6-2.Arduino

Arduino では、車体の制御、音声出力、赤外線センサ処理、接触センサ処理、PIC への発光パターン及び発光タイミングの指定、制限時間の管理、被弾後の無敵時間の設定を担当している。以下に、それぞれの処理を実現するプログラムについての解説を示す。

6-2-1.Arduino(プログラム)

Arduino が電源を供給されて実行する関数には、setup()と loop()の二つがある。setup()は Arduino が起動してから一度だけ呼び出される関数であり、デジタルピンの入出力設定等、繰り返す必要のない処理を実行する際に用いられる。loop()は Arduino が起動している間、繰り返し呼び出される関数であり、プログラムのメインとなる関数である。
以降に、setup()と loop()内の記述に関する解説を示す。

＜setup()内の記述＞

setup()には、各種デジタルピンの入出力設定と、オープニング用 BGM を再生する処理を記述した。以下に実際のプログラムを示す。

・void setup()

```
void setup(){

  pinMode(PINNO,OUTPUT);//圧電スピーカー

  /*--------接触センサ(内部プルアップ抵抗を使用)------*/
  pinMode(FL,INPUT_PULLUP);
  pinMode(FR,INPUT_PULLUP);
  pinMode(BL,INPUT_PULLUP);
  pinMode(BR,INPUT_PULLUP);
  /*-----------------------------------------------------*/

  /*--------PICとの通信------*/
  pinMode(RA0,OUTPUT);
  pinMode(RA1,OUTPUT);
  pinMode(RA2,INPUT);
  pinMode(RB0,OUTPUT);
  /*------------------------*/

  sound(0);//OPサウンド

}
```

まず、以上のプログラム中に記述されている Arduino 言語関数について解説する。
pinMode();は指定したデジタルピンの入出力モードを設定する関数である。例えば、デジタル 1 番ピンを入力モードに設定するには、pinMode(1,INPUT);と記述する。

次に、プログラム中の定数について解説する。

PINNO は圧電スピーカーと接続されたピンの番号に対応している。

FL,FR,BL,BR はそれぞれ、前面左,前面右,背面左,背面右に設置されたスイッチに接続されているピンの番号に対応している。スイッチ回路のプルアップ抵抗は Arduino 内部のものを用いるので、ピン FL,FR,BL,BR は INPUT_PULLUP モードに設定した。

RA0,RA1,RA2,RB0 は PIC との通信に用いられるピン番号に対応している。RA0,RA1 の 2 本のピンを 2 ビットの信号として用い、PIC への通信パターンを指定している。

表.4 PIC 対 Arduino の通信プロトコル表

RA1	RA0	パターン
0	0	被弾
0	1	タイムオーバー
1	0	接触
1	1	無敵時間

RA2 は PIC から Arduino に HP が 0 の時に High を送ってもらうためのピンである。RB0 は Arduino から PIC へ、通信パターンの指定が完了したことを伝えるためのピンである。このピンが HIGH になってから初めて、Arduino から PIC への通信が開始される。

次に、記述中の関数について解説する。

sound(int type)は、各種 BGM を再生する関数である。引数に与えられた整数で、再生する BGM を指定する。sound(0)はオープニング BGM、sound(1)はゲームクリア BGM、sound(2)はタイムオーバーBGM に対応している。

sound(int type)は、以下のように定義した。

・**void sound(int type)**

```
void sound(int type){
  switch(type){
  case 0:
    tone(PINNO,330,200) ;  // ミ
    delay(200) ;
    tone(PINNO,330,200) ;  // ミ
    delay(400) ;
    tone(PINNO,330,200) ;  // ミ
    delay(400) ;
    tone(PINNO,262,200) ;  // ド
    delay(200) ;
    tone(PINNO,330,200) ;  // ミ
    delay(400) ;
    tone(PINNO,392,200) ;  // ソ
    delay(800) ;
    tone(PINNO,196,200) ;  // ソ
    delay(1000) ;
    break;
  case 1:
    tone(PINNO,523,200) ;  //ド
    delay(200) ;
    tone(PINNO,587,200) ;  // レ
    delay(200) ;
    tone(PINNO,659,200) ;  // ミ
    delay(200) ;
    tone(PINNO,698,200) ;  // ファ
    delay(400) ;
    tone(PINNO,784,200) ;  // ソ
    delay(200) ;
    tone(PINNO,880,200) ;  // ラ
    delay(200) ;
    tone(PINNO,987,200) ;  // シ
    delay(200) ;
    tone(PINNO,1047,200) ;  // ド
    delay(750) ;
    tone(PINNO,523,200) ;  // ド
    delay(200) ;
    break;
```

```
  case 2:
    tone(PINNO,494,200) ;  //シ
    delay(200) ;
    tone(PINNO,698,200) ;  // フ
    delay(400) ;
    tone(PINNO,698,200) ;  // フ
    delay(200) ;
    tone(PINNO,698,200) ;  // フ
    delay(200) ;
    tone(PINNO,659,200) ;  // ミ
    delay(200) ;
    tone(PINNO,587,200) ;  // レ
    delay(200) ;
    tone(PINNO,523,200) ;  // ド
    delay(200) ;
    tone(PINNO,330,200) ;  // ミ
    delay(400) ;
    tone(PINNO,330,200) ;  // ミ
    delay(200) ;
    tone(PINNO,262,200) ;  // ド
    delay(200) ;
    break;
  }

}
```

