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6回

六脚(ろっきゃく)ロボットのモーション生成

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。

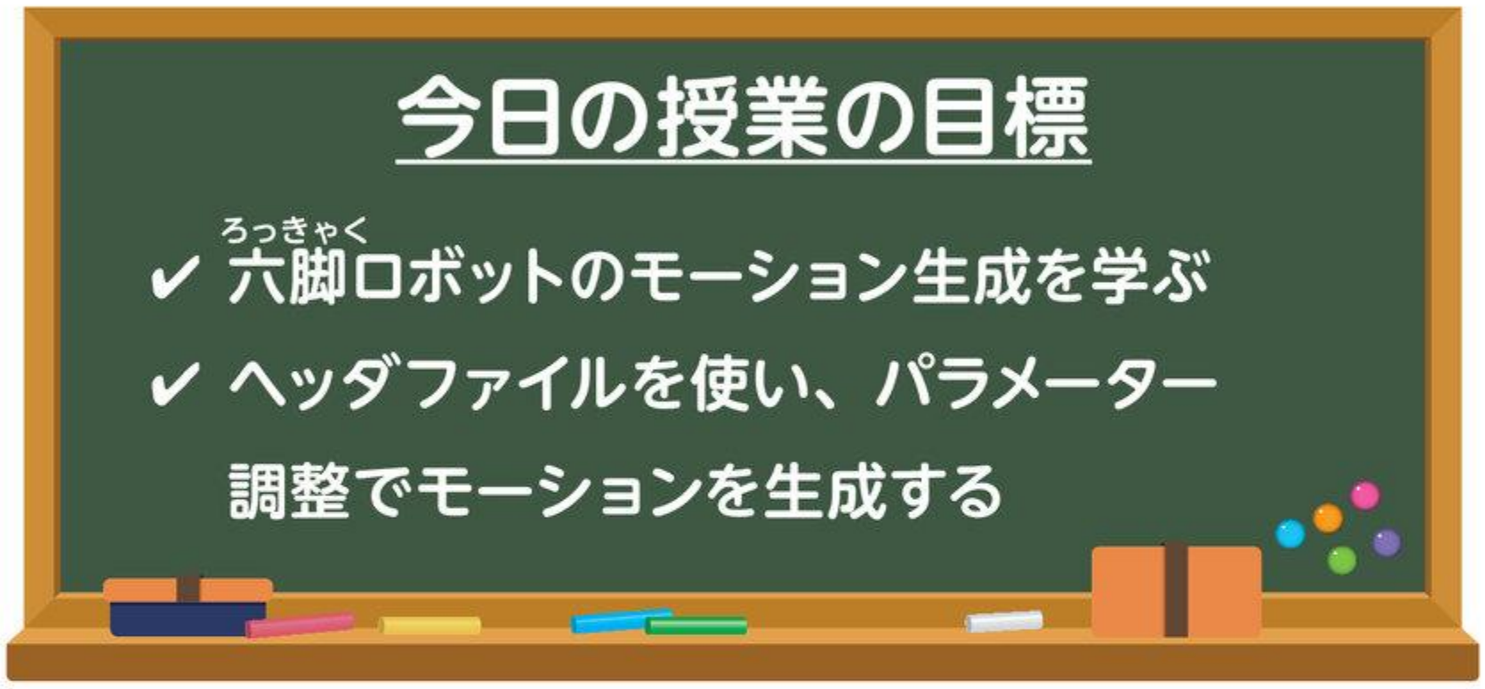
○　今回の目標をパネルで用意するか、黒板に予め書いておきます。

（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）

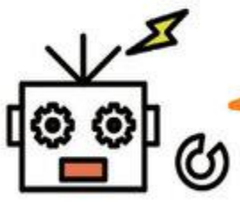
目安時間は授業時間 120 分のうち、休憩 10 分程度取ることを想定しています。
生徒の進捗状況により、休憩時間などを調整して授業を行ってください。

0. 六脚(ろっきゃく)ロボットのモーション生成（目安10分）

0.0. 「六脚(ろっきゃく)ロボットのモーション生成」でやること



今回の授業では、六脚ロボットの「モーション生成」に挑戦します。
前回は、六脚ロボットの２つのタッチセンサーを触角代わりに、自動的に障害物を回避する自律ロボット化を行いました。人間が操作しない状態でロボットが自動的に障害物を回避したり、方向転換する様子は、まるで生きているように見えませんでしたか？
ロボットのハードウェアは同じでも、プログラム中にあるパラメーターをかえることで、行動のパターンをかえることもできました。とはいえ、歩きまわっているだけでは、まだまだ動きが単調です。
そこで今回は、六脚ロボットのモーション生成の機能を追加します。モーション生成とは、ロボットが手を振ったり、ダンスを踊ったり、あらかじめデザインされたものを動作再生することです。モーション生成して動作再生する機能を自分の六脚ロボットに盛り込み、自分だけのオリジナルモーションを作って個性的な六脚ロボットを完成させましょう！



人間もロボットも「間の取り方」がダイジナノダ。

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0-0　必要なもの

1. 六脚ロボットの動作を設計する（目安90分）

1.0. モーションを考える

まずは、ロボットの動きを考える必要がありますね。これまでの授業で学んだ六脚ロボットの動作を少し振り返ると、全ての動きは、角、右脚、左脚、中脚についているサーボモーターの角度と時間を制御することで生成しました。新たに一連の動きを考えることを「モーションデザイン」といいます。それぞれのモーターの角度や動作時間を1コマ1コマ考え、データ化して、パラパラ漫画のようにつなぎあわせるようにつくります。

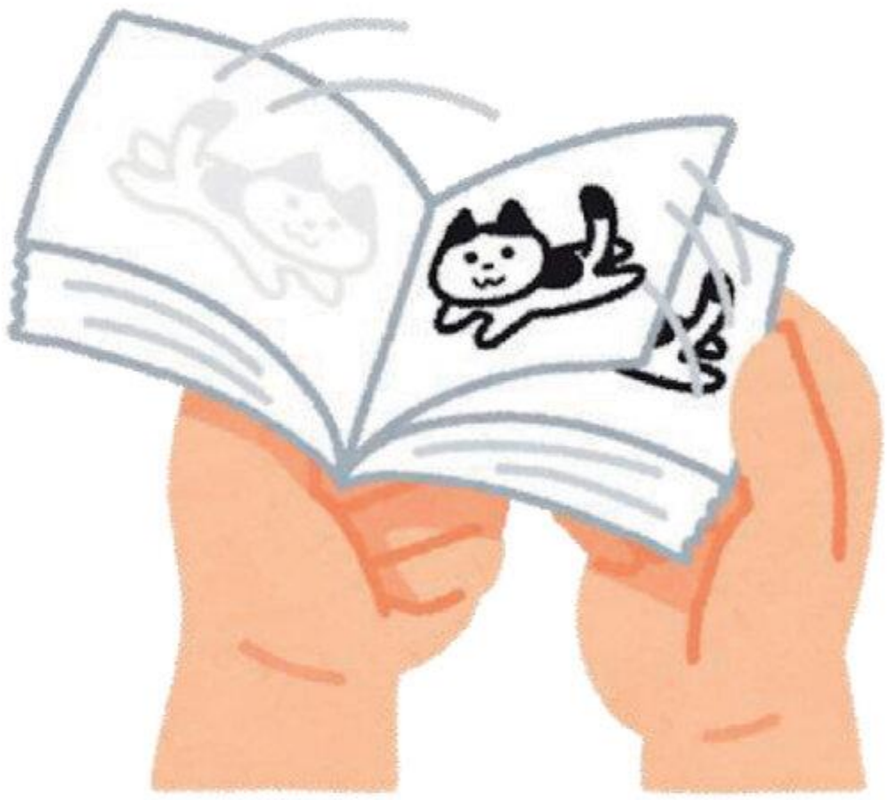


図1-0　モーションデザインのイメージ

POINT

モーションデザインとは、その名の通りロボットの動きをデザインすることです。前回行った「ロボットの自律化」では、ロボットの行動にタイマーを仕掛ける auto_timer や行き当たりばったりであたかも自分で判断しているかのように見せる random を使った乱数処理を作りました。こうした制御方法を考えるのもモーションデザインのひとつですが、今回はより細かくロボットのポーズをつくり、つなぎ合わせて一連の動作にしていきます。

