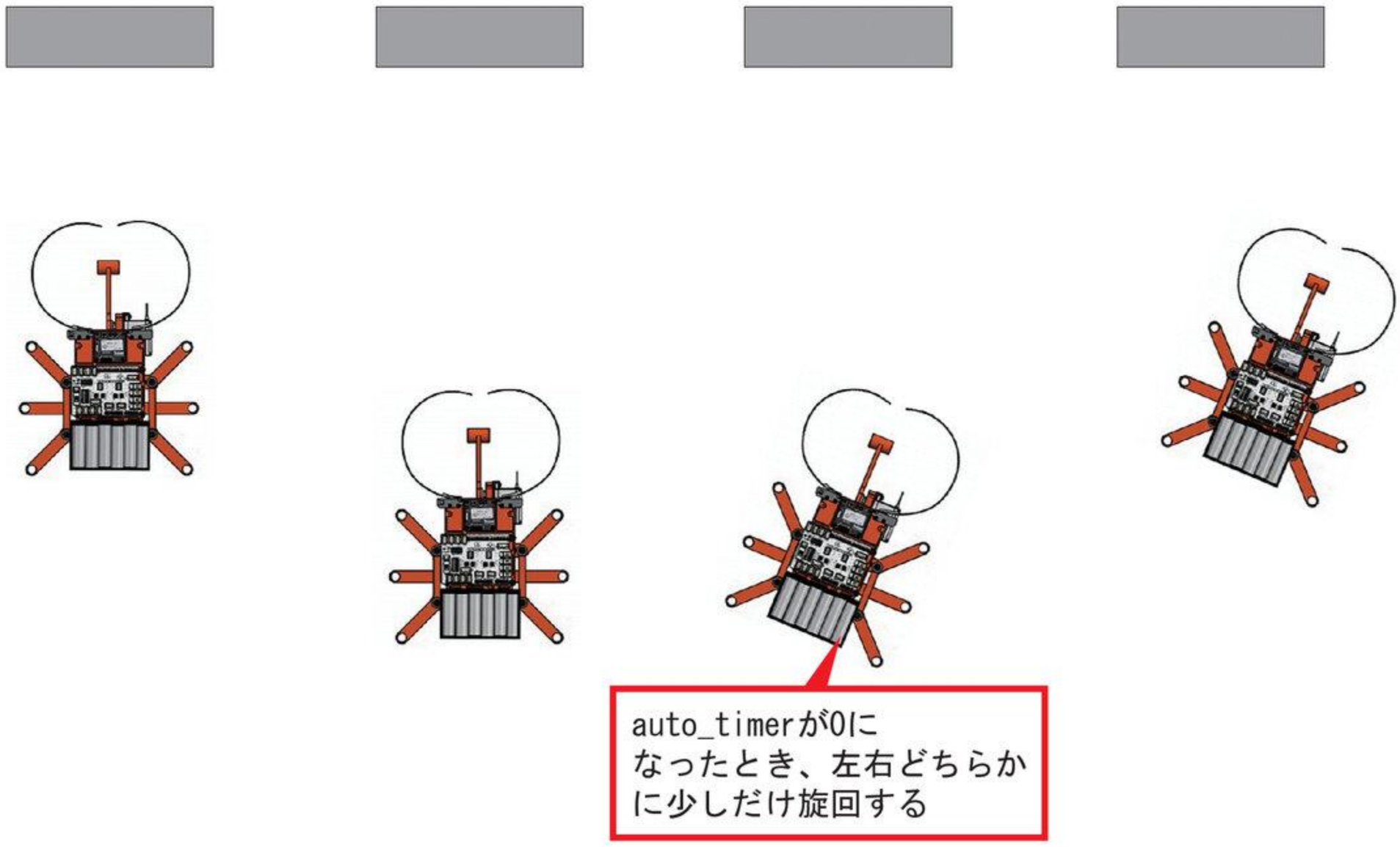


図1-5　自律モードのさらなる改善イメージ

それでは、まずこの部分を改善したプログラムを実行して、自律モードで動かしてみましょう。
どのような動きが追加されたでしょうか？

プログラムの書き込み

RoboticsProfessorCourse3　＞　HexRobot 5　＞　HexCrawler 2

このプログラムでは、乱数（random）という機能を使用しています。乱数とは、ランダム（不規則）に出される数字という意味で、イメージ的にはサイコロを振って出た数ということになります。
ロボットの行動を、サイコロを振ってたまたま出た数で決めると、人間がロボットの動作を完全に予測することはできなくなり、ロボットがまるで自律行動しているかのようになります。

プログラム「HexCrawler2」より抜粋

```
randomSeed(analogRead(0));
//乱数初期化（何もつないでいないアナログピンのノイズを利用する）

（中略）

if (mode_flag == MODE_BACKWARD){      //もし後退回避行動をしていたら
    if (random(1) == 0){              //0か1の乱数を発生させてもし1だったら
        mode_flag = MODE_TURN_LEFT;   //左旋回をする
    }
    else {                            //もし乱数が0だったら
        mode_flag = MODE_TURN_RIGHT; //右旋回をする
    }
```

豆知識

Arduinoで乱数を生成する際には「乱数列」という、数字が不規則に並べられた数列を使います。
たとえば「58300565039560104903」という乱数列があるとき、はじめの4文字をそのまま利用すれば5830という数値を得ることができますね。
ただ、Arduinoの乱数列は基本的に常に同じなので、「はじめの4文字を利用する」という処理にすると毎回「5830」が返ってきてしまい、乱数になりません。
ここで randomSeed という命令を利用します。これは「乱数列の○けた目をスタート地点にする」という処理です。今回はけたの指定に analogRead(0) 、つまり0番ピンに流れる電流のアナログ値を利用しています。