関数中の Arduino 言語関数について解説する。

tone(p,f,d)は p 番のデジタルピンから周波数 f の矩形波を d ミリ秒出力する Arduino 言語関数である。

delay(n)は n ミリ秒間、プログラムの実行を停止させる Arduino 言語関数である。

以上より、この関数のアルゴリズムは、tone 関数で音を出力し、delay 関数で次の音が再生されるまでの間を作りだしているものであると表現できる。

＜loop()内の記述＞

プログラムのメインとなる loop()内のアルゴリズムは、以下のフローチャートにより表現できる。

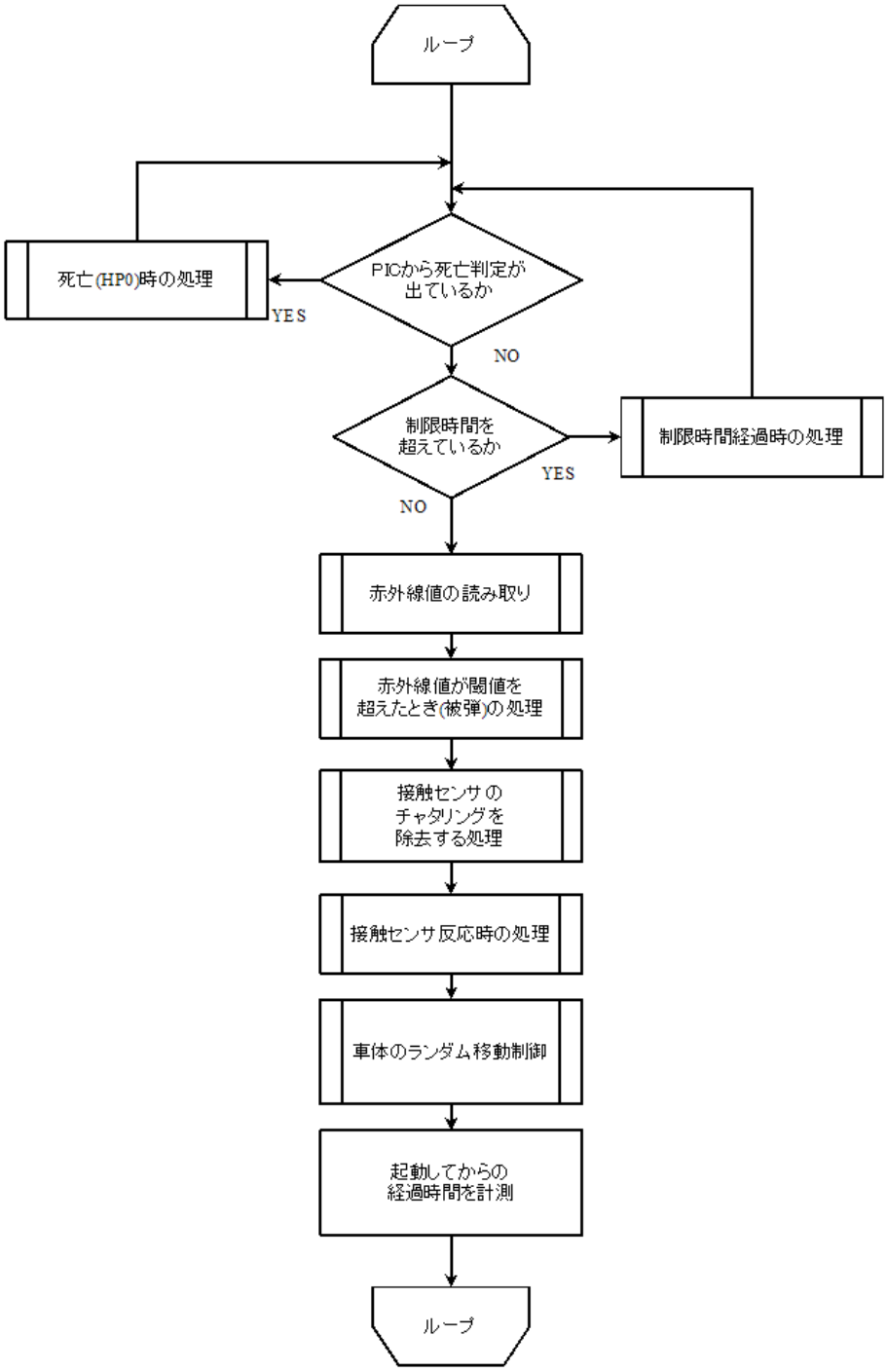

図 6.2 loop 関数フローチャート

以上のフローチャートを元に、実際にプログラムとして loop()内に記述したものを以下に示す。

・void loop()

```
void loop(){

  if(!digitalRead(RA2)){//生存時

    if(limit_time > game_limit){//タイムリミット圏内

      ledSetup();         //赤外線センサの値を読み取り
      ledResp();          //赤外線センサ反応時の処理
      bounceUpdater();    //接触センサチャタリング除去
      swResp();           //接触センサ反応時の処理
      tankControl();      //車体制御

      game_limit = millis();//起動してからの時間を計測

    }
    else{//タイムオーバー
      timeOver();//タイムオーバー時処理
    }
  }
  else{//死亡時
    deadAct();//死亡時処理
  }

}
```

まず、以上のプログラム中に記述されている Arduino 言語関数について解説する。

digitalRead()は、引数に対応する番号のデジタルピンの High/Low 状態を返す Arduino 言語の関数である。記述中の RA2 は、HP が 0 になったことを PIC から受信するためのピンの番号に対応している。したがって、!digitalRead(RA2)が真のとき、HP は 0 でないことを意味する。

millis()は、Arduino を起動してからカウントされた時間をミリ秒単位で取得する Arduino 言語の関数である。

次に、プログラム中の変数及び定数について解説する。

limit_time は制限時間に対応した定数である。制限時間は 60 秒なので、limit_time は 60000 として定義している。

game_limit は millis()で返された値、つまりゲーム開始から経過した時間を代入する変数である。初期値は 0 で初期化する。

次に、記述中の関数についてフローチャートと対応付けて解説する。
ledSetup()は「赤外線値の読み取り」の処理に対応した関数である。
ledResp()は「被弾時の処理」に対応した関数である。
bounceUpdater()は「接触センサのチャタリングを除去する処理」に対応した関数である。
swResp()は「接触センサ反応時の処理」に対応した関数である。
tankControl()は「車体のランダム移動制御」に対応した関数である。
timeOver()は「制限時間経過時の処理」に対応した関数である。
deadAct()は「死亡時の処理」に対応した関数である。
それぞれの関数の詳細については、後述の章で解説する。