1.1. モーションデータを再生する

では、モーションデザインしたことを実際にどのように生成していくかを理解するために、モーションの一例を実行してみましょう。
以下のプログラム「HexMotion0」では、ロボットのポーズを6つ組み合わせてモーションを生成しています。そして、1つのモーションをデータ化してコントローラーのボタンに紐づけています。
つまり、コントローラーの特定のボタンを押すと、モーションデータが再生されます。このプログラムの実行後、ロボットのボタン操作は**図1-1**のようになります。

プログラムの書き込み

RoboticsProfessorCourse3 > HexRobot6 > HexMotion0

実行結果：△ボタンを押すと、角を上下に振るモーションデータが再生されます。
□ボタンを押すと、左右の脚を交互に上げて腰を振るようなモーションデータが再生されます。
○ボタンと×ボタンにはデータを入れていないため、4秒間止まったままです。

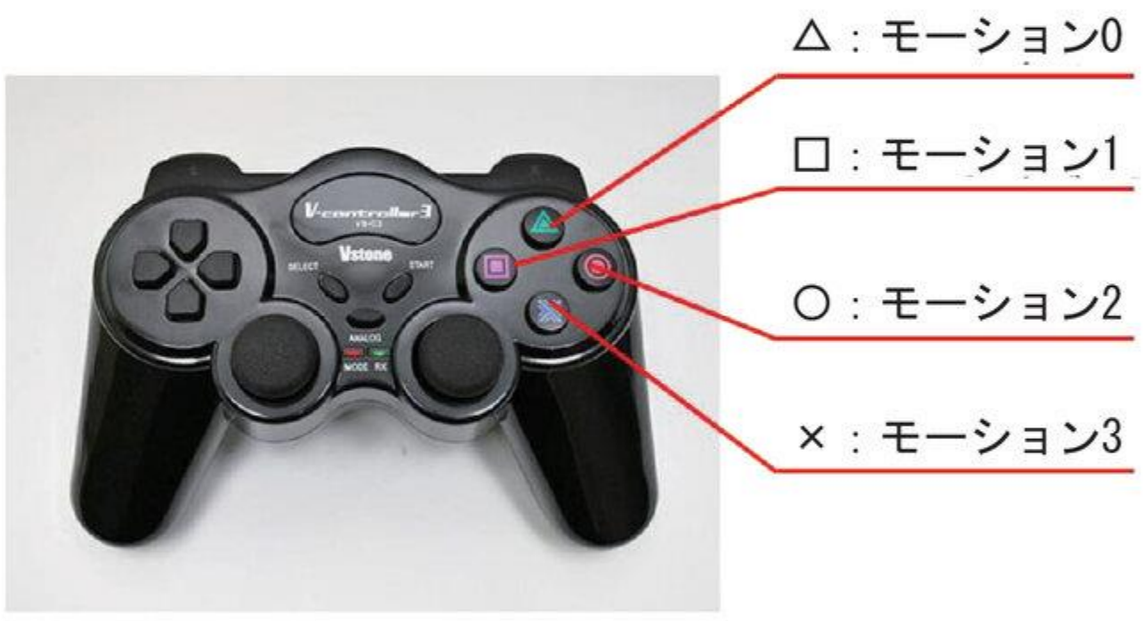


図1-1　プログラム「HexMotion0」のボタン操作

では、プログラム「HexMotion0」を見ていきましょう。

1.2. ヘッダーファイルを読み解く

プログラム「HexMotion0」はこれまでのプログラムと同様、「コントローラーのボタンの状態を確認する」「ボタンの状況に応じてサーボモーターの位置を制御する」という処理でできています。
コントローラーのボタンの状態の確認はタイマー割り込みを利用して行うため、setup()内でタイマー機能の宣言がされています。

プログラム「HexMotion0」より抜粋

```
ps2x.config_gamepad(13, 11, 10, 12);                    //コントローラーの初期化
MsTimer2::set(GAMEPAD_PERIOD, gamepad_update);  //タイマー割り込み機能を利用
(GAMEPAD_PERIODミリ秒ごとにgamepad_update()を実行するよう設定)

(中略)

MsTimer2::start();                                      //タイマー割り込みスタート
```

これで、動作中 GAMEPAD_PERIOD ミリ秒に1回のペースでコントローラーの状態を確認する処理 gamepad_update() が実行されるようになりました。

プログラム「HexMotion0」より抜粋

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

    //モーション実行ボタンの確認
    if (motion_flag == MOTION_NONE){
    //モーション実行中は他のモーションは実行できない
        if (ps2x.Button(PSB_TRIANGLE) == HIGH){   //もし△ボタンがおされたら
            motion_flag = MOTION_0;                         //モーション0を再生
            motion_frame_num = sizeof( motion_data0 ) / sizeof( motion_
            data0[0] );
            motiondata =  motion_data0;
            motion_frame = 0;
            motion_time = 0;
        }
(中略)
    }
}
```

たとえば PSB_TRIANGLE 、つまり△ボタンが押されていた場合、まずは motion_flag という変数を MOTION_0 という値にする、という処理が行われます。

では、いよいよサーボモーターの制御を担当する部分を見ていきましょう。

プログラム「HexMotion0」より抜粋

```
void loop(){
    if (motion_flag != MOTION_NONE){
//何らかのモーションの実行が指定されている場合


        if (motion_frame != motion_frame_num - 1){
//モーションのキーフレームが最後まで到達しているかどうか
            currentH = getJointPosition(
                       motiondata[motion_frame][IDX_HEAD],
                       motiondata[motion_frame + 1][IDX_HEAD],
                       motion_time,
                       motiondata[motion_frame][IDX_TIME]);

(中略)

        }
    }

    servoL.write(currentL);                    //現在位置をservoLに反映
    servoR.write(currentR);                    //現在位置をservoRに反映
    servoM.write(currentM);                    //現在位置をservoMに反映
    servoH.write(currentH);                    //現在位置をservoHに反映

    delay(SERVO_PERIOD);                       //SERVO_PERIODミリ秒待つ
}
```

次の目標位置にあわせて変数 currentH/L/R/M の値を変更し、その値の角度にあうようにサーボモーターを回転させる、という処理です。

変数 currentH/L/R/M の値の決定には motiondata[A][B] という文がたびたび登場していますね。これは「モーションデータ」という、色々なタイミングの各サーボモーターの位置をまとめて記したデータです。詳しくは後ろのページで解説します。

ここまで見てきて気づいた人もいるかもしれませんが、実はこのプログラムには数値を指定している部分がほとんどありません。タイマー割り込み機能も GAMEPAD_PERIOD ミリ秒に1回実行するとありますが、具体的に何ミリ秒かはどこにも書かれていませんね。加えて、サーボモーターの位置制御についても、具体的に何度回転させるかの指定が見当たりません。いつもなら H_BASE_POSITION という定数を90という値で定義しておくなどして回転角度を制御していたはずです。

画面の上部をよく見ると今回扱っている「HexMotion0」というプログラム名だけでなく、「HexMotionData.h」「HexParam.h」という2つの名前がありますね。
ためしに、「HexParam.h」を見てみましょう。

図1-2 「HexParam.h」を開く

角のサーボモーターの原点位置 H_BASE_POSITION の値は 55、タイマー割り込み機能を行う間隔 GAMEPAD_PERIOD の値は 100 など、先ほど見当たらなかった値の指定がまとめて書かれています。

また、「HexMotion0」のプログラム内にも、よく見るとこの「HexParam.h」の名前が登場しています。

プログラム「HexMotion0」より抜粋

