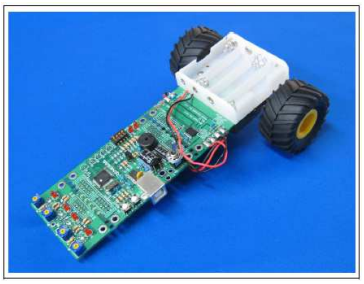
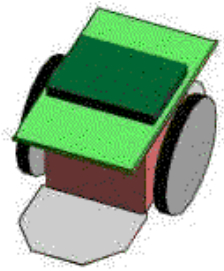
自立型ロボットに挑戦

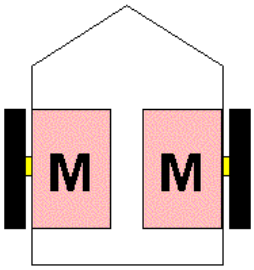


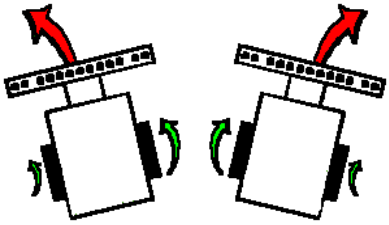
　自立型ロボットとは、リモコン(有線)型でもラジコン(無線)型でもなく、人間の力を借りずに自ら動くロボットのことです。今日は自立型ロボットの入門用として位置づけられているライントレースロボットを通しロボットプログラミングを体験して頂きます。

1．ロボットはどう動く？

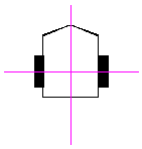
今回製作するロボットは、車輪走行型ロボットです。台車の左右に動輪を配置した左右二輪速度差方式を採用しています。

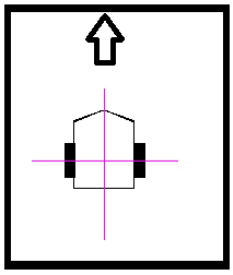


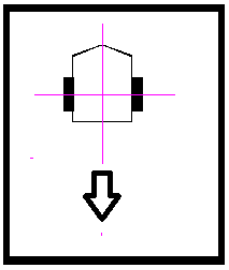


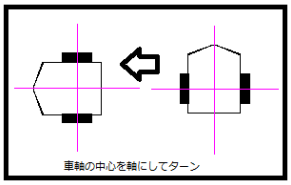


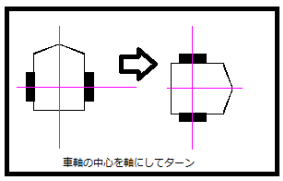
左右二輪速度差方式では、前進、後退、左右旋回、その場回転などの走行が可能です。
(前進・後進・右折・左折・右ピボット・左ピボット・右後ピボット・左後ピボット・停止)



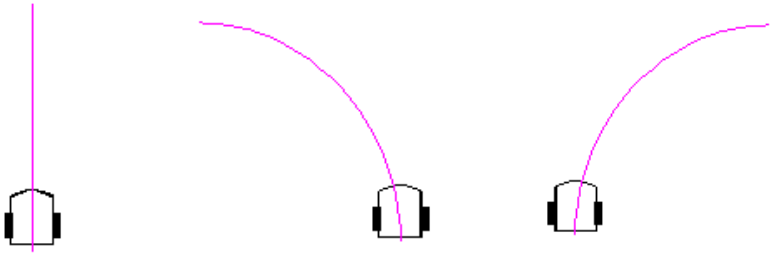




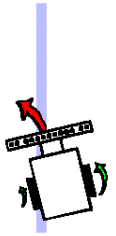




左右車輪の回転速度が等しければ前進（後退）
左が右より遅ければ左スラローム
右が左より遅ければ右スラローム
左右輪が逆回転すると、その場で回転します。（信地旋回と呼んでいます。）



