Advanced マイコンカー プログラム解説マニュアル

Advanced マイコンカー製作マニュアルで製作した マイコンカーのプログラム解説です

走行動画を掲載しています：https://youtu.be/kHmeGFhO_mk

改訂履歴

1.00	作成
1.00a	走行動画とログを追加
1.01	誤字脱字の修正

第 1.01 版
2023.08.29
ジャパンマイコンカーラリー実行委員会

目　次

1. 免責事項

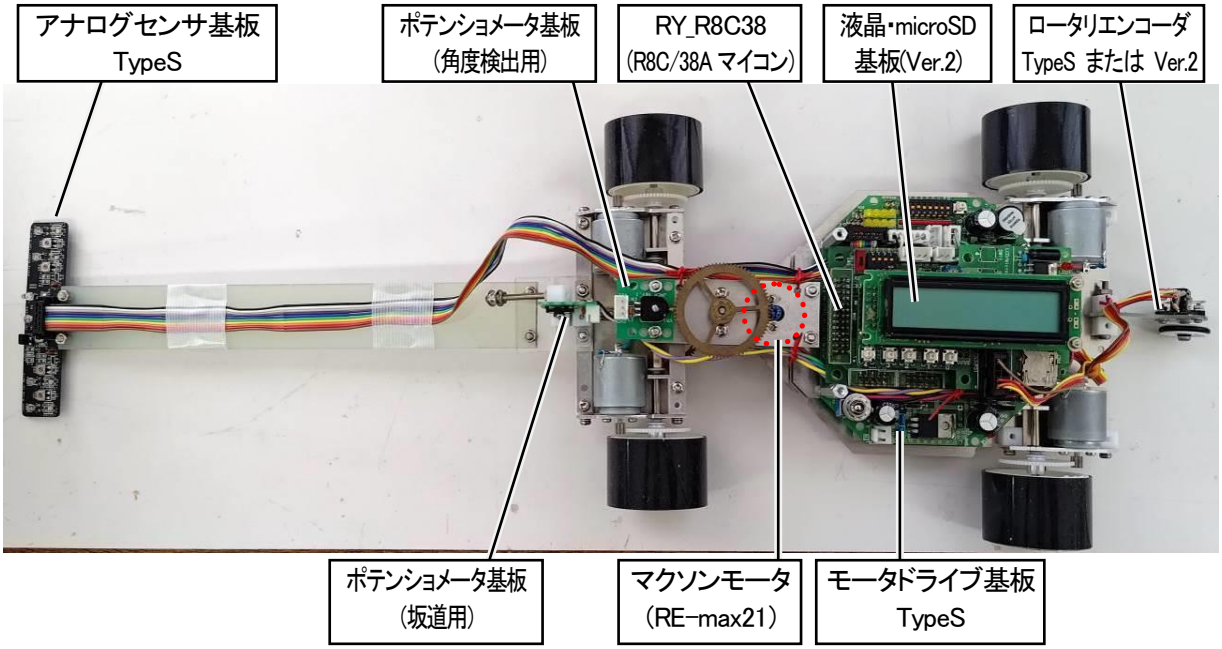
万が一「本マニュアル」による損失・損害が発生した時には、マニュアル制作団体、制作者はいかなる場合も責任を負いません。個人の免責が取れる範囲内であらかじめ了承した上でご使用くださるようにお願いをいたします。

2. 概要

本マニュアルは、別冊の「Advanced マイコンカー製作マニュアル」で製作した Advanced マイコンカーを制御するためのプログラムを解説しているマニュアルです。

2.1. Advanced マイコンカーの構成

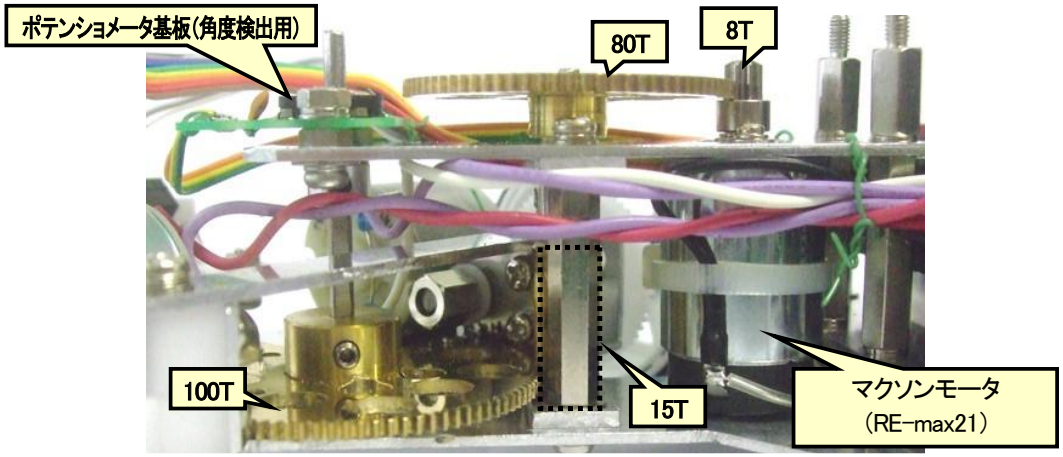
Advanced マイコンカーの構成を下記に示します。プログラムは、ロータリエンコーダ TypeS を使ったプログラムになっています。別なロータリエンコーダの場合は、速度、距離を調整してください。