```
#include "HexParam.h"        // 各サーボモーターの原点位置などの調整パラメーターデータ
#include "HexMotionData.h"   // 再生させるモーションデータ
```

#include はこれまでにも、様々な処理があらかじめ書かれたファイル（ライブラリ）を引用するときに使っていました。今回はその応用編です。

プログラム「HexMotion0」が保存されている場所（フォルダ）には、他に2つのファイルが保存されています。

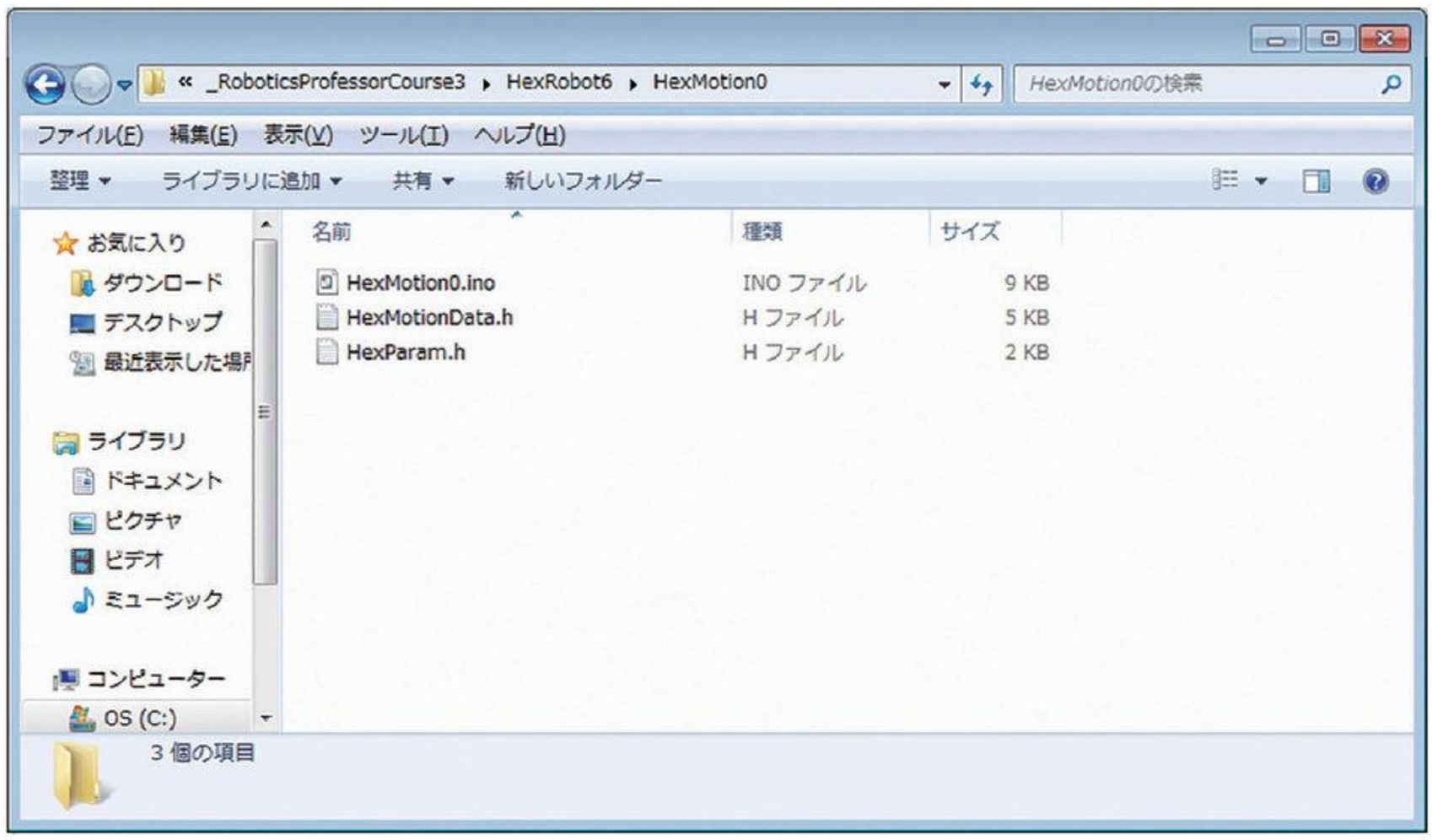


図1-3　「HexMotion0」の保存場所

「HexMotion0.ino」というのがプログラム「HexMotion0」のことです。つまり細かな処理を別のファイルに分けて書いておくことで、「HexMotion0」の中身を少なく、見やすくしようとしているのです。実際、前回扱ったプログラム「HexCrawler3」と比べると、「HexMotion0」は行の数が半分以下にまとめられています。

ちなみに #include をかくとき、今までのように、各種機能を使うためのライブラリを引用するときは 〈 〉 で、今回のように同じフォルダに保存した別ファイルを引用するときは "" でファイル名を囲みます。

POINT

ヘッダーファイル「HexParam.h」は、六脚ロボットの角、左右の脚、中脚の動力となるそれぞれのサーボモーターの原点位置の設定やコントローラーのボタンが押されたときの移動量と時間などの定義をしています。

1.3. モーションデータの書きかた

次に、「HexMotionData.h」のタブをクリックしてみましょう。このヘッダーファイルは、モーションの定義とモーションデータが記述されています。

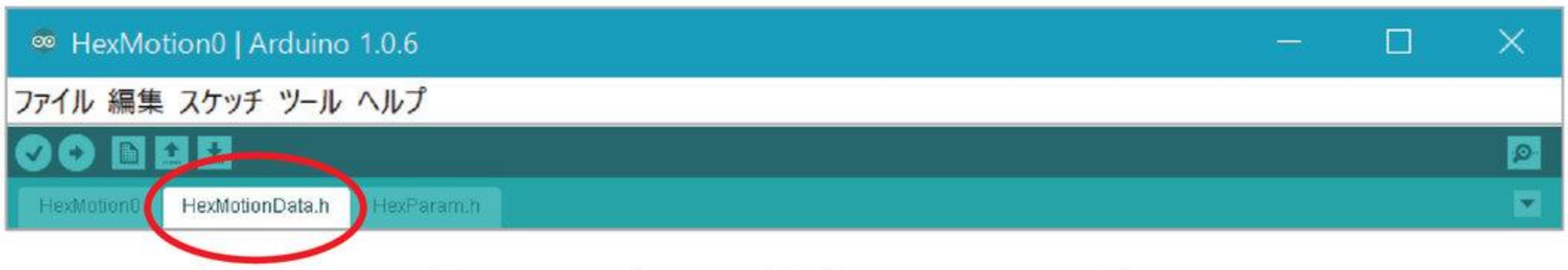


図1-4　ヘッダーファイル「HexMotionData.h」

プログラム「HexMotion0」/「HexMotionData.h」より抜粋（ばっすい）