6-2-2.Arduino(処理関数の説明)

loop()内で実行している関数それぞれについての解説を以降に示す。

＜ledSetup()について＞

ledSetup()は各方向の赤外線センサに接続されているアナログピン 0～3 番の電圧値を読み取る関数である。被弾している否かを判断するための電圧値の閾値の決定もこの関数で行う。

ledSetup()は以下のように定義した。

・ledSetup()

```
void ledSetup(){//赤外線センサの値読み取り
  indata0 = analogRead(0);
  indata1 = analogRead(1);
  indata2 = analogRead(2);
  indata3 = analogRead(3);

  static int border = (indata0 + indata1 + indata2 + indata3)/4;//各方向の平均値
  borderCopy = border;
}
```

まず、関数中の Arduino 言語関数について解説する。

analogRead(n)は、引数 n に対応したアナログ n 番ピンに印加されている電圧を分解能 10bit の整数で取得する Arduino 言語関数である。

次に、関数中に記述されている変数について解説する。

indata0,indata1,indata2,indata3 は、それぞれ前面、背面、右側面、左側面の赤外線センサからの値を格納するための整数型変数である。

border は被弾しているか否かを判断するための赤外線値の閾値を格納する、静的な整数型(static int)変数である。閾値は Arduino 起動直後の周囲の赤外線値を基準とするため、静的属性を変数に付与した。閾値は、各方向の赤外線センサの値の平均値（ (indata0 + indata1 + indata2 + indata3)/4 ）とした。

borderCopy は、border の値を他のスコープで用いるため、border の値を格納する整数型変数である。今回、閾値を動的に変更する処理は行わなかったが、閾値を変更・加工する必要がある場合は、この変数の値を操作することにより元の閾値を保持したまま閾値の加工を行うことができる。

＜ledResp()について＞

ledResp()は、赤外線銃の被弾処理を行う関数である。ledResp()のアルゴリズムは以下のフローチャートにより表現できる。

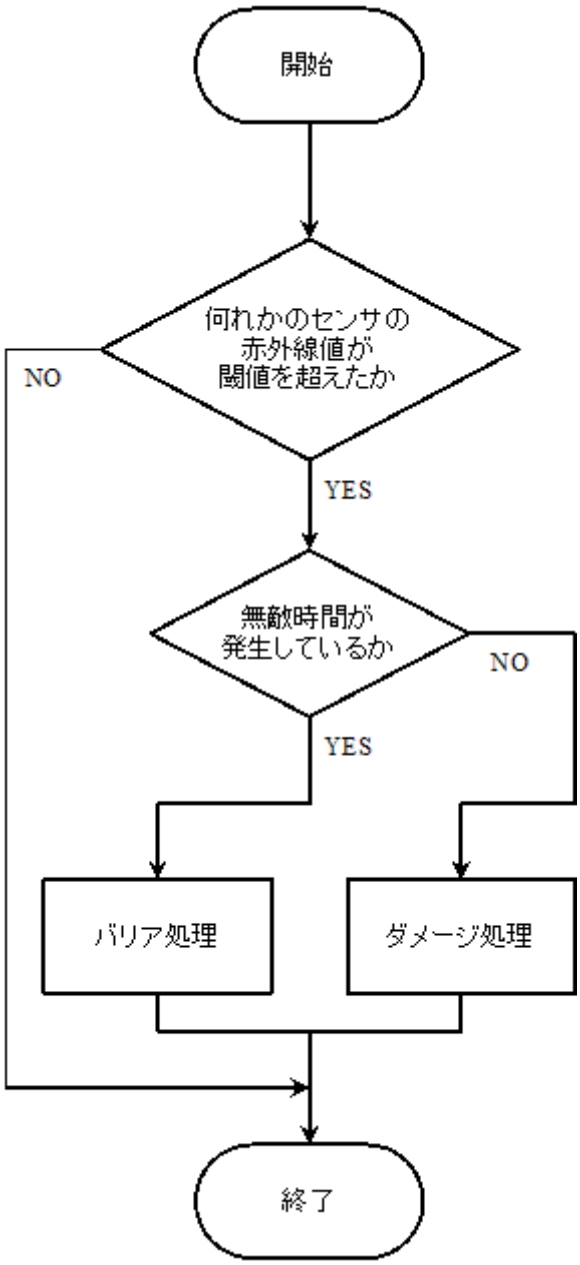

図 6.3 ledResp 関数フローチャート

以上のフローチャートを元に、実際にプログラムとして定義した ledResp()は以下のようになる。

・ledResp()

```
void ledResp(){//赤外センサON時の処理

  static unsigned long IRtime_old = 0;
  unsigned long IRtime_now = millis();


  if((indata0 - borderCopy) > upperThan ||
     (indata1 - borderCopy) > upperThan ||
     (indata2 - borderCopy) > upperThan ||
     (indata3 - borderCopy) > upperThan ){
      if((IRtime_now - IRtime_old) > barrier_time){
          tone(PINNO,392,200) ;  // ソ
          delay(200) ;
        /*------PICとの通信(ダメージ)------*/
        digitalWrite(RA0,LOW);
        digitalWrite(RA1,LOW);
        digitalWrite(RB0,HIGH);
        /*-----------------------------*/
        IRtime_old = IRtime_now;
      }
      else
      {
        tone(PINNO,262,200) ;  // ド
        delay(200) ;
        /*------PICとの通信(バリア表現)------*/
        digitalWrite(RA0,HIGH);
        digitalWrite(RA1,HIGH);
        digitalWrite(RB0,HIGH);
        /*---------------------------------*/
        IRtime_old = IRtime_now;
       }
  }
  /*---ピン出力リセット---*/
  digitalWrite(RB0,LOW);
  digitalWrite(RA0,LOW);
  digitalWrite(RA1,LOW);
  /*--------------------*/
}
```