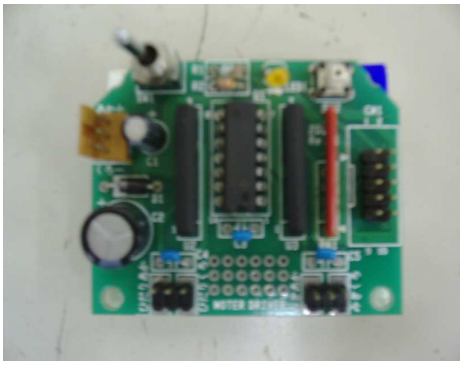
今回は電池でモータを駆動し、車輪を回します。
マイクロコンピュータ(CPU)によりモータの回転スピードや回転量を変えることでできます。また今回使用するロボットは、赤外線センサを搭載しています。センサより目標位置とのずれを検出し、常にロボットがラインの中心に位置修正しながら進むことが可能となります。



２．ＤＣモータをまわす（メカトロニクスの第１歩）

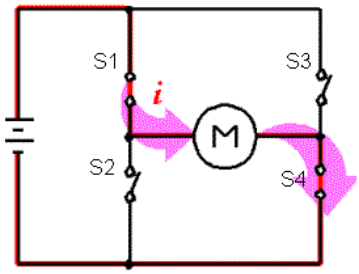
今回使用ロボットは、ＤＣモータを採用しています。
DC モータは、磁界の中で導線に電流を流すと力が作用する性質を利用したモータです。
（フレミングの左手の法則といい、中学校で学習します。）
ＤＣモータは、モータ端子に電圧を与えると(印加する)回ります。回転方向は、モータ端子の＋ー極性を逆に接続すると逆転します。また、モータの端子をショートさせると、ブレーキカが発生し、回転を止めることができます。

ＤＣモータ

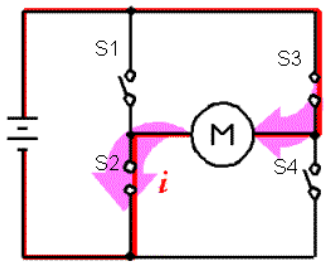


ＤＣモータドライブ回路

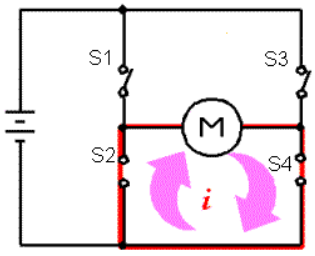
ＤＣモータを駆動するときは、一般的には４個のスイッチを利用してモータ電圧を加え回転方向を変更しています。



Ｓ１とＳ４をONにするとモータは正回転



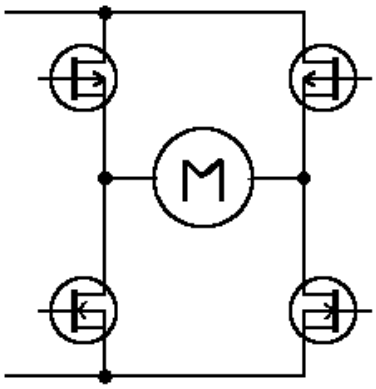
Ｓ２とＳ３をONにするとモータは逆転



Ｓ２とＳ４をONにすると停止(ブレーキ)

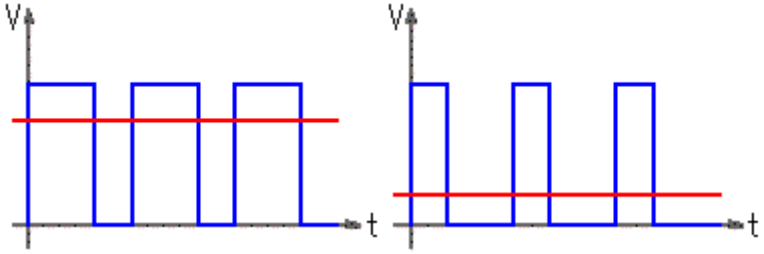
HブリッジによるＤＣモータ駆動 (高度な制御方法)