```
// モーション0のモーションデータ
// 角を2回上下させる
int motion_data0[][5] = {
// Head , Left , Right, Mid  , Time
    {H_BASE_POSITION       , L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 500},
// ←モーションの最初は必ず原点位置から！！
    {H_TARGET_POSITION_UP  , L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 500},
    {H_TARGET_POSITION_DOWN, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 500},
    {H_TARGET_POSITION_UP  , L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 500},
    {H_TARGET_POSITION_DOWN, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 500},
    {H_BASE_POSITION       , L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION,   0}
// ←モーションの終わりも必ず原点位置！！　最後に「,」を付けない！
};
```

motion_date0（モーションデータ0）となっている部分が、コントローラーの△ボタンを押したときに動作再生されるモーションデータです。黄色の部分が、角を上下させる動作のうちモーターの動作が切り替わる瞬間を表します。この瞬間のことを「キーフレーム」と呼びます。キーフレームの番号は0から始まるので、モーションデータ0にはキーフレーム0からキーフレーム5の6つのキーフレームが存在することになります。キーフレームの書き方は以下の通りです。

{ 角のサーボの位置 , 左脚のサーボの位置 , 右脚のサーボの位置 , 中脚のサーボの位置 , 時間 }

最後の「時間」というのは次のキーフレームに到達するまでの時間です。キーフレーム0から、500ミリ秒かけてキーフレーム1の姿勢をめざすという事ですね。そのため、最後のキーフレーム（キーフレーム5）の時間は0となります。

1.4. オリジナルモーションを作ってみよう①

それでは、六脚ロボットのオリジナルモーションを作っていきましょう。まずは、ヘッダーファイル「HexMotionData.h」に書かれている「モーション0」、「モーション1」を読み解いて理解していきましょう。

1）モーション0

このモーションデータは、角を2回上下させる構成になっています。

各要素の関節位置は、それぞれ *_BASE_POSITION（原点位置）から開始されます。

また、角、左脚、右脚、中脚の原点位置は、「HexParam.h」で定義された角度に従います。

「モーション0」のプログラムを良く見ると、角の位置が以下のように遷移しています。

キーフレーム0：　H_BASE_POSITION（原点位置）
キーフレーム1：→H_TARGET_POSITION_UP（角を上げる）
キーフレーム2：→H_TARGET_POSITION_DOWN（角を下げる）
キーフレーム3：→H_TARGET_POSITION_UP（角を上げる）
キーフレーム4：→H_TARGET_POSITION_DOWN（角を下げる）
キーフレーム5：→H_BASE_POSITION（原点位置）

つまり、このモーションデータでは、角を「原点位置→上→下→上→下→原点位置」の順に動かすようにプログラムされています。加えて、各モーションの時間を500ミリ秒に設定しています。

プログラム「HexMotion0」/「HexMotionData.h」より抜粋

```
// モーション0のモーションデータ
// 角を2回上下させる
int motion_data0[][5] = {
  // Head , Left , Right, Mid  , Time
      {H_BASE_POSITION       , L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 500},
      {H_TARGET_POSITION_UP  , L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 500},
      {H_TARGET_POSITION_DOWN, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 500},
      {H_TARGET_POSITION_UP  , L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 500},
      {H_TARGET_POSITION_DOWN, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 500},
      {H_TARGET_POSITION_DOWN, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 500},
      {H_BASE_POSITION       , L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION,   0}
};
```

2) モーション1

このモーションデータは、腰を2回振る構成になっています。

さきほどと同様に、各要素の関節位置は、それぞれ *_BASE_POSITION（原点位置）から開始されます。角の制御との違いは、各脚の動きの自由度を上げるために角度を指定している点です。

脚の３つの関節はリンク機構で動いていますので、＋60 と指定しても実際に脚が60°動くわけでありません。60°動くのは脚を動かしているサーボモーター部分です。

各脚関節のサーボモーターの移動方向と調整値のプラスマイナスの関係は**図1-5**のようになります。

プログラム「HexMotion0」/「HexMotionData.h」より抜粋

```
// モーション1のモーションデータ
// 腰を2回振る
int motion_data1[][5] = {
    // Head , Left , Right, Mid  , Time
    {H_BASE_POSITION, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION     , 500},
    {H_BASE_POSITION, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION - 60, 500},
    {H_BASE_POSITION, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION + 60, 500},
    {H_BASE_POSITION, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION - 60, 500},
    {H_BASE_POSITION, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION + 60, 500},
    {H_BASE_POSITION, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION     ,   0}
};
```

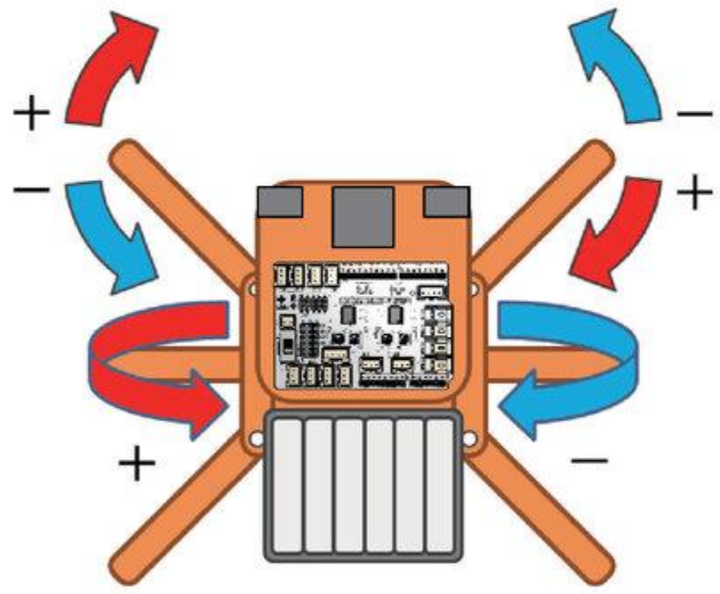


図1-5　各脚関節のサーボモーターの移動方向と調整値のプラスマイナス

では、モーションデータの構成がわかってきたところで、実際にプログラムを変更してモーションデータをつくってみましょう。

ステップアップ

ヘッダーファイル「HexMotionData.h」にある「モーション2」、「モーション3」のデータを書き直して、オリジナルモーションを作ってみよう。キーフレームの数はコピー＆ペーストで増やせるので、自分の好きな動きをさせてみよう。

ヒント

各関節の動き出しと終わりは原点位置だから、それ以外を変更してみよう。キーフレームは全部で５つ用意されているけれど、コピー＆ペーストで増やすことができるよ。下記のプログラム中の黄色の部分をコピーしよう。

プログラムができ上がったら、プログラム「HexMotion0」を実行してコントローラーで動かしてみましょう。でき上がったモーションの動作再生は、「モーション2」はコントローラーの○ボタン 、「モーション3」は×ボタンです。

プログラム「HexMotion0」/「HexMotionData.h」より抜粋