まず、関数中の Arduino 言語関数について解説する。
digitalWrite()は引数で指定したデジタル出力ピンの HIGH/LOW 状態を変更する関数である。

次に、関数中に記述されている変数及び定数について解説する。
upperThan は、現在の赤外線値と閾値を比較した際、赤外線値が閾値をどの程度上回ったときに、被弾したと判断するかを決めるための、定数である。upperThan の値を小さく設定するほど射程距離と感度は上がるが、ノイズによる誤作動の確率も上がってしまう。したがって、様々な環境での機器の使用を考慮すると、余裕をもった値を定義する必要がある。今回は窓から遠い薄暗い廊下での展示が予定されていたので、upperThan の値は 50 として定義した。
IRtime_old,IRtime_now はそれぞれ、最後に被弾した時間と現在の時間を格納する。
IRtime_old は被弾時に更新し、IRtime_now は ledResp()実行時に初期化する。
barrier_time は被弾後に発生する無敵時間の長さを定義する定数である。ゲームのルールでは無敵時間は 2 秒なので、barrier_time は 2000 と定義した。

次に、被弾時の処理に関して解説する。

赤外線銃からの攻撃が被弾したと判断した後、無敵時間の有無でダメージ処理とバリア処理に分岐する。ダメージ処理では、ダメージを受けたことを表現する tone 関数による効果音(ソの音)を再生し、ダメージを受けたことを知らせる通信を PIC と行う。バリア処理も同様に、効果音(ド)の再生と、無敵時間中に被弾したことを知らせる通信を PIC と行う。
なお、ハード及びソフト的に、被弾時の方向を特定することができる設計となっているが、製品の仕様上、被弾方向の特定は行う必要がないので、被弾時の処理は被弾方向に依らず一括化した。

＜bounceUpdater()について＞

bounceUpdater()は、接触センサであるスイッチのチャタリング除去処理を行う関数である。bounceUpdater()のアルゴリズムは以下のフローチャートにより表現できる。