※ステアリング部分のモータ、ギヤ比について

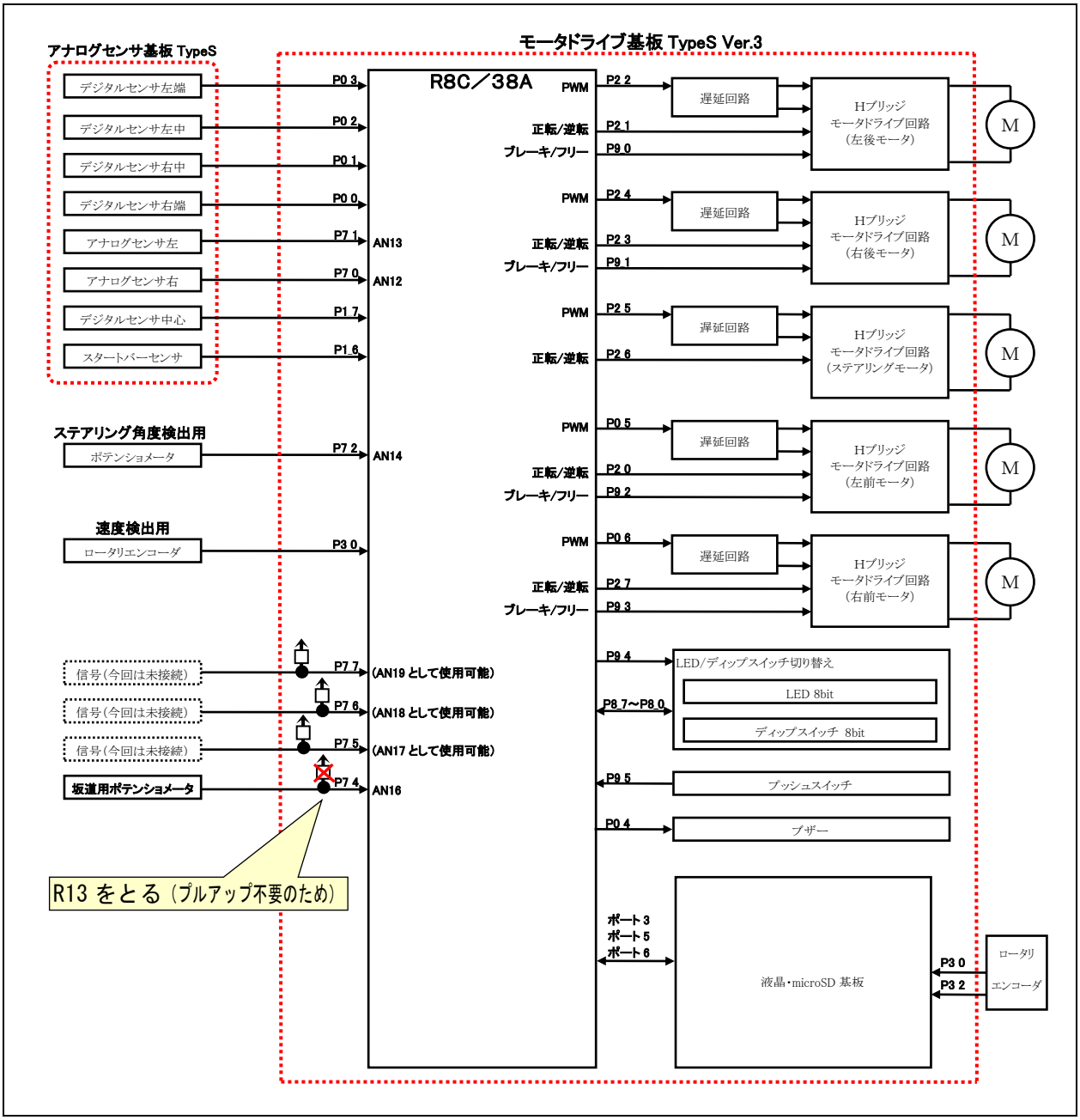
今回は、テアリングモータとして maxon motor 製の「RE-max21」の 9V 版を使用しています。

ギヤ比は、ピニオンギヤ 8T／真鍮ギヤ 80T×真鍮ギヤ 15T／真鍮ギヤ 100T＝120:8000＝1/66.7 としています。



2.2. ブロック図

Advanced マイコンカーのブロック図を下記に示します。



※R13 は取らないと、坂道用ポテンショメータの抵抗の分圧比が変わってしまい、変化量が比例になりません。
必ず R13 は未実装にします。

2.3. ポート表

■ポート 0

bit	接続先の基板	信号名	0	1	入出力方向 マイコンから見て
7	未接続	-	-	-	出力
6	モータドライブ	左前モータ	PWM		出力
5	モータドライブ	右前モータ	PWM		出力
4	モータドライブ	ブザー	-	-	出力
3	アナログセンサ	デジタルセンサ左端	白	黒	入力
2	アナログセンサ	デジタルセンサ左中	白	黒	入力
1	アナログセンサ	デジタルセンサ右中	白	黒	入力
0	アナログセンサ	デジタルセンサ右端	白	黒	入力

■ポート 1

bit	接続先の基板	信号名	0	1	入出力方向 マイコンから見て
7	アナログセンサ	デジタルセンサ中心	白	黒	入力
6	アナログセンサ	スタートバー検出センサ	白	黒	入力
5	RY_R8C38	RxD0	-	-	入力
4	RY_R8C38	TxD0	-	-	出力
3	RY_R8C38	ディップスイッチ(SW3)	ON	OFF	入力
2	RY_R8C38	ディップスイッチ(SW2)	ON	OFF	入力
1	RY_R8C38	ディップスイッチ(SW1)	ON	OFF	入力
0	RY_R8C38	ディップスイッチ(SW0)	ON	OFF	入力

■ポート 2

bit	接続先の基板	信号名	0	1	入出力方向 マイコンから見て
7	モータドライブ	右前モータ	正転	逆転	出力
6	モータドライブ	ステアリングモータ	正転	逆転	出力
5	モータドライブ	ステアリングモータ	PWM		出力
4	モータドライブ	右後モータ	PWM		出力
3	モータドライブ	右後モータ	正転	逆転	出力
2	モータドライブ	左後モータ	PWM		出力
1	モータドライブ	右後モータ	正転	逆転	出力
0	モータドライブ	左前モータ	正転	逆転	出力

■ポート 3

bit	接続先の基板	信号名	0	1	入出力方向 マイコンから見て
7	未接続	-	-	-	出力
6	未接続	-	-	-	出力
5	未接続	-	-	-	出力
4	未接続	-	-	-	出力
3	未接続	-	-	-	出力
2	未接続	-	-	-	出力
1	未接続	-	-	-	出力
0	液晶・microSD	ロータリエンコーダ	パルス		入力

■ポート 4

bit	接続先の基板	信号名	0	1	入出力方向 マイコンから見て
7	RY_R8C38	水晶振動子(20MHz)	−	−	出力
6	RY_R8C38	水晶振動子(20MHz)	−	−	入力
5	RY_R8C38	LED	消灯	点灯	出力
4	未接続	−	−	−	出力
3	未接続	−	−	−	出力
2	RY_R8C38	VCC	電源		入力
1	端子なし	−	−	−	出力
0	端子なし	−	−	−	出力

■ポート 5

bit	接続先の基板	信号名	0	1	入出力方向 マイコンから見て
7	液晶・microSD	SW1	ON	OFF	入力
6	液晶・microSD	液晶 E	液晶との信号やり取り		入力／出力※
5	液晶・microSD	液晶 RW	液晶との信号やり取り		出力
4	液晶・microSD	液晶 RS	液晶との信号やり取り		出力
3	液晶・microSD	液晶 D3	液晶との信号やり取り		出力※
		SW2	ON	OFF	入力※
2	液晶・microSD	液晶 D2	液晶との信号やり取り		出力※
		SW3	ON	OFF	入力※
1	液晶・microSD	液晶 D1	液晶との信号やり取り		出力※
		SW4	ON	OFF	入力※
0	液晶・microSD	液晶 D0	液晶との信号やり取り		出力※
		SW5	ON	OFF	入力※

※プログラムで入出力を切り替えて使用します

■ポート 6

bit	接続先の基板	信号名	0	1	入出力方向 マイコンから見て
7	未接続	−	−	−	出力
6	未接続	−	−	−	出力
5	未接続	−	−	−	出力
4	液晶・microSD	microSD DAT0(RXD1)	microSD との通信		入力
3	液晶・microSD	microSD CMD(TXD1)	microSD との通信		出力
2	液晶・microSD	microSD CLK(CLK1)	microSD との通信		出力
1	液晶・microSD	microSD CD	microSD との通信		出力
0	液晶・microSD	モニタ LED	点灯	消灯	出力