```
// モーション2のモーションデータ
// オリジナルのモーションを作って入力しよう！
int motion_data2[][5] = {
    // Head , Left , Right, Mid  , Time
    {H_BASE_POSITION, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 1000},
    {H_BASE_POSITION, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 1000},
    {H_BASE_POSITION, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 1000},
    {H_BASE_POSITION, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION, 1000},
    {H_BASE_POSITION, L_BASE_POSITION, R_BASE_POSITION, M_BASE_POSITION,    0}
};
```

モーションデータを書きかえ、思い通りの動きをつくれましたか？ うまくいかない場合は、モーションデータを修正してくり返し試してみましょう。もし、モーションデータの変更でエラーが出た場合は、プログラムの中身をチェックしましょう。

コラム　キーフレーム間の動作について

サーボモーターへの目標位置の更新は20ミリ秒間隔（かんかく）、つまり１秒間に50回のスピードで行われていますので、たとえば4秒のモーションを生成するのに200個（50回×4）ものコマを用意しなければいけません。しかし、これはとても大変な作業になるので、今のコマから次のコマまでどのくらいの時間をかけて動くのかを指定する **{時間}** を加え、プログラムのほうで20ミリ秒毎の目標位置を設定して作ります。そうすることで、動きが変わるキーになるコマ（キーフレーム）だけを人間がデザインして書けばモーションデザインができるようになります。これを図にすると、**図1-6**のようになります。

ここでは、簡単に一つの関節だけのモーションデータとキーフレーム、実際に再生されるモーションの各コマのデータを表しています。●がキーフレームデータ、○が各コマのデータになります。

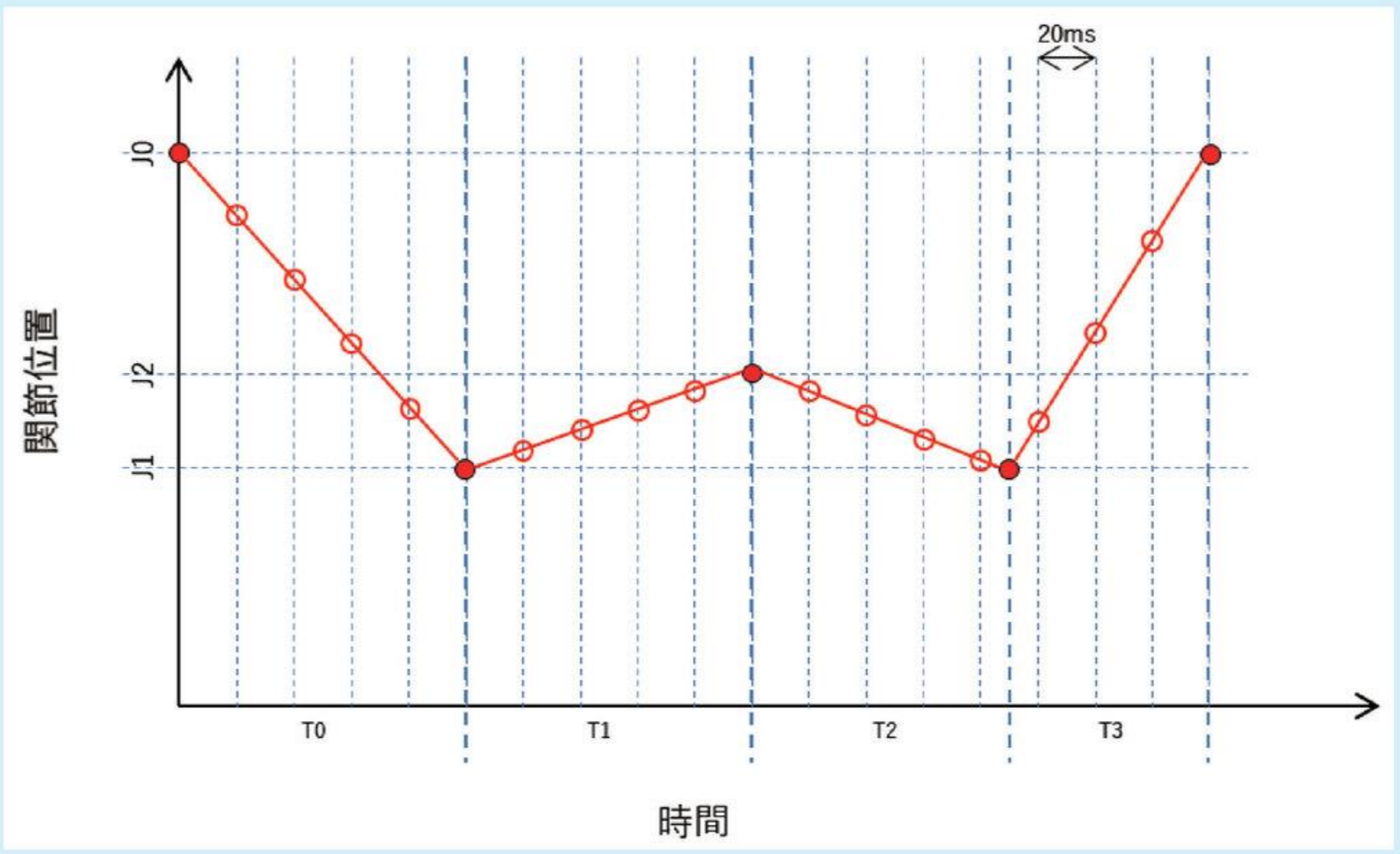


図1-6　キーフレームと時間

キーフレームは、関節位置と次のコマまでの時間のペアで表されます。

キーフレーム０：{J0, T0}、キーフレーム1：{J1, T1}、キーフレーム2：{J2, T2}、

キーフレーム 3：{J1, T3}、キーフレーム4：{J0, ?}

ただし、時間はキーフレームとキーフレームの間になるため、最後のキーフレームには時間のデータがありませんので注意して下さい。プログラム上ではデータ無しは都合がわるいため、０としてありますが処理（しょり）上では無視されます。六脚（ろっきゃく）ロボットでは、キーフレームのデータは以下のように、角、左脚（あし）、右脚（あし）、中脚（あし）、時間の５つの要素で構成され、このキーフレームが複数並べられることで一連のモーションデータになります。