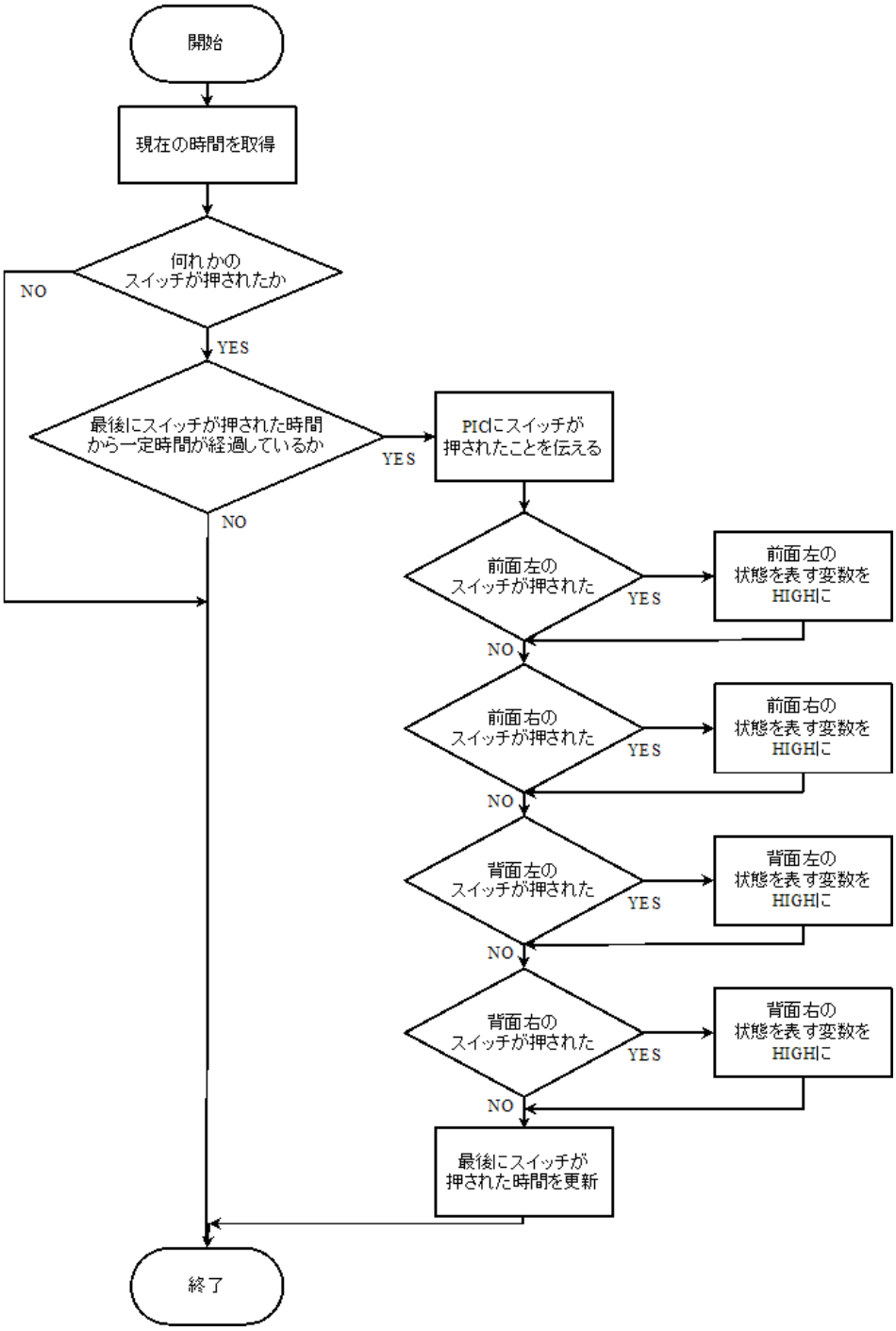

図 6.4 bounceUpdater 関数フローチャート

以上のフローチャートを元に、実際にプログラムとして定義した bounceUpdater()は以下のようになる。

・**bounceUpdater()**

```
void bounceUpdater(){

  static unsigned long swTime_old = 0;
  unsigned long swTime_now = millis();

  if(digitalRead(FL) || digitalRead(FR) || digitalRead(BL) || digitalRead(BR)){
    if((swTime_now - swTime_old) > bounce_time){
   /*------PICとの通信------*/
    digitalWrite(RA0,LOW);
    digitalWrite(RA1,HIGH);
    digitalWrite(RB0,HIGH);
    digitalWrite(RB0,LOW);
    digitalWrite(RA0,LOW);
    digitalWrite(RA1,LOW);
   /*----------------------*/
     if(digitalRead(FL) == HIGH){
        swFL=HIGH;
    }
     if(digitalRead(FR) == HIGH){
        swFR=HIGH;
    }
     if(digitalRead(BL) == HIGH){
        swBL=HIGH;
    }
     if(digitalRead(BR) == HIGH){
        swBR=HIGH;
    }
    swTime_old = swTime_now;

 }
}
}
```

関数中に記述されている変数及び定数について解説する。

swTime_now,swTime_old はそれぞれ、スイッチが押された時の現在の時間と、スイッチが最後に押された時の時間を格納する変数である。

bounce_time は、スイッチが押された時の現在の時間と、スイッチが最後に押された時の時間を比較した際、スイッチが押されたと判断する時間差を決めるための定数である。つまり、bounce_time の値を小さく設定するほどスイッチは敏感になり、大きく設定するほど鈍感になる。今回用いるスイッチは障害物を検知する接触センサとして使用するので、その使用方法に適した 0.5 秒間隔相当の 500 で定義した。

swFL, swFR, swBL, swBR は、それぞれ前面左,前面右,背面左,背面右に設置されたスイッチの状態を表す変数である。それぞれのスイッチが押されているか否かを High/Low で表現し、初期値は Low である。

＜swResp()について＞

swResp()は、接触センサ反応時の処理を行う関数である。反応した接触センサの位置によって、処理を分岐させる。共通の処理として最後に、スイッチの状態を表す変数を初期値の偽に戻す。swResp()のアルゴリズムは以下のフローチャートにより表現できる。