0番ピンには何も接続していないため、電流の強さを指定していません。そのため、特に制御されたわけではないランダムな強さの電流が流れています。このアナログ値をけた数の指定に使えば、毎回乱数列のちがった場所から4文字を抜き取ることになり、実質ランダムな数値を得ることができます。

1.6. 自律モードをもっと改善

random(); を使用することで、ロボットの動作が予測できないものになり、少し自律型六脚ロボットらしくなってきましたね！
今日の仕上げとして、前進動作にも random(); を取り入れて、ロボットがフラフラと前進するようにしてみましょう。

ステップアップ

プログラム「HexCrawler2」を変更して、ロボットが前進しているときに以下のような動作をするようにしてみよう。

・1000分の1の確率で前進のかわりに右旋回、または左旋回を行う。
・右旋回や左旋回を行う時間は、1秒から3秒の間でランダムに決定する。

ヒント

左右タッチセンサーがOFFのとき、ロボットが行う動作は何通りあるか、それぞれどのようなときに行う動作か、よく考えてからプログラムの書きかえをしていこう！
「1から6の間のランダムな数」は random(1, 6); という文でつくれるよ！

講

解答例プログラムは下記です。
RoboticsProfessorCourse3>HexRobot5>HexCrawler3

ロボットが自分の意志で判断しているかのような偶発性を持たせています。人工知能（AI：artificial intelligence）の研究でも用いられている手法のひとつとして、ロボットが自分の判断で動いているかのような制御機能をプログラムしています（ファジィ理論）。
random( );内の数値を変えることによって、動きを変える確率がどのように変化するかを確認させてください。

1.7. random関数

振らせるサイコロの出目のパターン数を少なくしてやると、偶発的な旋回動作が発生するチャンスが増えます。自分好みにカスタマイズして、自分の六脚ロボットに個性をもたせましょう。他の人のロボットと一緒に動かして、動きの違いを比較してみましょう。
この乱数でサイコロをふらせて、ロボットの行動を決めるプログラムは、次回にも鍵になるので覚えておいて下さい。