一般的なＤＣモータ駆動回路では、トランジスタあるいはＦＥＴによる半導体スイッチでHブリッジを構成しています。



Hブリッジ回路例

DC モータ回転速度は、モータに与える電圧を高くすると速く、低くすると遅く回ります。ミニマイコンカーでは、モータ電圧をアナログ的に変化するのではなく、ＯＮ－ＯＦＦを繰り返すスイッチング制御を採用しています。モータに加わる平均電圧を調整して、速度制御しているのです。モータ回転速度は、スイッチング方式で ON 時間の割合が大きいと、モータの平均印加電圧が高くなるので、速くなります。逆に、ON 時間の割合が小さいときは、平均電圧が低くなるので、モータ回転速度が遅くなります。



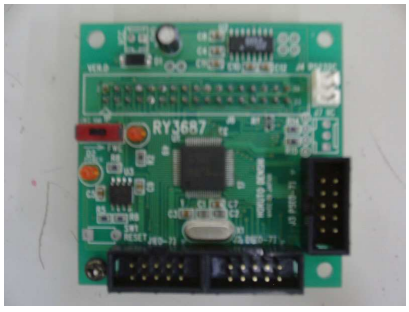
PWM波形（赤い線が平均電圧）

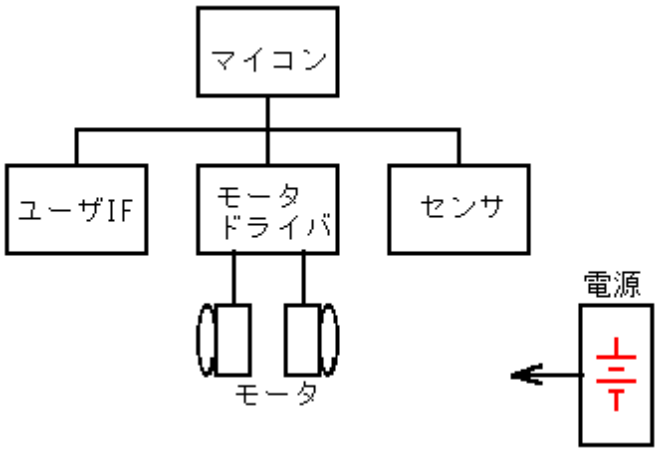
ＯＮ－ＯＦＦのスイッチングは数百Ｈｚ～数十ｋＨｚで行います。極めて短い時間のＯＮ－ＯＦＦですから、ＯＦＦの間もモータは惰性で回り続けます。そうするとモータ回転がガタつくようなことはなく滑らかに回転します。電圧をアナログ的に変化させると、電源との電位差を熱にしてエネルギー放出するためエネルギー効率がよくありません。スイッチング方式では熱としてエネルギー放出される量が少ないので、エネルギー効率のよい方式です。モータドライバはＦＥＴにより構成していますので、トランジスタ方式に比べより効率的なモータ駆動が可能です。

今回のミニマイコンカーでは FET 回路がパッケージとなっている専用 IC を使用しています。

３．マイコンを使う (ロボットの頭脳部)

ロボットは、マイクロコンピュータ（マイコン）と呼ばれる小型コンピュータで制御します。ロボットをどう動かすかは、ソフトウェア(プログラム)次第です。プログラムはパソコンを用いて作成します。通常開発言語はＣ言語を用います。高級言語を用いることで、ＣＰＵが変更されても今まで利用していたプログラムを大きく変更することなく引き継ぐことが出来ます。