■ポート 7

bit	接続先の基板	信号名	0	1	入出力方向 マイコンから見て
7	モータドライブ	CN6 信号(接続なし)			入力
6	モータドライブ	CN6 信号(接続なし)			入力
5	モータドライブ	CN6 信号(接続なし)			入力
4	モータドライブ	坂道検出用ポテンショメータ	電圧入力(0～5V)		入力
3	モータドライブ	プルアップ抵抗入力	常に"1"を入力		入力
2	モータドライブ	角度検出用ポテンショメータ	電圧入力(0～5V)		入力
1	アナログセンサ	アナログセンサ左	電圧入力(0～5V)		入力
0	アナログセンサ	アナログセンサ右	電圧入力(0～5V)		入力

■ポート 8

bit	接続先の基板	信号名	0	1	入出力方向 マイコンから見て
7	モータドライブ	ディップスイッチ または、LED	■入力 ディップスイッチの ON/OFF データ入力 ■出力 LED の点灯/消灯データの 出力		入出力※
6	モータドライブ	ディップスイッチ または、LED			入出力※
5	モータドライブ	ディップスイッチ または、LED			入出力※
4	モータドライブ	ディップスイッチ または、LED			入出力※
3	モータドライブ	ディップスイッチ または、LED			入出力※
2	モータドライブ	ディップスイッチ または、LED			入出力※
1	モータドライブ	ディップスイッチ または、LED			入出力※
0	モータドライブ	ディップスイッチ または、LED			入出力※

※プログラムで入出力を切り替えて使用します

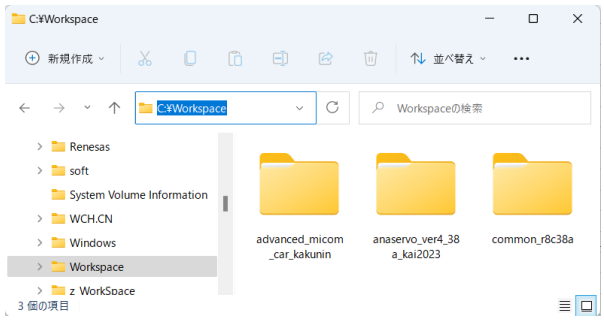
■ポート 9

bit	接続先の基板	信号名	0	1	入出力方向 マイコンから見て
7	端子なし	−	−	−	−
6	端子なし	−	−	−	−
5	モータドライブ	プッシュスイッチ	ON	OFF	入力
4	モータドライブ	ポート 8 のディップスイッチ、LED 切り替え	スイッチ	LED	出力
3	モータドライブ	右前モータ	ブレーキ	フリー	出力
2	モータドライブ	左前モータ	ブレーキ	フリー	出力
1	モータドライブ	右後モータ	ブレーキ	フリー	出力
0	モータドライブ	左後モータ	ブレーキ	フリー	出力

3. マイコンカーの動作確認

各種センサ出力の確認、ステアリングモータの動作の確認は、専用のプログラムで行います。プログラムのインストール、確認手順を説明します。

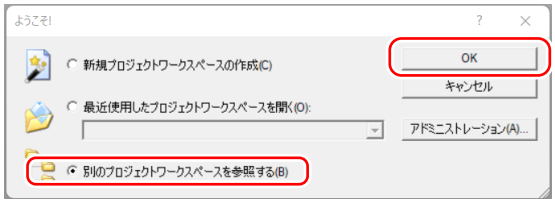
3.1. ワークスペースの展開



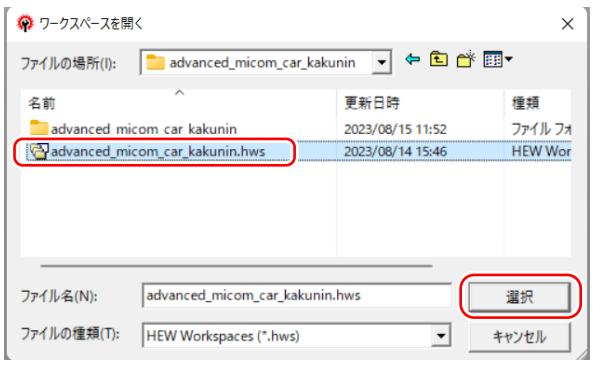
「workspace.zip」を、「C:¥Workspace」フォルダなどに解答します。
「advanced_micom_car_kakunin」が、動作確認用プログラムです。
「anaservo_ver4_38a_kai2023」が、走行用プログラムです。
「common_r8c38a」フォルダには、include するファイル（液晶や microSD など）が入っているフォルダです。

3.2. プログラムの転送と設定の初期化

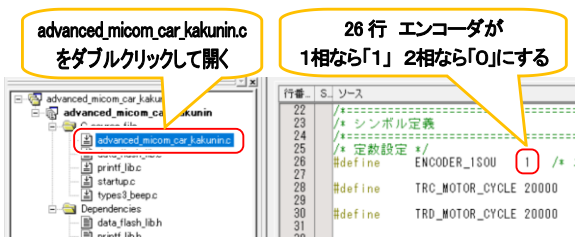
ルネサス統合開発環境を立ち上げます。



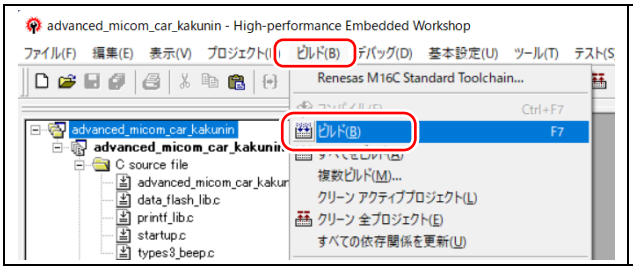
「別のプロジェクトワークスペースを参照する」を選択し、OK をクリックします。



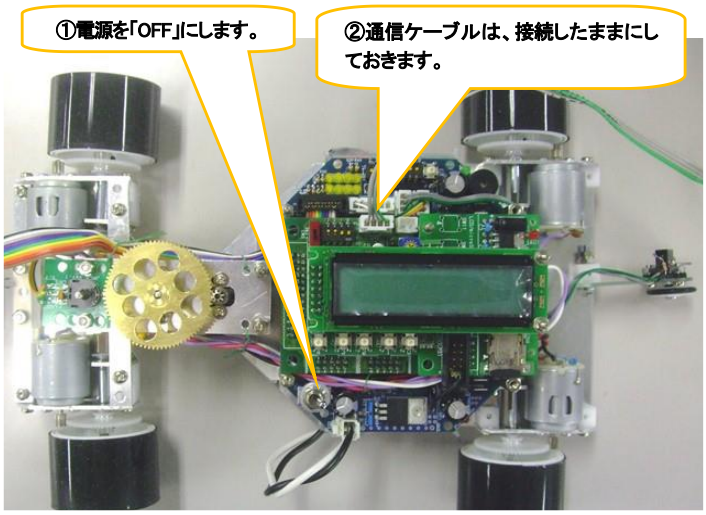
「advanced_micom_car_kakunin」フォルダの中にある「advanced_micom_car_kakunin.hws」を選択します。



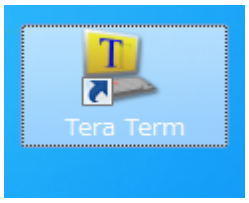
「advanced_micom_car_kakunin.c」をダブルクリックして開き、26行目を書き換えます。
●ロータリエンコーダが1相なら「1」にします。
（変更しなくてもよい）
●ロータリエンコーダが2相なら「0」にします。