```
{H_BASE_POSITION , L_BASE_POSITION , R_BASE_POSITION , M_BASE_POSITION , 500 },
```

コラム　モーションデータと二次元配列

六脚ロボットのキーフレームの要素は５つで固定ですが、モーションデータ中にあるキーフレームの数は作りたいモーションによってマチマチです。このモーションデータを扱うために、このプログラムでは二次元配列という方法を使います。「HexMotionData.h」の中で、「モーション0」に `int motion_data0[][5]` と記述されています。これは int型（Arduinoでは2byteで -32768 ～ +32767の整数を表現できる）の数字が５個でワンセットになったデータがさらに複数個並んでいるデータの塊ですよ、という意味になります。最初の[]がキーフレーム、２番目の[]が各キーフレームの要素を意味します。

たとえば、`motion_data0[2][4]` であれば、キーフレーム２(0から始まるので3)の４番目（0から始まるので5）の要素、各要素は｛角、左脚、右脚、中脚、時間｝ですから、要素の5番は時間になります。結果、`motion_data0[2][4]` は、「モーションデータ０」の「キーフレーム２」の時間を示していることになります。二次元配列については完全に理解しなくても良いので、使い方だけ確認しておきましょう。

豆知識

二次元配列

マイコンのメモリー上の連続した領域に、ずらりと並べて管理するのが配列です。
プログラムの先頭で型と配列名、縦(行)と横(列)の大きさの宣言を行います。

`_data1［2］［4］` であれば、下の表のオレンジ色のデータを指定します。

_data1[0][0]	_data1[0][1]	_data1[0][2]	_data1[0][3]	_data1[0][4]
_data1[1][0]	_data1[1][1]	_data1[1][2]	_data1[1][3]	_data1[1][4]
_data1[2][0]	_data1[2][1]	_data1[2][2]	_data1[2][3]	_data1[2][4]
_data1[3][0]	_data1[3][1]	_data1[3][2]	_data1[3][3]	_data1[3][4]

講

二次元配列の説明はここでは詳しく説明をしません。使い方のみ確認させてください。
なお、二次元配列では最初の[ ]の中の数字は省略できる（コンピューターが数えてくれる）ため、キーフレームの数をモーション毎に変えてもキーフレームの数を数えてプログラムに反映する必要はありません。

ここまでを整理すると、モーションデータを作成する上でのルールは、以下の６つになります。

POINT

1. 各キーフレームの要素の数は5つで、並びは｛角、左脚、右脚、中脚、時間｝
2. 各キーフレームの最後に「,（コンマ）」をつけ忘れない
3. キーフレームの数は幾つでも増やせる
4. 最後のキーフレームの時間は0にする（入れても使われない）
5. 最後のキーフレームの最後に「,（コンマ）」をつけない
6. 最初と最後のキーフレームは、必ず全てのサーボモーターの原点位置にする

図1-7は、プログラム「HexMotion0」の状態と動き（モーション）を図にしたものです。このプログラムでは、基本的に六脚ロボットは電源を入れた直後、setup()の処理で各関節を原点位置にします。その後は基本的に原点位置で静止していて、ボタン操作によって4つのモーションを再生します。これらのモーションは、必ず原点状態から出発して、モーションを終わったときには、必ず原点位置に戻ってくるようにしています。この状態を守っている限り、ロボットは不連続な動きをしません。これが６番目の約束事になります。
なお、このような図は、状態遷移図、もしくは、ステートマシン図と呼ばれ、ロボットの状態と処理がどのような関係で繋がっているのかをわかりやすく表すことができます。

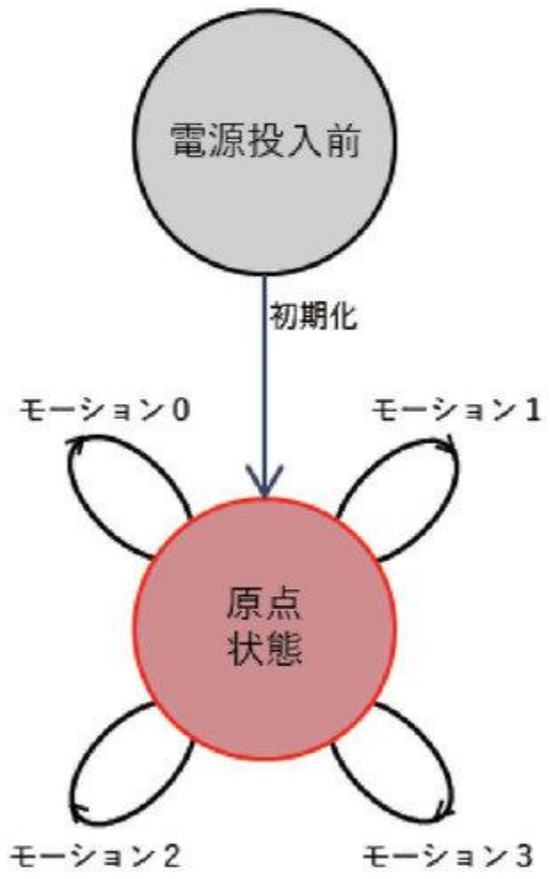


図1-7　モーションデータの開始と終わり

1.5. モーションデータに操縦モードを機能追加する

これで六脚ロボットのモーションを自由に生成できるようになりましたが、これだけでは、せっかく前回までにコントローラーで歩行させたり、自律行動をさせたりできていたのに大幅に機能が減ってしまっていますね。

ここからは、モーションの考え方を盛り込みながら、コントローラー操作をどのように動作再生と共存させていくかを考えていきましょう。