ロボットの原理ブロック図

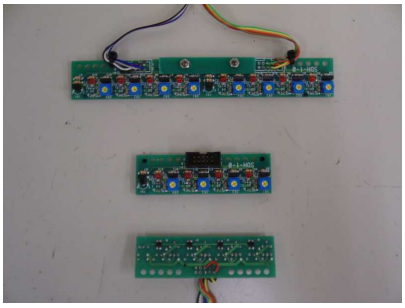
４．ロボットとの通信

PC 上で作成したロボット用プログラムを、マイコンのＣＰＵが理解できるマシン語に変換するのが、コンパイラ，リンカーです。出来上がったマシン語プログラムを、シリアル通信などでマイコン上のメモリに転送します。今回使用するCPUは、シリアル通信端子を備えています。パソコン上で作成されたロボット用プログラムは、シリアル通信を通じてパソコンからロボットに転送して利用します。

パソコンとＰＣの接続

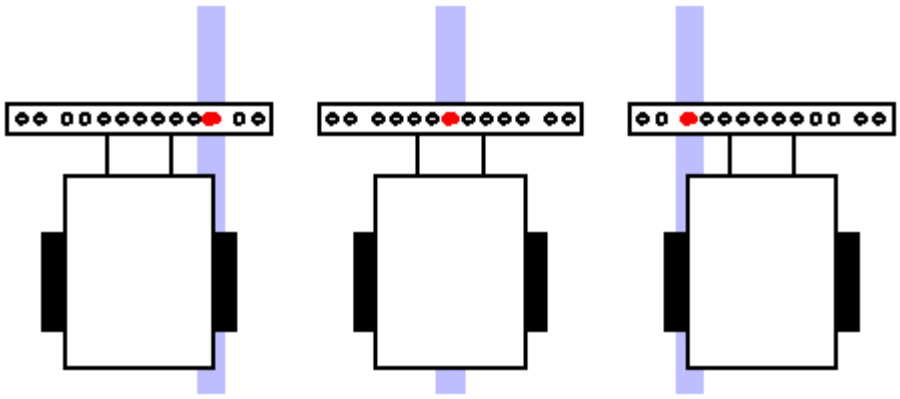
５．センサを使う (人間にたとえると目にあたります)

今回使用するロボットは、赤外線センサを搭載しています。ロボットから赤外線を照射し、コースラインに反射した光をロボットのセンサが受光します。ロボットはこの赤外線の反射光を検出しています。



ロボットのセンサ例

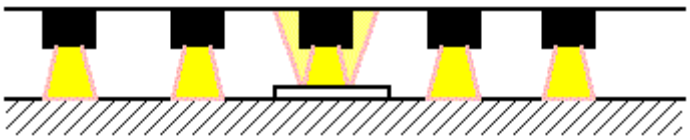
一般的なトレースロボットセンサは赤外線(光)を照射し、床からのはね返り光のあるなしをチェックしています。センサを複数個配置しているので、どのセンサが反応しているかをチェックすれば、目標位置とのズレを検出できます。たとえば、ロボットの中央センサのみラインを検出しているときは直進、左側のどれか１つのセンサがラインを検出したときはロボットがラインより右にズレている、右側センサがラインを検出したときは左にズレていると判断することができます



左にズレた　直進　右にズレた
姿勢の判断

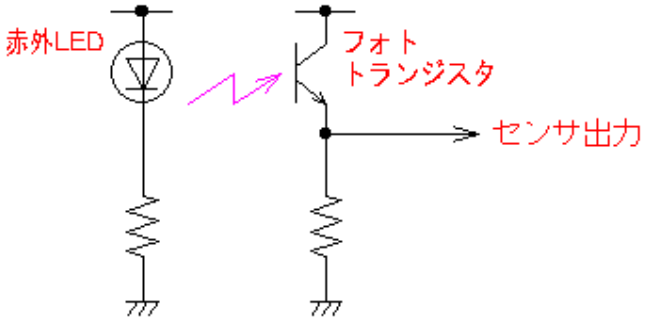
ラインセンサーの動作原理と回路

ライントレースでは、黒い床に白いラインが配置されているので、黒い床面上に位置しているセンサには光がほとんど反射しませんが、ライン上にあるセンサには光が反射してきます。多くの光を検出したセンサは、ラインを検出したと判断してマイコンに伝えます。