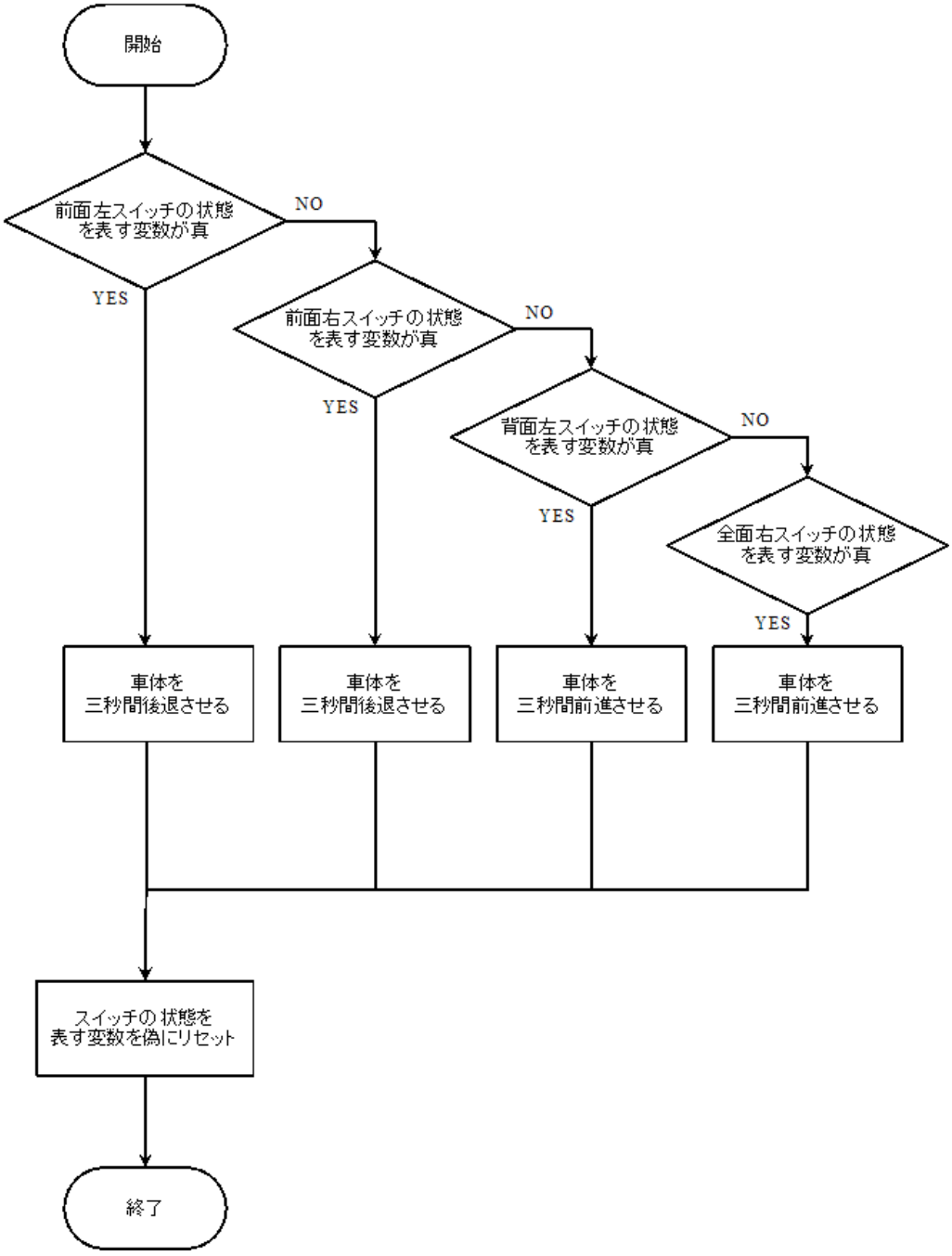

図 6.5 swResp 関数フローチャート

以上のフローチャートを元に、実際にプログラムとして定義した swResp ()は以下のようになる。

・**swResp ()**

```
void swResp(){
  if(swFL == HIGH){
   Tank.Back(255,255);
   delay(3000);

 }
 else if(swFR == HIGH){
   Tank.Back(255,255);
   delay(3000);
 }
 else if(swBL == HIGH){
   Tank.Forward(255,255);
   delay(3000);
 }
 else if(swBR == HIGH){
   Tank.Forward(255,255);
   delay(3000);
 }
 swFL = LOW , swFR = LOW , swBL = LOW , swBR = LOW;
}
```

関数中に記述されているオブジェクトについて解説する。

Tank は MovingControlP 型のインスタンスである。MovingControlP はモータードライバへの信号を制御するために作成したライブラリである。

このライブラリは Forward,Back,SpinCW,SpinCCW,Stay の 5 つのメソッドを持ち、それぞれ、前進,後退,時計回り超信地旋回,反時計回り超信地旋回,停止を行う信号パターンをモータードライバへ送る処理を行う。これらメソッドは左右のモーターの回転速度を指定するための引数を与えることができるよう実装した。モーターの回転速度の制御は、Arduino 関数 analogWrite による PWM 制御で行う都合上、モーターの回転速度を指定する引数の最大値は 255 である。

ライブラリ MovingControlP についての解説は、後の章で行う。

swResp()では、障害物に対して反射するよう車体の制御を行う(これによる車体の動きを以下 退避移動 と呼ぶ)。車体の移動速度を考慮すると、障害物から十分に離れた位置へ移動し製品の安全を確保するためには、3 秒間の移動が適当であると判断した。退避移動時間の指定に delay()を用いているため、退避移動中は被弾判定を行わず、結果的に制限時間が消費されプレイヤーが不利となってしまう。これは障害物に衝突する前に攻撃を被弾させる必要性とゲーム的な駆け引きが生まれると判断し、この仕様を採用した。

＜tankControl()について＞

tankControl()は、ランダムな時間、ランダムな方向へ車体を移動させる関数である。tankControl()のアルゴリズムは以下のフローチャートにより表現できる。