「ビルド→ビルド」(F7キー)でビルドを実行し、
「0 Errors, 0 Warnings」か確認してください。
エラーがなければ「ツール→R8C Writer」を起動して、プログラムを転送してください。(**※プログラムの転送手順については、「ルネサス統合開発環境操作マニュアル R8C/38A 版」の「5 書き込み」を参照してください。**)



プログラムの転送が終わったあと、Advanced マイコンカーの電源を切り、書き込みケーブルは接続したままにしておきます。

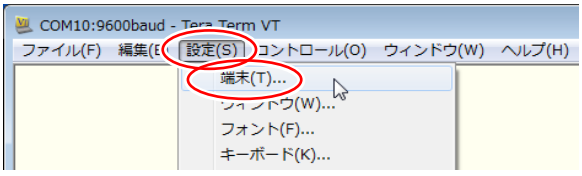


シリアル通信ソフトとして、Tera Term を使用します。
インストールされていない場合は、
https://ja.osdn.net/projects/ttssh2/releases/
からダウンロード、インストールしてください。
「Tera term」を実行します。

シリアルを選択します。

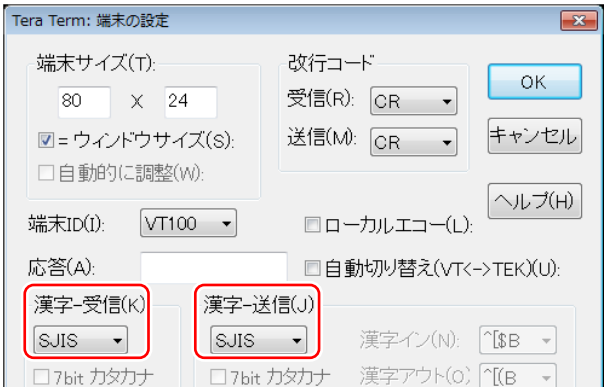
ポートは、「Prolific USB-to-Serial Comm Port」と書かれた番号、または接続しているシリアル番号にします。

OKをクリックします。

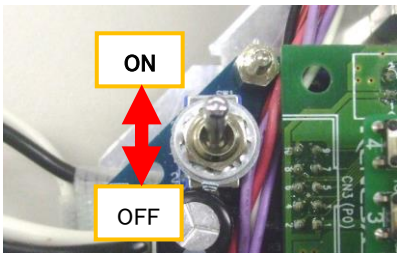


日本語が文字化けするので設定をします。

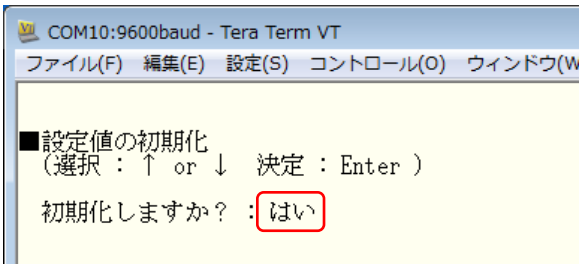
「設定→端末」を開きます。



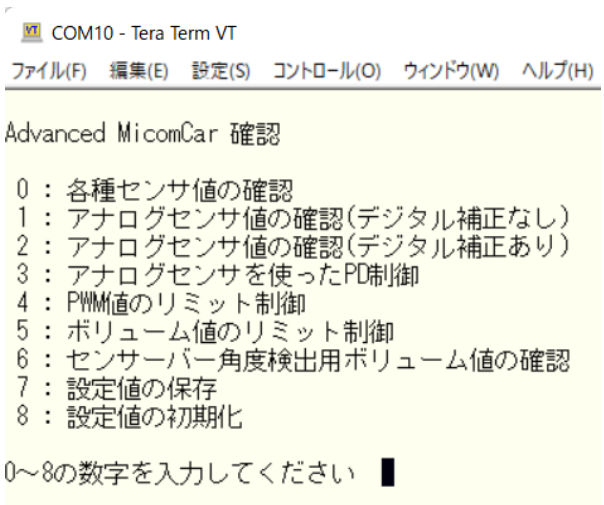
□で囲んだ部分 2 箇所を「UTF-8」から「SJIS」にします。設定ができたら OK をクリックし、設定は完了です。



Advanced マイコンカーの電源を ON にします。

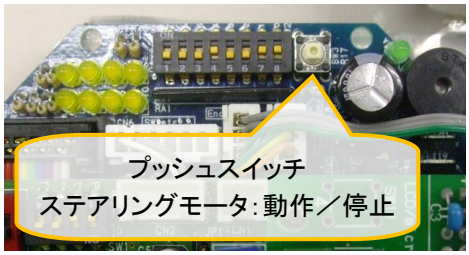


Advanced マイコンカーの電源を入れたときに「初期化しますか？」と表示された場合は「はい」を選択し、初期化してください。



初期化すると、「メニュー」画面が表示されます。

操作は、各メニューに対応する数字キーを押すと、確認をすることができます。



パラメータの調整が終わったあと、ステリングモータを動かし、動作確認をします。

モータドライブ基板のプッシュスイッチを押すとステアリングモータが動作し、もう一度プッシュスイッチを押すと、動作が停止します。

3.3. アナログセンサとデジタルセンサの確認

3.3.1. アナログセンサ基板の仕様

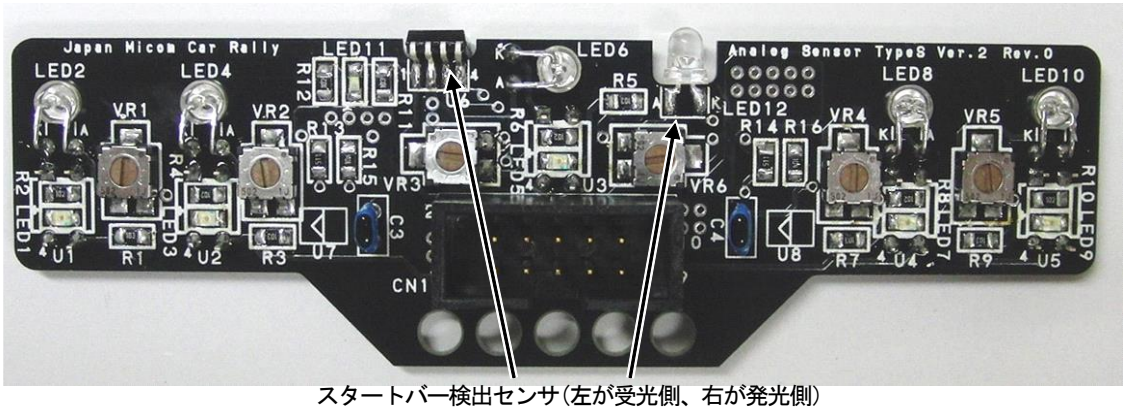
アナログセンサ基板 TypeS Ver.2 および Ver.3 の仕様を、下記に示します。

名称	アナログセンサ基板 TypeS Ver.2 および Ver.3
コースを見るデジタルセンサの個数	5 個（浜松ホトニクス製　光変調型フォト IC S7136）
コースを見るアナログセンサの個数	2 個 Ver.2 は、シャープ製の GP2S700 を使用 Ver.3 は、コーデンシ製の SG-105 を使用
スタートバーを見るセンサの個数	1 個（浜松ホトニクス製　光変調型フォト IC　S6846）
電圧	DC5.0V±10%
レジスト（基板色）	黒色
基板寸法	W94×D28×厚さ 1.0mm
部品実装時の寸法（実測）	最大 W94×D28×H13mm