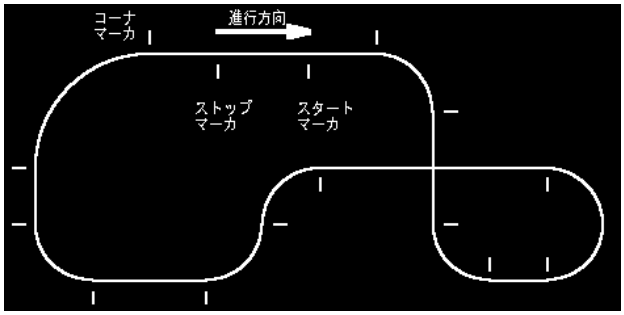
ラインセンサ原理

光センサは、発光素子と受光素子で構成されています。(作ります)
発光素子には赤外線 LED がよく利用されます。受光素子には、フォトトランジスタやフォトダイオードなどが利用されています。これらの受光素子は、光が当たると電流が流れるようになる素子なので、発光素子から照射した光が対象物に当たって反射してくれば、光センサ回路回路に反射量に比例した電流が流れて対象物の検出ができます



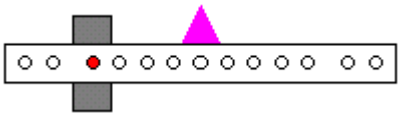
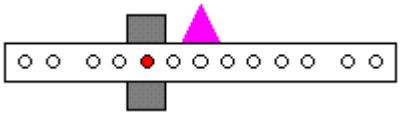
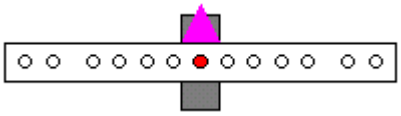
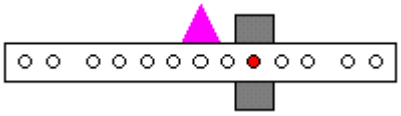
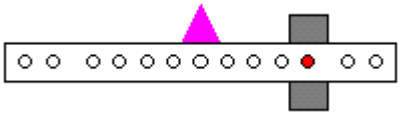
(自分で作るときに参考にして下さい)

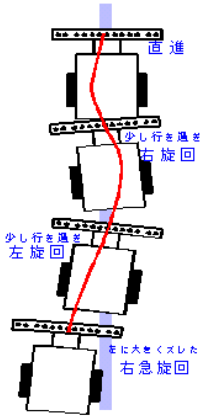
6．ラインとレースしよう



ラインとレースとは、床面に描いたラインをロボットがセンサを利用して読み取り、ラインに沿って走行することです。ライントレース競技では、黒い床面に白いテープを貼ってコースを構成しています。ロボットはこのラインを光センサで読み取り、ライン追従して走行します。一般的なライントレースのコースは、直線と曲線の組み合わせでつくられています。マイクロマウスロボトレース競技では、直線と曲線の境にマーカが設置してあり途中に十字に交差する場所「クロスライン」があります。コースは平面が一般的ですがマイコンカーラリー競技では山あり谷ありで立体交差、さらに直角に曲がる「クランク」、途中でラインが切れる「レーンチェンジ」のある競技会もあります。

制御のルール例

センサの反応	ロボットの状態	制御ルール
	右に大きくズレた	左急旋回
	右に少しズレた	左旋回
	ライン中心上	直進
	左に少しズレた	右旋回
	左に大きくズレた	右急旋回