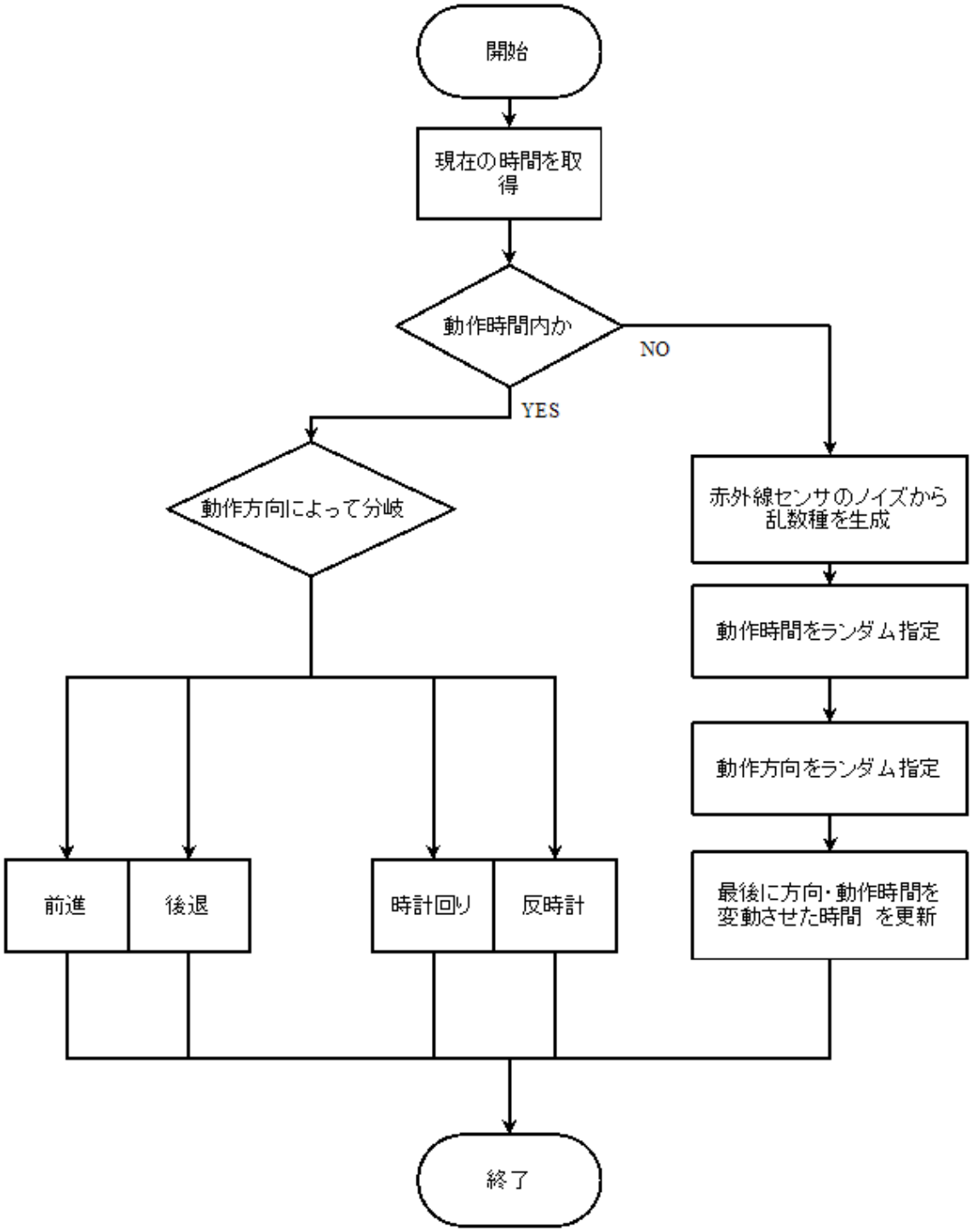

図 6.6 tankControl 関数フローチャート

以上のフローチャートを元に、実際にプログラムとして定義した tankControl ()は以下のようになる。

・tankControl()

```
void tankControl(){
     static unsigned long tankTime_old = 0;//最後に方向・動作時間を変動させた時間
     unsigned long tankTime_now = millis();//現在の時間

     if(randTime > (tankTime_now - tankTime_old)){
       switch(randDirection){
         case 0:
           Tank.Forward(255,255);
           break;
         case 1:
           Tank.Back(255,255);
           break;
          case 2:
           Tank.SpinCW(255,255);
           break;
          case 3:
           Tank.SpinCCW(255,255);
           break;
       }
     }
     else{
        randomSeed(analogRead(1));//赤外線センサのノイズからランダムシード生成
        randTime = random(1000,3000);//動作時間 500～3000ミリ秒をランダムに指定
        randDirection = random(4);//方向をランダムに指定 0:前進 1:後退 2:右旋回 3:左旋回
        tankTime_old = tankTime_now;
     }

}
```

まず、関数に記述されている Arduino 言語関数について解説する。

randomSeed()は、疑似乱数ジェネレータを初期化し、乱数列の任意の点からスタートさせる関数である。固定された乱数列による疑似乱数を用いたランダム移動では、車体の動きがパターン化してしまう。製品の仕様上、パターンを読まれてしまってはゲームが成立しない。そこで、赤外線センサの不規則なノイズを乱数種に用いて疑似乱数ジェネレータを初期化することによって、完全にランダムな車体制御を実現した。

次に、関数中に記述されている変数及び定数について解説する。

tankTime_old,tankTime_now はそれぞれ、最後に動作方向・動作時間をランダムに決定した時間と、現在の時間を格納する変数である。

randTime はランダムに指定した動作時間を格納する変数である。製品の仕様上、激しく動きすぎず、尚且つゲームが成立する動きの多彩さを両立するため、今回は、１秒～3 秒の間を範囲にランダムに指定している。

「最後に動作方向及び動作時間をランダムに決定した時間」と「現在の時間」との差が、randTime を超えた場合、再度乱数種を初期化し、動作方向・時間をランダムに決定する処理を行う。

randDirection は、ランダムに指定した動作方向を格納する変数である。製品の仕様では、前進,後退,時計回り超信地旋回,反時計回り超信地旋回をランダムに繰り返すことになっている。0,1,2,3 をそれぞれ前進,後退,時計回り超信地旋回,反時計回り超信地旋回の動作に対応させている。randDirection にランダムな 0～3 までの範囲の乱数を格納し、randDirection の値によって動作を分岐させることにより。ランダムな移動方向制御を実現している。

＜deadAct()について＞

deadAct()は、死亡時の処理を行う関数である。

HP が 0 になったという通信が PIC から送られている場合、ゲームクリアを表現する BGM を一度だけ再生し、車体に停止動作を行わせる。なお、HP の管理は PIC が行っているため、この関数では PIC との通信は行わない。

実際にプログラムとして定義した deadAct()は以下のようになる。

・deadAct()

```
void deadAct(){
   if(!end_flag){
     delay(1000);
     sound(1);//ゲームクリアサウンド
     end_flag = true;
     }
     Tank.Stay();//車体停止
}
```

関数中に記述されている変数及び定数について解説する。

end_flag は、初期値 false の bool 型変数である。deadAct()は死亡判定中、繰り返し実行される。しかし、ゲームクリア時の BGM を流すのは一度だけであるので、end_flag を BGM 再生済みフラグとして用いて、一度きりの再生を実現した。

次にプログラムの工夫について述べる。

ダメージ時の効果音の再生直後に、ゲームクリア BGM を再生すると、音楽的に不自然な聞こえ方になる現象が、テスト時に発見された。そこで、BGM 再生前に delay(1000)で間を空けることにより、自然な BGM の再生を実現した。

＜timeOver()について＞

timeOver()は、制限時間経過時の処理を行う関数である。

loop()における処理で、制限時間が経過したと判断した場合、タイムオーバーを表現するBGM を一度だけ再生し、車体に停止動作を行わせる。Arduino が制限時間を管理しているため、PIC に制限時間が経過したことを伝える通信を行う。