3.3.2. 外観

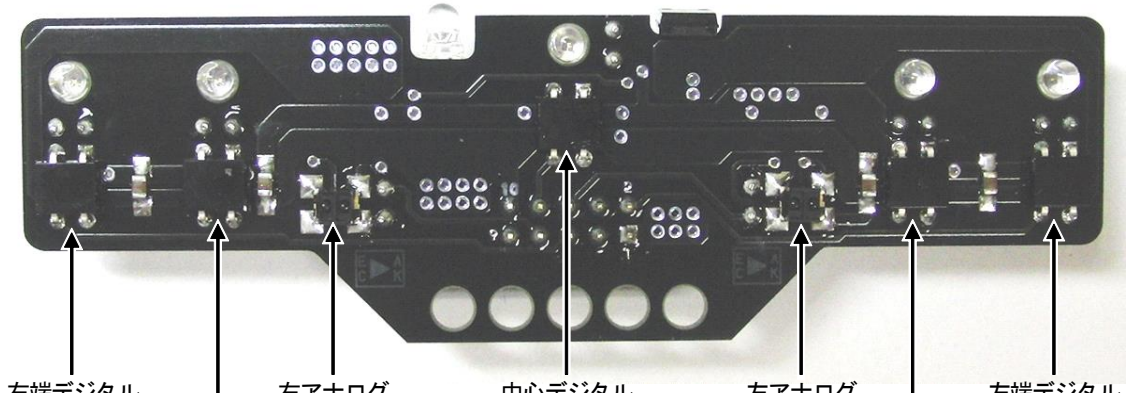
※アナログセンサ基板 Ver.2 の外観を下記に示します。

部 品 面



スタートバー検出センサ（左が受光側、右が発光側）

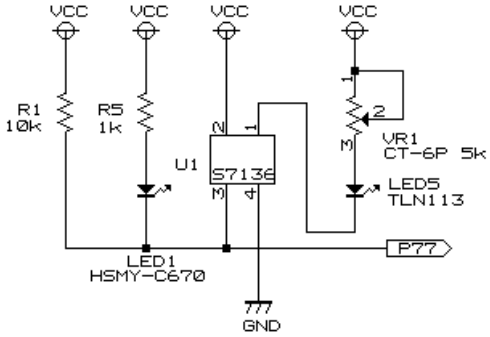
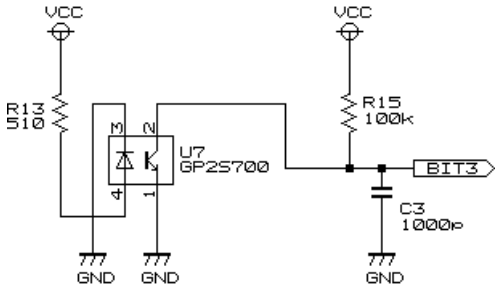
半　田　面



※半田面は、裏から見ているため左右が逆になります。

3.3.3. デジタルセンサとアナログセンサ

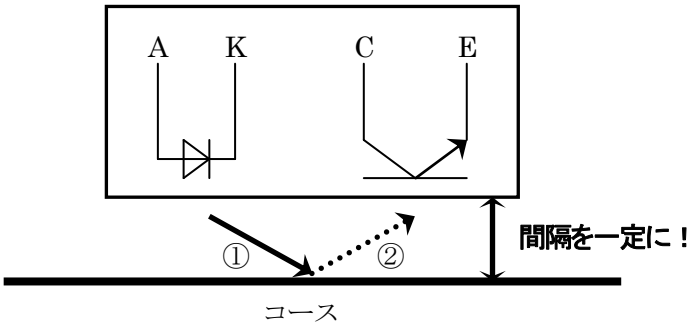
マイコンカーで使用する場合の、デジタルセンサとアナログセンサの特徴を下記に示します。

項目	デジタルセンサ	アナログセンサ
回路例	 センサのピン振り 1:赤外 LED のカソード　2:＋電源 3:出力　4:GND	 センサのピン振り 1:エミッタ　2:コレクタ 3:カソード　4:アノード
センサ出力	センサ下部が白色(灰色)か黒色かの判断	センサ下部が、白色か灰色か黒色か、さらに黒に近い灰色か、白に近い灰色かなど、細かく検出可能
外乱	強い、S7136 は特に強い	非常に弱い
コースとの間隔	2mm～10mm、幅が広い	約 2～4mm　常に一定にする必要がある
ポート	センサからの信号はデジタル出力なので、マイコンのどのポートでも入力可能	センサからの信号はアナログ出力なので、マイコンのアナログ入力端子のみで入力可能

アナログセンサはデジタルセンサに比べ、外乱に弱いですがコースの状態を細かく知ることができます。この情報をうまく使えばステアリング制御を非常に細かくできるため、ライントレースを非常に滑らかに行うことができます。

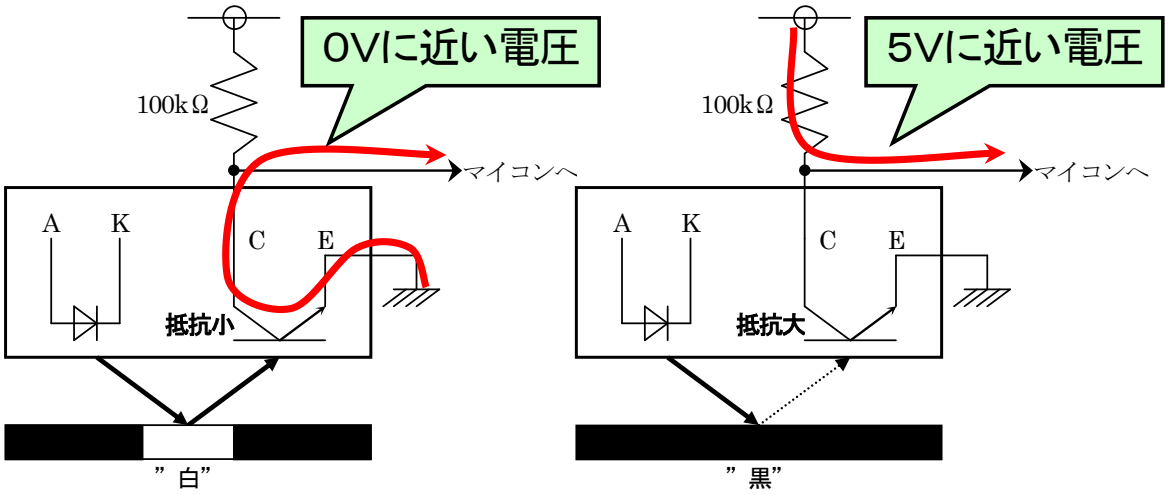
3.3.4. アナログセンサの動作原理

アナログセンサは、赤外 LED とフォトトランジスタの 1 組で構成されています。赤外 LED から出力される光りをコースに当てて、その反射光をフォトトランジスタで受けます（下図）。このとき、白色は光りを反射、黒色は吸収することを利用します。センサ下部が白色なら、赤外 LED から出た光は多く反射して、フォトトランジスタに届きます。黒色なら反射が少ないのでフォトトランジスタにあまり届きません。灰色はその中間です。