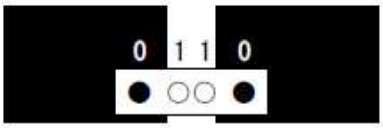
この制御ルールを適応すると、ロボットは直線に戻るように走ります。

実際にプログラムするときは、センサが反応する全てのパターンを想定した、もっとたくさんの場合分けが必要です。ただし、どれだけ場合分けを増やして、細かな旋回半径の設定をしても、あまり速い速度で走ると車輪がスリップしてコースから飛び出してしまうので注意が必要です。タイヤのグリップ力を考慮し速度設定します。直線区間と曲線区間を見分けて加減速できるようになるとさらに効率よく走れ、タイムを縮めることができます。以上でライントレースロボットの概要説明は終了します。ロボットエンジニア目指して皆さん頑張って下さい!!

ミニマイコンカー参考資料 少しC言語プログラムを見てみよう

(1) 中央を走行しているとき

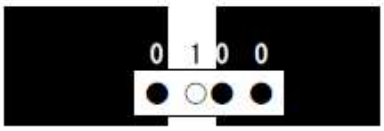
```
111 :                   case 0x06:
112 :                           // 0000 0110 センター→まっすぐ
113 :                           motor( 100, 100 );
114 :                           break;
```



センサーが“0x06”の状態です。この状態は、上図のようにラインの中央を走行している状態です。左のモーターを「100%」右のモーターを「100%」で回し、直進させます。

(2) 少し右側を走行しているとき

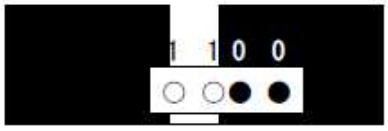
```
116 :                   case 0x04:
117 :                           // 0000 0100 少し右寄り→左へ小曲げ
118 :                           motor( 85, 100 );
119 :                           break;
```



センサーが“0x04”の状態です。この状態は、上図のようにラインの少し右側を走行している状態です。左のモーターを「85%」右のモーターを「100%」で回し、中央にセンサーが来るようにします。

(3) 中くらい右側を走行しているとき

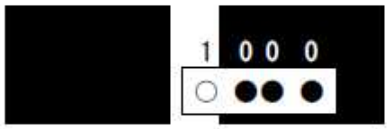
```
121 :                   case 0x0c:
122 :                           // 0000 1100 中くらい右寄り→左へ中曲げ
123 :                           motor( 70, 100 );
124 :                           break;
```



センサーが“0x0c”の状態です。この状態は、上図のようにラインの中くらい右側を走行している状態です。左のモーターを「70%」右のモーターを「100%」で回し、中央にセンサーが来るようにします。

(4) 大きく右側を走行しているとき

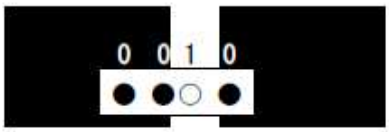
```
126 :                   case 0x08:
127 :                           // 0000 1000 大きく右寄り→左へ大曲げ
128 :                           motor( 55, 100 );
129 :                           break;
```



センサーが“0x08”の状態です。この状態は、上図のようにラインの大きく右側を走行している状態です。左のモーターを「55%」右のモーターを「100%」で回し、中央にセンサーが来るようにします。

(5) 少し左側を走行しているとき

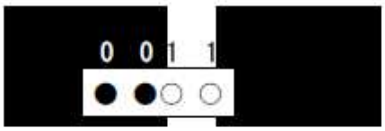
```
131 :                   case 0x02:
132 :                           // 0000 0010 少し左寄り→右へ小曲げ
133 :                           motor( 100, 85 );
134 :                           break;
```



センサーが“0x02”の状態です。この状態は、上図のようにラインの少し左側を走行している状態です。左のモーターを「100%」右のモーターを「85%」で回し、中央にセンサーが来るようにします。

(6) 中くらい左側を走行しているとき

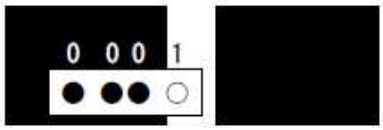
```
136 :                   case 0x03:
137 :                           // 0000 0011 中くらい左寄り→右へ中曲げ
138 :                           motor( 100, 70 );
139 :                           break;
```