コラム　情報社会と乱数

Arduinoに限らず、多くのコンピューターは乱数の生成に「乱数列」を利用しています。
たとえばある機器とある機器がネットワークを介して通信するとき、やり取りする情報が他の機器に読みとられるのを防ぐため「暗号化」することがありますが、この暗号をつくったり、暗号を解除したりするのにも乱数列がよく使われています。
もし、この乱数列が実はランダムではなく、何らかの傾向（たとえば偶数の方が登場しやすい、2の次は6になる確率が高いなど）があると、第三者が暗号を解除するときのヒントになってしまいます。そのため、よりランダム度の高い「高品質な」乱数の研究がさかんに行われています。

ところで、「延々とつづく数列で、0~9の数字が規則性なく並んでいる」と言われて「円周率」を想像した人もいるのではないでしょうか。
円周率を乱数列に使えば、「高品質な」乱数を難なく入手することができそうです。

しかし、ここに問題が隠れています。
小数点以下の数字が、特定の規則性や偏りを持たず延々と並んでいる数の事を「正規数」といいますが、実はまだ円周率が「正規数」かどうかはわかっていないのです。
もし、円周率が正規数であるときちんと証明されれば、これ以上ないほど高品質な乱数が得られたことになり、私たちの身の回りの情報機器はより強固なセキュリティを備えることができるはずです。

スーパーコンピューター（スパコン）が円周率を○兆けたまで計算した！　などといったニュースを聞くことがありますね。
円周率の計算そのものが、コンピューターの計算性能をはかるための試験（ベンチマークテストといいます）に向いているためでもあるのですが、円周率のけたをより多く求めること自体も、実は今後世の中全般で役立つ可能性があるのです。

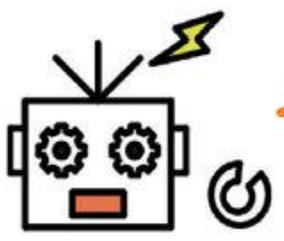
「円周率をそこまで細かく求めて何の役に立つんだ」「わざわざスパコンに円周率を計算させるなんて無駄だ」といった意見もありますが、いま役立っていない研究が将来的に欠かせないものになるというのはよくあることです。そういう意味では、今の視点だけで「無駄な研究」と断言して止めてしまうのは少しもったいないと思いませんか。

「ロボットづくり」という未来の技術を学ぶみなさんだからこそ、こういった考えも身につけておけるとよいですね。

2.　まとめ（目安5分）

今回は、六脚(ろっきゃく)ロボットを自律(じりつ)化させる方法を段階的に説明してきました。前回までの操縦(そうじゅう)するロボットと違い、改良を加えていくことで自分の考えで行動する、ちょっと生きているような感じが出てきたと思います。

次回は、六脚(ろっきゃく)ロボットの最終回ですが、踊りや腰をふるなどのモーションをロボットに組み込(こ)んで、自分だけの六脚(ろっきゃく)ロボットを完成させます。



次回は、ボタン1つでオリジナルモーションを再生するモーション生成だよ～！

《次回必要なもの》

今回使用したロボットと乾電池の予備の他に、以下を準備しておきましょう。

USB ケーブル 1	コントローラー 1	AC アダプター 1
		

図2-0　次回必要なもの

講

○　今回は六脚ロボットの「自律化」を行いました。
auto_timer での機能追加や random(); を使った疑似乱数の仕掛けがされていることが理解出来たか確認をしてください。

・　六脚ロボットの自律化
・　六脚ロボットの自律化のプログラム設計

○　次回は、「六脚ロボットのモーション生成」を行います。操縦機能、自律機能の他に、コントローラーのボタン1つでオリジナルモーションを加える方法について学習します。
歩行パターンでのサーボモーターの状態などの内容も大切になりますので、第2回、第3回のテキストの復習も推奨してください。