実際にプログラムとして定義した timeOver()は以下のようになる。

・timeOver()

```
void timeOver(){
    /*------PICとの通信(タイムオーバー)------*/
    digitalWrite(RA0,HIGH);
    digitalWrite(RA1,LOW);
    digitalWrite(RB0,HIGH);
    /*--------------------------------------*/
     if(!end_flag){
     sound(2);//タイムオーバーサウンド
     end_flag = true;
     }
    Tank.Stay();//車体停止
}
```

関数中に記述されている変数及び定数について解説する。

end_flag は、先述の deadAct()と共通の変数を用いている。timeOver()と deadAct()はどちらもゲーム終了時に実行される関数なので、フラグを共有しても影響はない。

6-2-3.ライブラリについて

作成した車体制御ライブラリ MovingControlP は MovingControlP.h と MovingControlP.cpp の二つのファイルで構成されている。以降に、MovingControlP を構成するファイルとそのソースを示す。

・MovingControlP.h

```
#ifndef MovingControlP_h
#define MovingControlP_h
#include "Arduino.h"


class MovingControlP
{
  public:
         MovingControlP(int PinNoR1,int PinNoR2,
                        int PinNoL1,int PinNoL2);// コンストラクタ
    void Forward(int rPower , int lPower);
    void Back(int rPower , int lPower);
    void Stay();
    void SpinCCW(int rPower , int lPower);
    void SpinCW(int rPower , int lPower);

  private:
    int M_PinNoR1; // モーターの接続されているピン番号を保存する変数
    int M_PinNoR2;
    int M_PinNoL1;
    int M_PinNoL2;
} ;

#endif
```

・MovingControlP.cpp

```
#include "Arduino.h"
#include "MovingControlP.h"

MovingControlP::MovingControlP(int PinNoR1,int PinNoR2,
                               int PinNoL1,int PinNoL2)
{
     M_PinNoR1 = PinNoR1;
     M_PinNoR2 = PinNoR2;
     M_PinNoL1 = PinNoL1;
     M_PinNoL2 = PinNoL2;

}

//前進
void MovingControlP::Forward(int rPower , int lPower)
{
  analogWrite(M_PinNoR1,rPower);
  analogWrite(M_PinNoR2,0);
  analogWrite(M_PinNoL1,lPower);
  analogWrite(M_PinNoL2,0);
}
//後退
void MovingControlP::Back(int rPower , int lPower)
{
  analogWrite(M_PinNoR1,0);
  analogWrite(M_PinNoR2,rPower);
  analogWrite(M_PinNoL1,0);
  analogWrite(M_PinNoL2,lPower);
}
//停止
void MovingControlP::Stay()
{
  analogWrite(M_PinNoR1,0);
  analogWrite(M_PinNoR2,0);
  analogWrite(M_PinNoL1,0);
  analogWrite(M_PinNoL2,0);
}
//信地旋回：反時計
void MovingControlP::SpinCCW(int rPower , int lPower)
{
  analogWrite(M_PinNoR1,rPower);
  analogWrite(M_PinNoR2,0);
  analogWrite(M_PinNoL1,0);
  analogWrite(M_PinNoL2,lPower);
}
//信地旋回：時計
void MovingControlP::SpinCW(int rPower , int lPower)
{
  analogWrite(M_PinNoR1,0);
  analogWrite(M_PinNoR2,rPower);
  analogWrite(M_PinNoL1,lPower);
  analogWrite(M_PinNoL2,0);
}
```

以上に示したライブラリ中に記述されている引数及び変数について解説する。

まず、コンストラクタ及びその引数,変数についてであるが、コンストラクタでは、渡された引数を元に、Arduino のピンとモータードライバのピンとの関連付けを行っている。
PinNoR1 は、右側モーターを制御するモータードライバの 5 番ピン、
PinNoR2 は、右側モーターを制御するモータードライバの 6 番ピン、
PinNoL1 は、左側モーターを制御するモータードライバの 5 番ピン、
PinNoL2 は、左側モーターを制御するモータードライバの 6 番ピンへ接続されている Arduino のデジタルピンの番号に対応した引数である。
M_PinNoR1,M_PinNoR2,M_PinNoL1,M_PinNoL2
はそれぞれ
PinNoR1,PinNoR2,PinNoL1,PinNoL2
の値を格納する変数である。

次に、各メソッド及びその引数について解説する。
rPower,lPower はそれぞれ右と左のモーターの回転速度を指定するために用いられる引数である。先述のとおり、モーターの回転速度の指定は、Arduino 関数 analogWrite による PWM 制御で行う都合上、モーターの回転速度を指定する引数の最大値は 255 である。各車体制御メソッド毎に左右モーター回転速度を指定可能なライブラリの構造を採用することにより、複雑な車体制御を行う際の記述を簡略化することができた。

Forward(int rPower , int lPower)は車体を前進させるよう、モータードライバへ信号を送るメソッドである。引数の組み合わせ方によっては、直進以外にも、迂回を行いながら前進させる車体制御が可能である。

Back(int rPower , int lPower)は車体を前進させるよう、モータードライバへ信号を送るメソッドである。引数の組み合わせ方によっては、直進以外にも、迂回を行いながら後退させる車体制御が可能である。

Stay()は車体を停止させるよう、モータードライバへ信号を送るメソッドである。モーターの速度を 0 に指定することにより車体を停止させるため、引数は与えられない。

SpinCCW (int rPower , int lPower)は車体を反時計方向に信地旋回させるよう、モータードライバへ信号を送るメソッドである。引数の組み合わせ方によっては、滑らかに旋回する信地旋回と、その場で機敏に旋回する超信地旋回を選択することが可能である。

SpinCW (int rPower , int lPower)は車体を時計方向に信地旋回させるよう、モータードライバへ信号を送るメソッドである。引数の組み合わせ方によっては、滑らかに旋回する信地旋回と、その場で機敏に旋回する超信地旋回を選択することが可能である。