センサーが“0x03”の状態です。この状態は、上図のようにラインの中くらい左側を走行している状態です。左のモーターを「100%」右のモーターを「70%」で回し、中央にセンサーが来るようにします。

(7) 大きく左側を走行しているとき

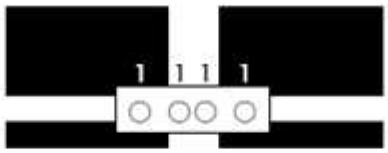
```
141 :                   case 0x01:
142 :                           // 0000 0001 大きく左寄り→右へ大曲げ
143 :                           motor( 100, 55 );
144 :                           break;
```



センサーが“0x01”の状態です。この状態は、上図のようにラインの大きく左側を走行している状態です。左のモーターを「100%」右のモーターを「55%」で回し、中央にセンサーが来るようにします。

(8) クロスラインを検出しているとき

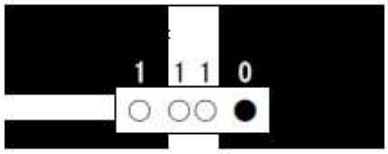
```
146 :                   case 0x0f:
147 :                           // 0000 1111 クロスライン検出
148 :                           motor( 100, 100 );
149 :                           cnt1 = 0;
150 :                           pattern = 21;
151 :                           break;
```



センサーが“0x0f”の状態です。この状態は、上図のようにクロスラインを検出している状態です。左のモーターを「100%」右のモーターを「100%」で回し、直進させます。cnt1 変数をクリアして、パターン 21 に行きます。

(9) 左ハーフラインを検出しているとき

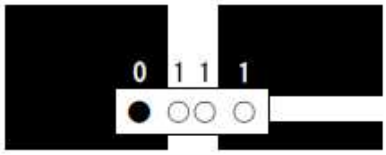
```
153 :                   case 0x0e:
154 :                           // 0000 1110 左ハーフライン検出
155 :                           motor( 100, 100 );
156 :                           cnt1 = 0;
157 :                           pattern = 31;
158 :                           break;
```



センサーが“0x0e”の状態です。この状態は、上図のように左ハーフラインを検出している状態です。左のモーターを「100%」右のモーターを「100%」で回し、直進させます。cnt1 変数をクリアして、パターン 31 に行きます。

(10) 右ハーフラインを検出しているとき

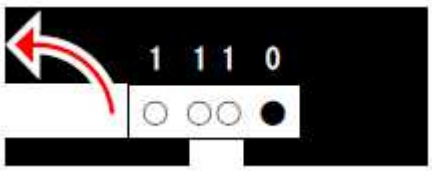
```
160 :                   case 0x07:
161 :                           // 0000 0111 右ハーフライン検出
162 :                           motor( 100, 100 );
163 :                           cnt1 = 0;
164 :                           pattern = 41;
165 :                           break;
```