① 赤外 LED 側から赤外線を出します。
② コースに反射した光をフォトトランジスタで受けます。

コースの色と、フォトトランジスタの出力電圧の関係を下記に示します。



センサ下部が白色なら、フォトトランジスタに多くの光りが届くのでエミッターコレクタ間の抵抗が少なり、マイコンへは低い電圧が出力されます。
センサ下部が黒色なら、フォトトランジスタに光りがあまり届かないのでエミッターコレクタ間の抵抗が大きくなり、コレクタに接続されているプルアップ抵抗を通してマイコンへは高い電圧が出力されます。
灰色は、その中間です。

このように、電圧の違いで、黒色、灰色、白色を判断することができます。

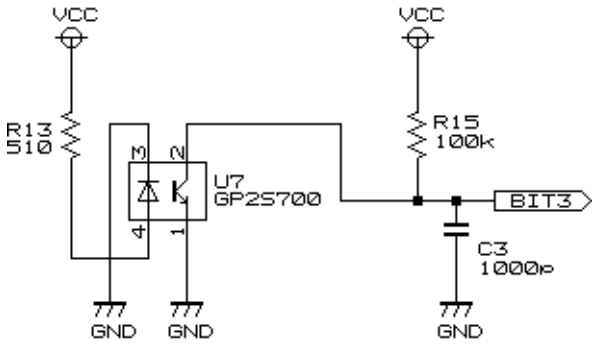
3.3.5. マイコンの A/D 変換で電圧をデジタル値で取り込む

R8C/38A マイコンには A/D 変換器が内蔵されていて、端子に入力されている電圧を知ることができます。R8C/38A マイコンの、A/D 変換器の特徴を下記に示します。

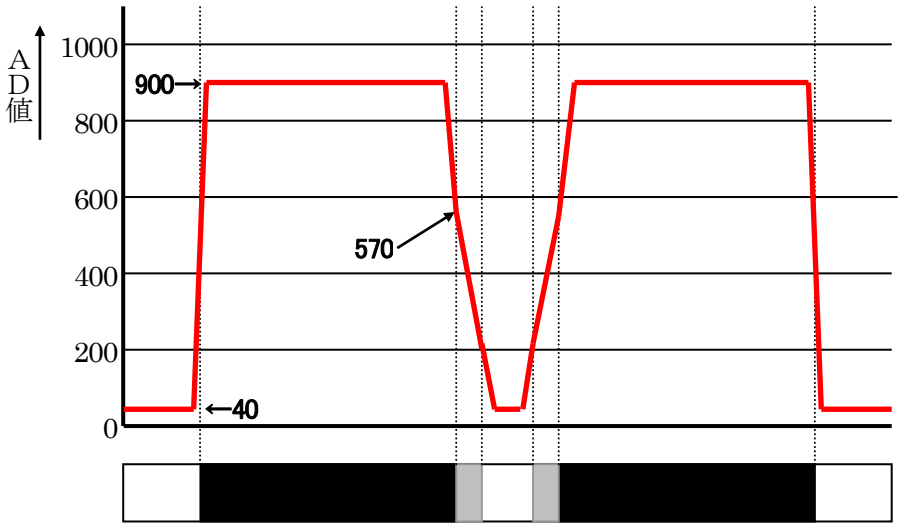
- アナログ入力端子は、ポート 0 の bit0～7、ポート 1 の bit0～3、ポート 7 の bit0～7 の 20 端子ある
  ※ただし、RY_R8C38 ボードはポート 1 の bit0～3 は、ディップスイッチに繋がっています。
- 0～5V(マイコンボードの電源電圧)の電圧を、0～1023(2^{10}-1)の値に変換する
- 1 度に A/D 変換できるのは 1 端子のみ、どの端子電圧を A/D 変換するかはプログラムで設定する

A/D 変換について、詳しくは「マイコン実習マニュアル(R8C/38A 版)」のプロジェクト「ad」を参照してください。

3.3.6. コースの状態を取り込む



上記のような回路を組み、コースとセンサの間隔を約 3mm 一定にしてコースの端から端まですらしていき、そのときの A/D 取得値を簡単なグラフにしてみました。



白色が 40、黒色が 900、中心の黒、灰、白部分は 900 から 40 へ変化していきます。正確には比例していませんが、ほぼ比例していると考えて差し支えありません。

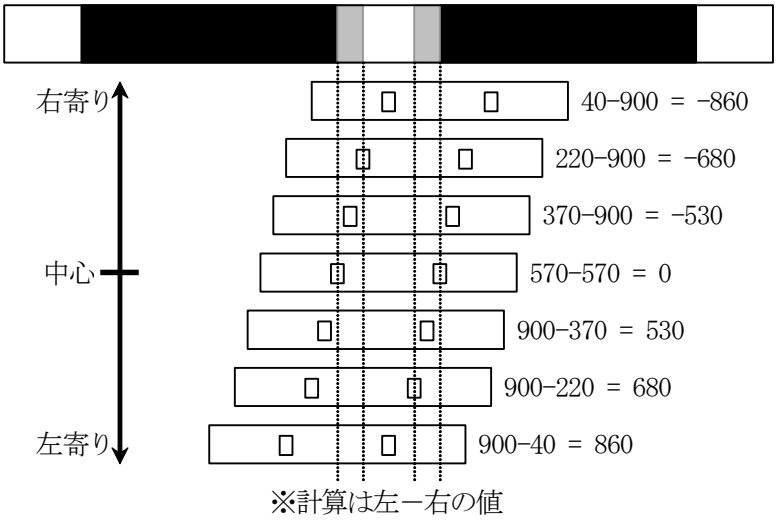
マイコンカーはコース中心の白色と灰色をトレースするので、この部分を詳しく見てみます。コース中心部分は 40、ずれるにしたがって値が大きくなり、黒色部分では約 900 になります。A/D 値をチェックすることにより、中心部分のずれを細かく知ることができます。

ただし、1 つ問題があります。例えば黒色と灰色のちょうど境目は 570 くらいですが、数字を見ただけでは右側の境目なのか左側の境目なのかが分かりません。アナログセンサ 1 個では右にずれているのか左にずれているのか分からないのです。