これまで扱ってきた「歩行」と、今回扱っている「モーション」を両方実装しようとしたとき、ネックになるのは歩行モードとモーションモードを切りかえる瞬間です。

前のページで説明したとおり、モーションは「原点で始まり、原点で終わる」が原則なので、歩行制御中の中でも特に「片方の脚が浮いているとき（遊脚相）」にはモーションモードに切りかえることはできません。

歩行モード中は前進・後退・左右旋回とも「片側を遊脚相にして動かす」「全脚接地」「もう片側を遊脚相にして動かす」「全脚接地」という流れで脚を制御しているので、全脚接地に戻ったときだけモーションモードへの切りかえを受け付けられるようにしましょう。

これを状態遷移図にすると、**図1-8**のようになります。

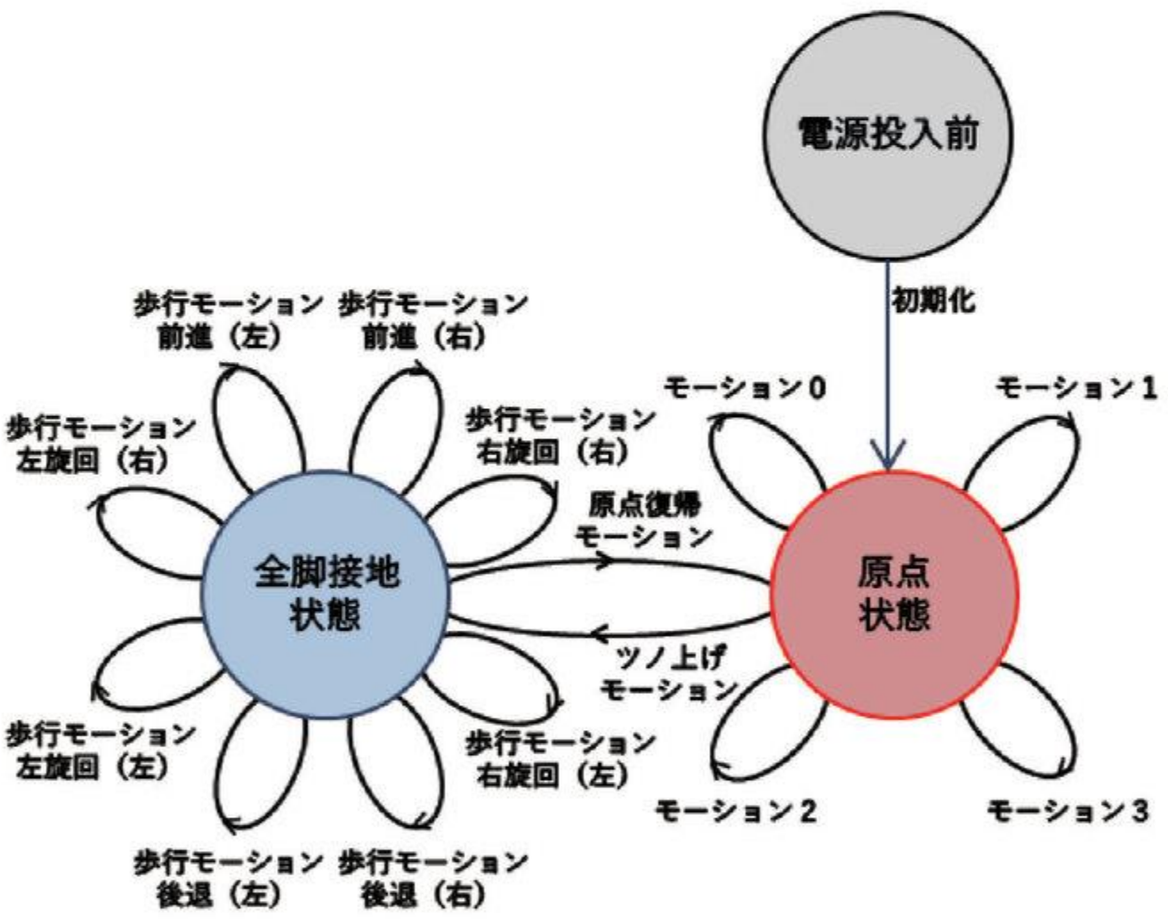


図1-8　モーションデータと歩行動作を組み合わせる

全脚接地状態から原点状態への遷移、つまり歩行モードの途中でモーションデータの再生に移りたいとき、左右の前後脚や角の位置は原点にあるとは限りません。そのため、遷移するときにいったんすべての脚・角を原点位置に戻すモーション（原点復帰モーション）を追加します。

また、逆にモーションデータを再生したあと歩行モードへ移りたいときは、モーションデータの再生終了時にすべての脚・角が原点に戻っているはずなので特別なモーションは必要ありません。ただ、このあと前回扱った「自律モード」を実装するので、前回と同様に角を上げるというモーションを追加しておきましょう。

表1-0　状態遷移表

ボタンの状態			△	□	○	×	↑	↓	←	→	―
状態	原点		0	1	2	3	ツノ上げ	ツノ上げ	ツノ上げ	ツノ上げ	なし
	全脚接地	右	原点復帰（右）	原点復帰（右）	原点復帰（右）	原点復帰（右）	前進（右）	後退（右）	左旋回（右）	右旋回（右）	原点復帰（右）
		左	原点復帰（左）	原点復帰（左）	原点復帰（左）	原点復帰（左）	前進（左）	後退（左）	左旋回（左）	右旋回（左）	原点復帰（左）

この状態遷移表では、状態遷移図には書かれていなかった全脚接地状態と、さらに下位の状態、左右が追記されています。これは歩行モーションを開始するときに、次に上げる脚が　「右側なのか左側なのか？」を内部で管理するためです。

この状態遷移表をもとにしたのが、プログラム「HexMotion1」です。

プログラムの書き込み

RoboticsProfessorCourse3 > HexRobot6 > HexMotion1

なお、このプログラムをロボットに書き込んで実行する前に、プログラム「HexMotion0」で調整した「HexParam.h」と「HexMotionData.h」を「HexMotion1」に移植しましょう。

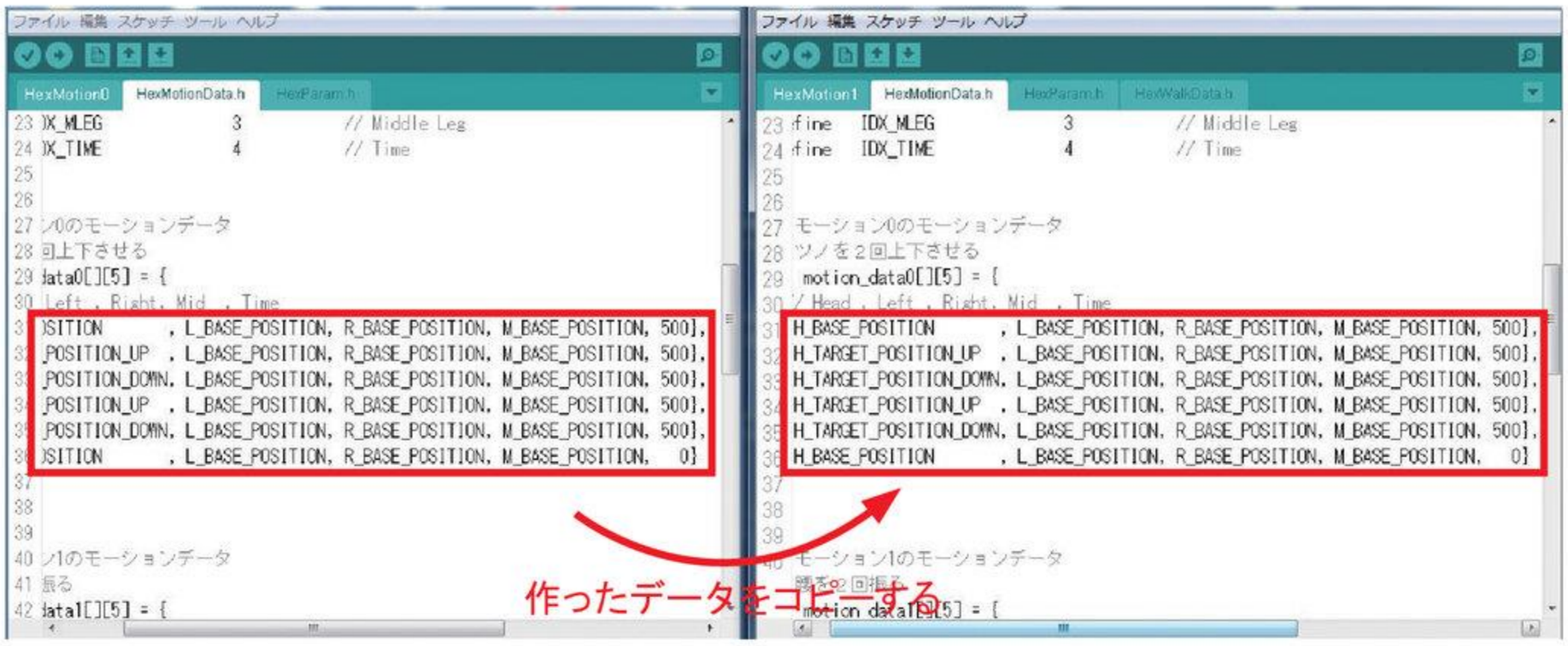


図 1-9　オリジナルモーションの移植

プログラムを実行してロボットを動かしてみましょう。

プログラムの書き込み

RoboticsProfessorCourse3 > HexRobot6 > HexMotion1

実行結果：オリジナルモーションの動作再生に加えて、コントローラーの十字ボタンで歩行操作ができるようになります。

図1-10　プログラム「HexMotion1」のボタン操作

さて、プログラム「HexMotion1」の基本構造は、ここまで説明してきた通りですが、おおまかにフローチャートにすると、**図1-11**のようになります。
「歩行操作」と「動作再生」を組み合わせるときに、各ボタンが押されたときに設定に応じてサーボモーターの位置を計算し直す、servo_update() と、**表1-0**の状態遷移表をプログラムに反映させた state_manager() が加わりました。
プログラム「HexMotion0」から追加された全脚接地状態とそこに繋がる各モーションは、新たにヘッダーファイル「HexWalkData.h」を作りまとめました。たとえば、歩行速度の調整をしたい場合などは、このヘッダーファイルの歩行モーションデータを変更するだけで他のプログラムに影響を与えることなくできます。

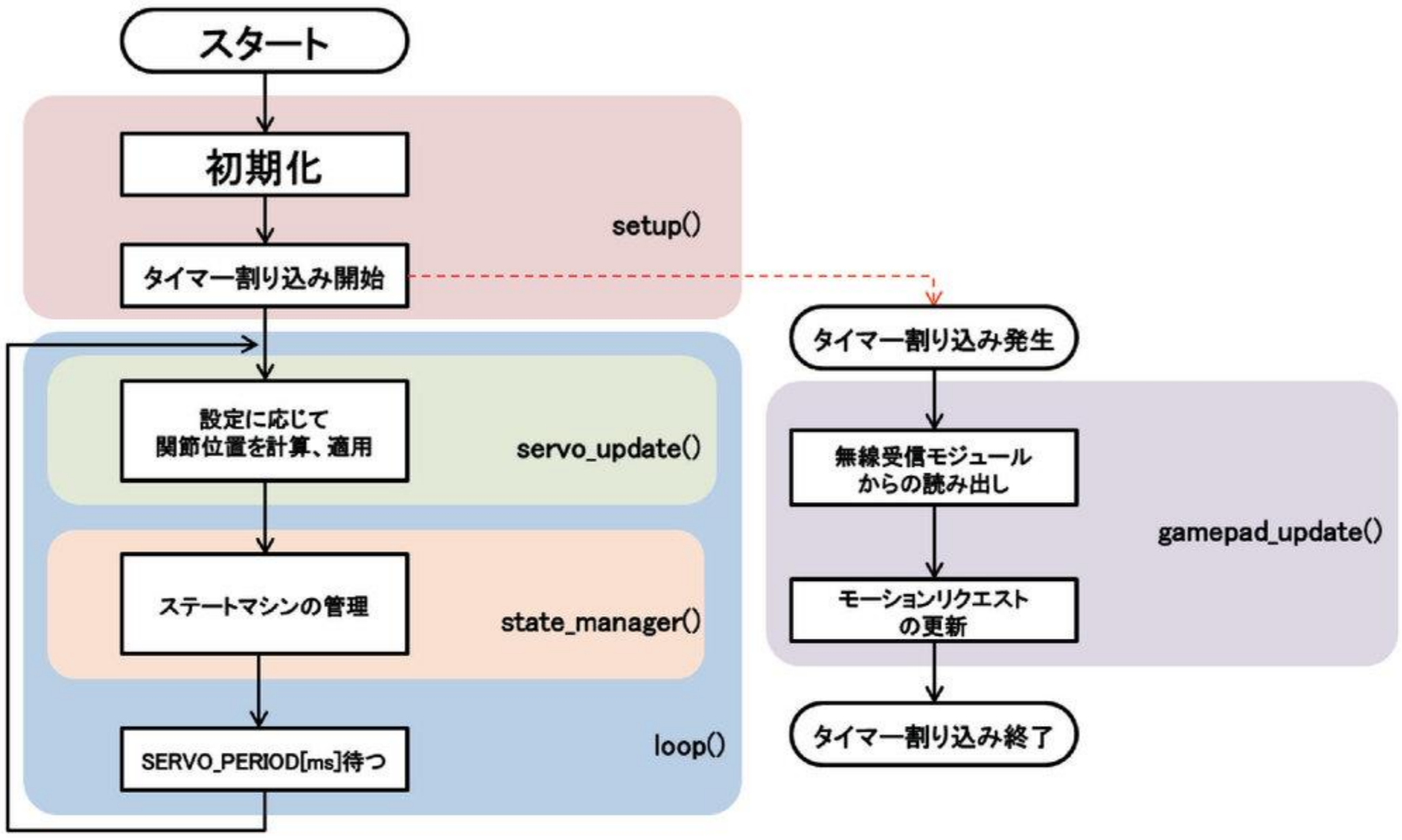


図1-11　プログラム「HexMotion1」のフローチャート

「HexMotion0」での動作再生とは異なり、歩行モーションは中脚が原点位置にある以外、他の脚と角の位置は固定ではないので、以下のようなプログラムを作ります。

プログラム「HexMotion1」/「HexWalkData.h」より抜粋