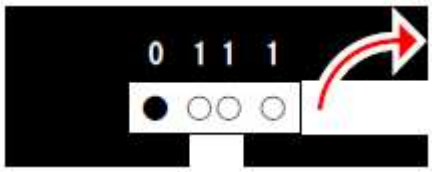


センサーが“0x07”の状態です。この状態は、上図のように右ハーフラインを検出している状態です。左のモーターを「100%」右のモーターを「100%」で回し、直進させます。cnt1 変数をクリアして、パターン 41 に行きます。

```
case 21:
        // クロスライン検出後のトレース、クランク検出
        beep(Def_C3);                          クロスライン検出後初めにドの音を出します。
                                               クロスラインルーチンに入ったか確認するためです。
        switch( ( sensor() & 0x0f ) ){         直角を見つけるまでは通常トレースします。
        case 0x06:
                // 0000 0110 センター→まっすぐ
                motor( 100, 100 );
                break;

        case 0x04:
                // 0000 0100 少し右寄り→左へ小曲げ
                motor( 85, 100 );
                break;

        case 0x0c:
                // 0000 1100 中くらい右寄り→左へ中曲げ
                motor( 70, 100 );
                break;

        case 0x08:
                // 0000 1000 大きく右寄り→左へ大曲げ
                motor( 55, 100 );
                break;

        case 0x02:
                // 0000 0010 少し左寄り→右へ小曲げ
```

```
            motor( 100, 85 );
            break;

        case 0x03:
            // 0000 0011 中くらい左寄り→右へ中曲げ
            motor( 100, 70 );
            break;

        case 0x01:
            // 0000 0001 大きく左寄り→右へ大曲げ
            motor( 100, 55 );
            break;

        default:
            break;

        }

        if( cnt1 >= 1000 ){
            switch( ( sensor() & 0x0f ) ){
            case 0x0e:
                // 0000 1110 左クランク検出
                motor( 0, 90 );
                cnt1 = 0;
                pattern = 22;
                break;

            case 0x07:
                // 0000 0111 右クランク検出
                motor( 90, 0 );
                cnt1 = 0;
                pattern = 22;
                break;

            default:
                break;

            }
        }
        break;

case 22:
    // クランクの曲げ動作継続処理
    if( cnt1 >= 1000 ){
```





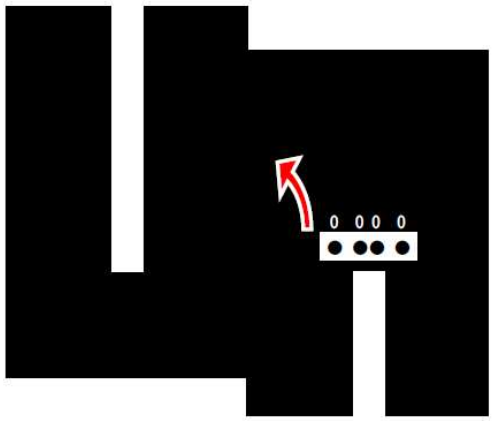
```
            pattern = 11;
        }
        break;
```

if 文では、cnt1 変数が 1000 以上の（1000［ms］経過した）場合、{} 内の文を実行します。1000［ms］経過するまで実行しないようにしているのは、クランクの曲げ動作を継続させるためです。{} 内では、パターン 11 に行きます。

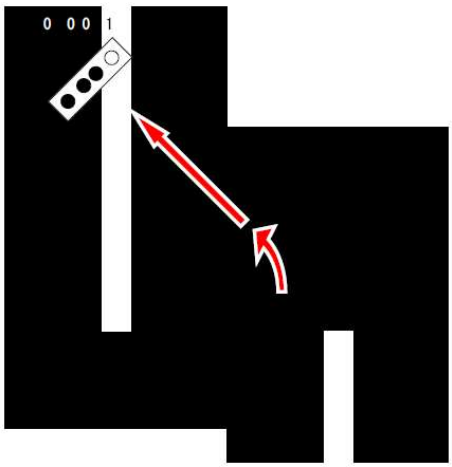
車線変更(レーンチェンジ)攻略イメージ

センサーが“0x00”の状態です。この状態は、上図のように左レーンチェンジを検出している状態です。左のモーターを「n%」右のモーターを「100%」で回し、左レーンチェンジを曲がります。cnt1 変数をクリアして、パターン 32 に行きます。

「n%」の部分には任意の値を入れ、正しく曲がれるように調整をしてください。



センサーが“0x01”の状態です。この状態は、上図のように左レーンチェンジの終了を検出している状態です。パターン 11 に行きます。



これができたら、マイコンカー約50mコースを完走できるでしょう。　丁寧にプログラムを作成して下さい。