そこで、アナログセンサを 2 個、40mm の間隔で取り付け、次の計算を行います。

センサの値 ＝ 左センサの値 － 右センサの値

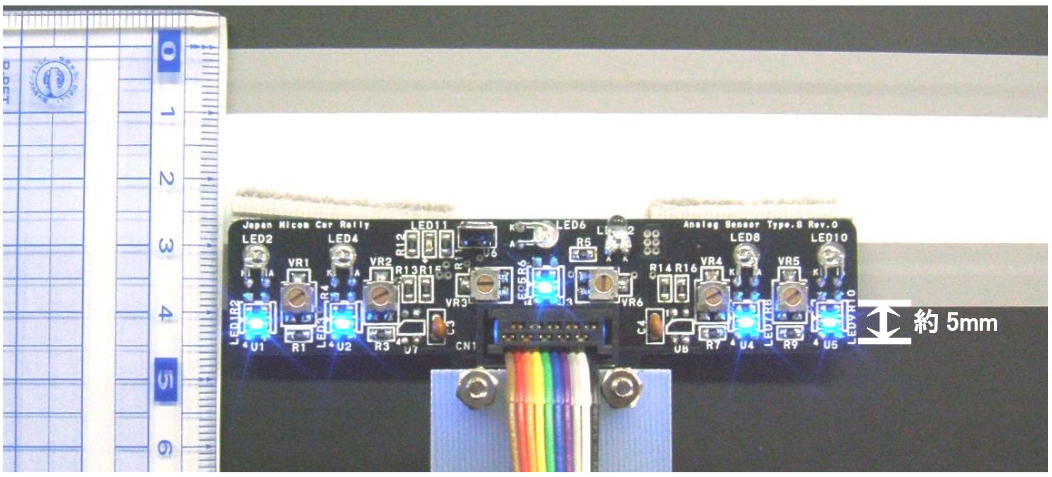
この計算を行うことによりセンサの値は、左側にずれているなら正の数、右側なら負の数となり、左右のどちら側に寄っているのか分かります。



今回の例では、センサの値が−860 から 860 まで変化します。センサの状態が正の数なら左に寄っていますのでハンドルを右へ、負の数なら右へ寄っていますのでハンドルを左へ曲げます。また値の大きさで、どのくらいの強さで曲げるかを調整することができます。例えば 50 なら弱く曲げる、800 なら強く曲げる、というように制御します。

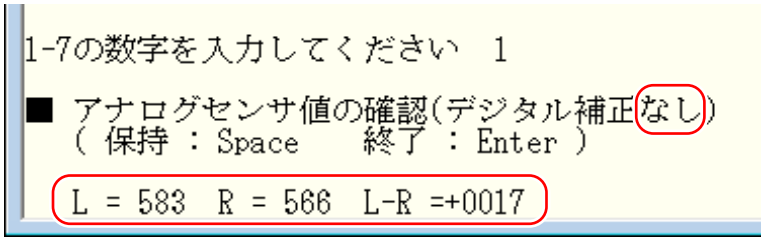
3.3.7. デジタルセンサの調整

写真のように、アナログセンサ基板をコースの上に置き、デジタルセンサ下部が、灰色・白色で反応するようにしてください。5個とも調整してください。

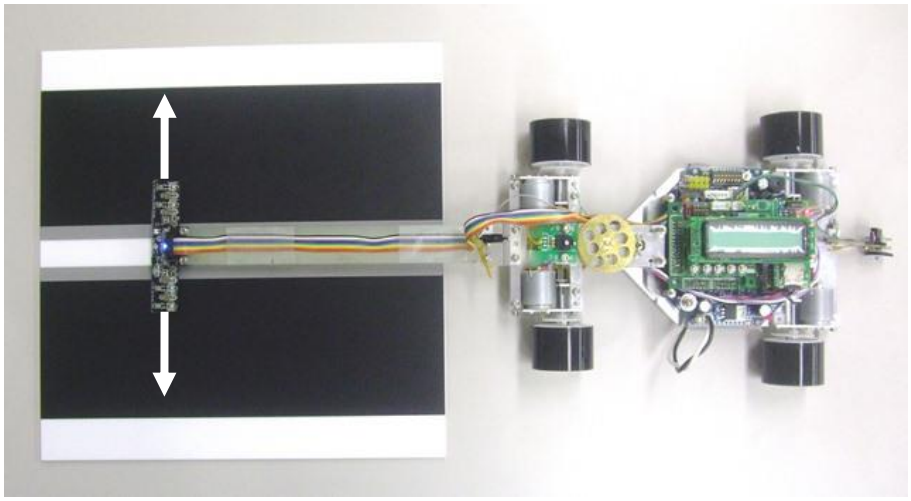


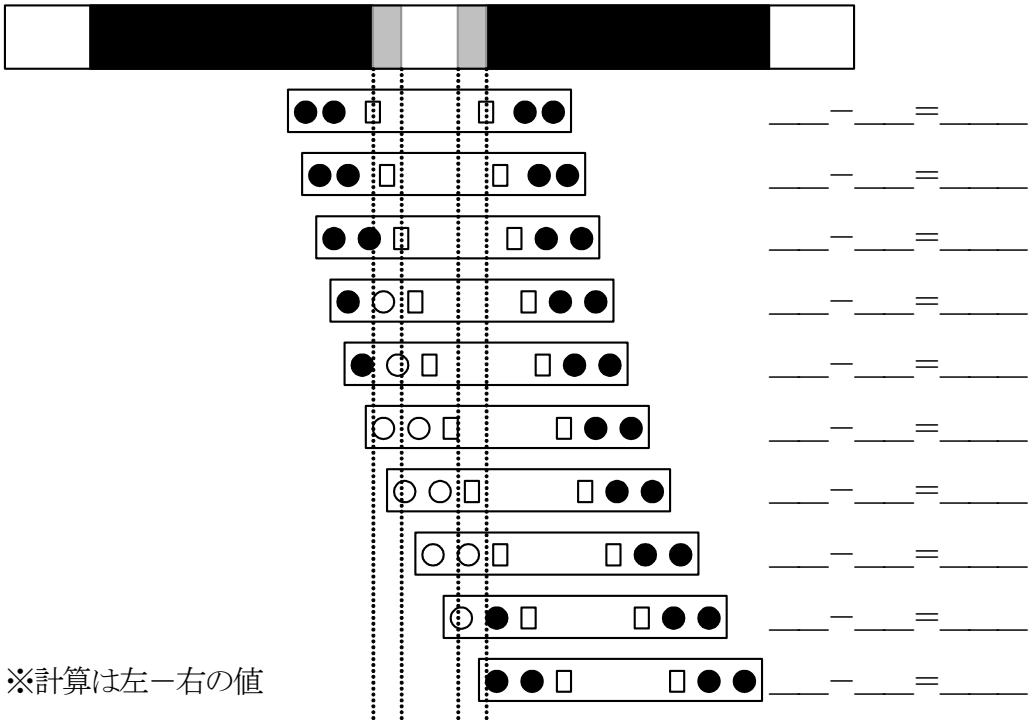
3.4. アナログセンサ値の確認（デジタル補正なし）

TeraTerm を立ち上げて、シリアル接続してください。マイコンカーの電源を入れるとメニューが表示されます。
メニュー画面で「1」キーを押し、「アナログセンサ値の確認（デジタル補正なし）」画面に切り替えます。



このとき、ハンドルを左右に動かし、アナログセンサの値がどうなるか、確かめてみてください。

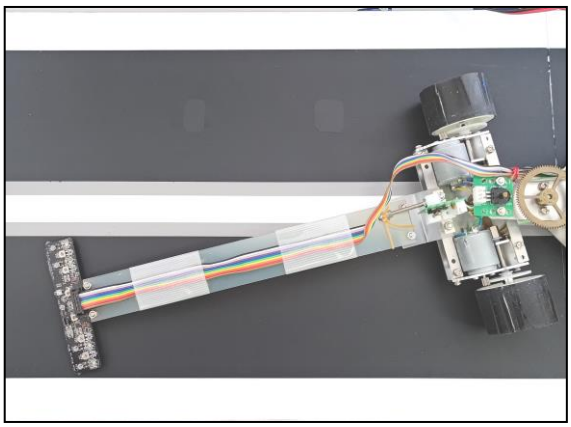




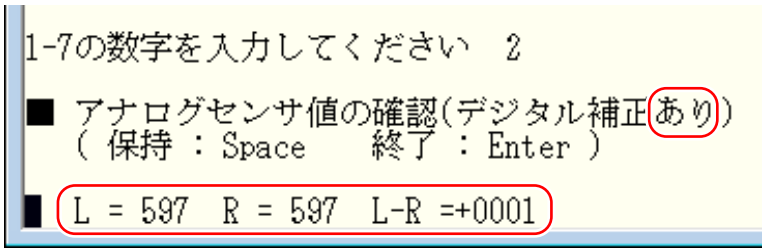
※□はアナログセンサ、●はデジタルセンサ反応無し、○はデジタルセンサ反応あり　の状態を表しています。