```
#define        POSITION_ANY        -32768        // 未決定の関節位置（その時の現在値を使う）
```

キーフレーム内で POSITION_ANY と書かれた部分は位置を変更せず、キーフレーム開始時の位置をそのまま維持するようになっています。

プログラム「HexMotion1」より抜粋

```
// モーション動作をするための関節位置を計算する関数
// int spos : 始まりの関節位置
// int epos : 終わりの関節位置
// int bpos : バッファされている関節位置
// int ctm  : 経過時間
// int ttm  : フレーム時間
int   getJointPosition(int spos, int epos, int bpos, int ctm, int ttm){
      int pos;

      // 始点と終点が 未定義（POSITION_ANY)のときはバッファの関節位置を使う
      if (spos == POSITION_ANY) spos = bpos;
      if (epos == POSITION_ANY) epos = bpos;
```

※バッファとはデータを保存する領域を意味します。「前回サーボモーターを動かしたとき、どの位置で停止したか」を一時的に保存しておいて、POSITION_ANY が使われた部分には保存しておいた前回の位置の値が代入されるのです。

通常、サーボモーターへの駆動指示値は　0 ～ 180（度）の範囲でしか受け付けられません。この範囲外の数値を指示されても、モーターは動作できません。六脚ロボットが暴走することを防止するため、あえて範囲外の数値を入れているわけです。大幅に数値の大きさが違えば、暴走する心配がないので、プログラムのテクニックと思ってください。

1.6. 「自律モード」も機能追加する

ここまでに、六脚ロボットで、歩行操作とモーション再生を両立できるようになりました。そこで、前回の内容を思い出して、自律行動も盛り込んでみましょう。

ステップアップ

プログラム「HexMotion1」を書きかえ、タッチセンサーを使った自律行動機能を追加してみよう！
できたら、前回扱ったようなランダム要素も盛り込んでみよう！

ヒント
自律行動が追加された後のコントローラー操作は、**図 1-12** のようになるね！

図1-12　自律行動が追加された後のコントローラー

講

解答例は以下のプログラムです。
RoboticsProfessorCourse3 > HexRobot6 > HexMotion2

「HexMotion2」には乱数を利用したランダム要素も追加してありますが、前回と異なり旋回時間がランダムになるだけでなく、一定確率でモーション0 ～ 3が再生されるようになっています。
発生確率はヘッダーファイル「HexAction.h」を書きかえることで調整可能です。

1.7. オリジナルモーションを作ってみよう②

チャレンジ課題

ヘッダーファイル「HexAction.h」のサイコロの出目に関わるパラメーターを調整して、自分の六脚(ろっきゃく)ロボットの性格を作り上げて完成させよう。他のヘッダーファイルのパラメーターも変更して、歩くスピードや脚の動き幅を変更してもいいよ。自分の六脚(ろっきゃく)ロボットを自分の好きなようにカスタマイズしよう。

さきほどの6つのルールを守って、自由に六脚(ろっきゃく)ロボットに自分の好きな動きを振りつけてください。各関節の位置は、それぞれのサーボモーターの原点位置（*_BASE_POSITION）からプラスマイナスして書きかえるとわかりやすいです。

図1-13　ヘッダーファイル「HexAction.h」

1.8. 使用したプログラムを一覧にして整理してみよう

ここでは、今回の授業で使ったプログラムの整理をしましょう。

メインプログラム	
HexMotion0	六脚ロボットでモーション生成をするプログラム
HexMotion1	六脚ロボットでモーションと歩行をするプログラム１
HexMotion2	六脚ロボットでモーションと歩行をするプログラム２
ヘッダファイル	
HexMotionData.h	六脚ロボットのモーションデータ
HexParam.h	六脚ロボットの原点位置調整パラメーターなど
HexWalkData.h	六脚ロボットの歩行モーションデータ
HexAction.h	六脚ロボットの自律モードの偶発パラメータなど

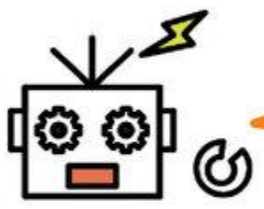
図1-14　プログラム一覧

豆知識

実際のロボット開発現場では、1つのプログラムのソースコードが1つのファイルから作られていることは、ほとんどありません。ソースコードが複数のファイルで構成されるとき、別のファイルにあるモーションやコマンドを参照したり、関数を呼び出したりするためには、別々のファイルをつくるととても便利だからです。
たとえばプログラムを「ソースファイル」「ヘッダーファイル」で構成します。機能単位で別々のファイルにして､それぞれのプログラムについて動作確認をしたり、複数の人が分担することができたりした方が開発しやすいからです。
実行プログラムはソースファイルが1つあれば作れますが、1つのファイルに何千行、何万行とかくよりも、ある程度のまとまりで分けてかくことで、誰が見てもわかりやすく、間違いがあったときに探しやすいものになります。
そして、後々機能を追加したい場合でもヘッダーファイルを修正することでまかなうことができます。
今回の六脚ロボットのように、ボタン1つでオリジナルモーションを再生させたり、自律モードに切りかえたりするロボットでは、このようにそれぞれの機能によってファイルを分けることが有効なのです。

2．まとめ（目安５分）

今回は、六脚(ろっきゃく)ロボットでモーション生成をする方法を考えました。そして、これまで学んできた歩行動作や自律行動のプログラミングをモーション生成と組合せ、六脚(ろっきゃく)ロボットのプログラムを完成させました。動作順序のプログラムは、かなり入り組んで見えるため理解が難しいかもしれません。
しかし、この授業では、ロボットを動かす機能を徐々につくり上げて、つけ足していく手順を重視して説明しました。いきなり完成されたプログラムをかき上げることは、上級者でも簡単ではありません。ですから、フローチャートや状態遷移(じょうたいせんい)図など、自分が作ろうとしているプログラムを図にして目で見て考えることが大切です。
プログラムとは状態遷移(じょうたいせんい)表を置きかえたものです。状態遷移(じょうたいせんい)図、状態遷移(じょうたいせんい)表という考え方を理解していれば、プログラミングを経験していく中で自然と自分で書き出すことができるようになります。



便利な機能をつくるには整理するのがダイジだね。
次回は不思議アイテムで液晶パネルを使うよ～！

《次回必要なもの》

USB ケーブル 1	マイコンボード 1	姿勢検出シールド 1	液晶ディスプレイシールド 1
			

図２-0　次回必要なもの

講

- ○ 第6回はモーション生成を行い、コントローラーのボタン1つで動作を実行する方法を学習しました。ロボットを多機能にするためには、プログラムを分けるヘッダーファイルの有効性を理解できたか確認をしてください。
- ○ 次回は、「不思議アイテムⅢ-２」液晶パネルの制御について学習します。