右写真のように、アナログセンサ基板がコース中心から大きくずれた場合、アナログセンサの右も左も黒色になるため、「左－右＝ほぼ0」になってしまい、コース中心の0と違いが分からなくなります。

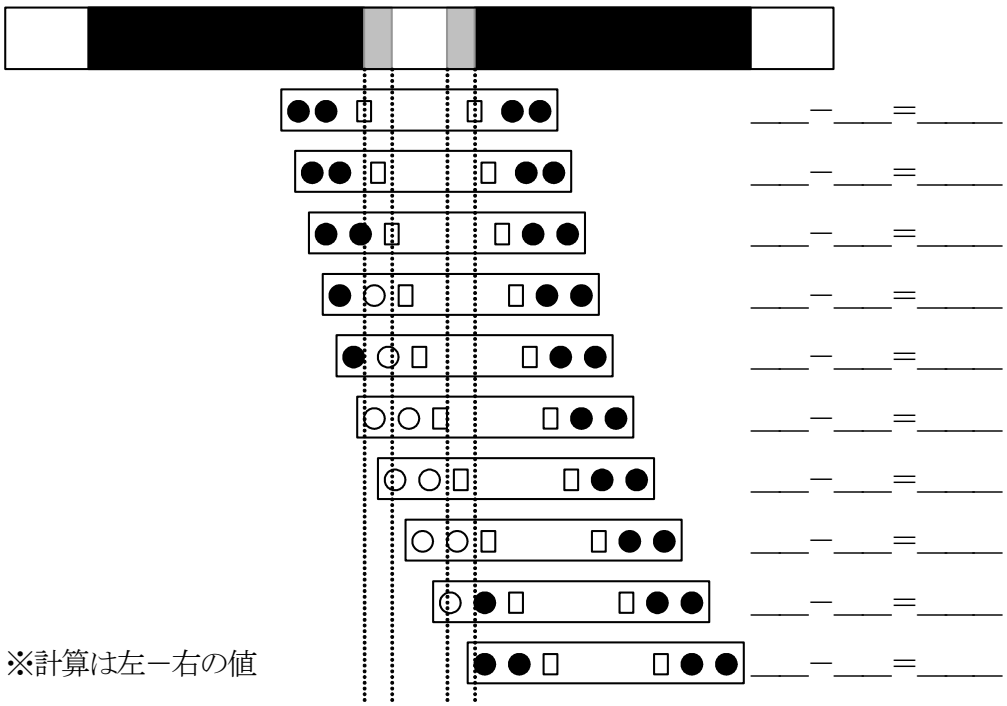
そこで次で説明する「デジタル補正あり」で、デジタルセンサが反応したらコース中心から大きくずれているので、「左－右」の値を使わずに、直接700や800などの値をアナログセンサの差分の値として使うようにします。こうすることにより、コース中心から大きくずれても、中心に戻るようにステアリングモータを制御することができます。



メニュー画面で「2」キーを押し、「アナログセンサ値の確認（デジタル補正あり）」画面に切り替えます。



デジタルセンサが反応したら、アナログセンサ値は使わずに直接、数値をセンサ値として補正しています。アナログセンサの値がどうなるか確かめてみてください。特に、デジタルセンサが点灯したときのアナログセンサの値を確認してください。



※□はアナログセンサ、●はデジタルセンサ反応無し、○はデジタルセンサ反応あり　の状態を表しています。

3.5. ステアリングモータの PD 制御

本マニュアルでは、「**マクソンモータ ＝ ステアリングモータ ＝ サーボモータ**」として扱います。
※ステアリングとは・・・・・乗り物の進行方向を任意に変えるためのかじ取り装置のこと。(出典:wikiペディア)
※サーボモータとは・・・指示を出した通りに、位置／速度／回転力(トルク)などを正確に実現するサーボ機構に使用されるモータのこと。(出典:fabcross for エンジニア)

3.5.1. PID 制御とは？

自動制御方式の中でもっとも良く使われる制御方式に PID 制御という方式があります。この PID とは
Ｐ：Proportinal（比例）
Ｉ：Integral（積分）
Ｄ：Differential（微分）
の３つの組み合わせで制御する方式で、きめ細かくステアリングモータの PWM を調整してスムーズな制御を行うことができます。
PID 制御についての詳細は、ホームページや書籍が多数出ていますのでそちらを参照してください。

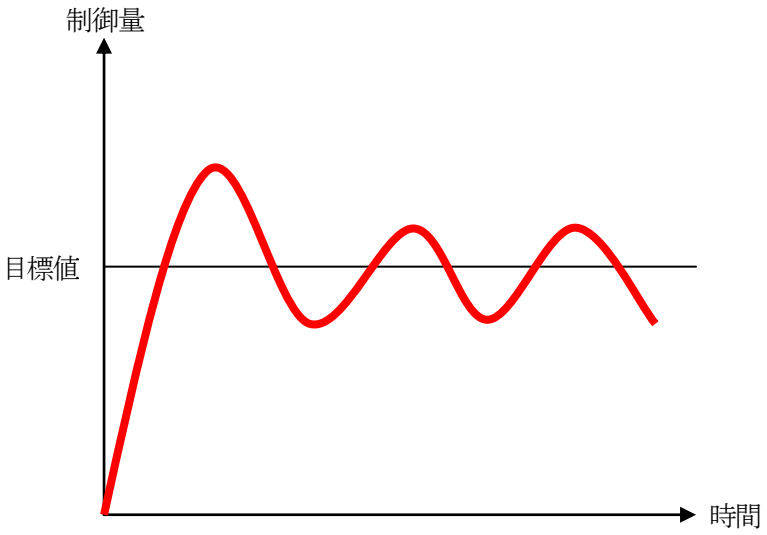
今回、ステアリングモータの制御は比例制御と微分制御を行います。PD 制御と呼びます。

3.5.2. P(比例)制御

比例制御とは、目標値からのずれに対して比例した制御量 P を与えます。P を計算する式は下記のようになります。

制御量 P＝kp×p
kp ＝ 定数
p ＝ 現在の値 － 目標の値
＝ 現在のアナログセンサ値 － 目標のアナログセンサ値
＝ getAnalogSensor() － 0
＝ getAnalogSensor()

目標のアナログセンサ値は、ちょうどコースの中心の値である 0 になります。
制御としては早く目標値に近づけたいので、ずれが大きいほどステアリングモータの PWM を多くします。そのため、目標値に到達しても速度を落としきれず、目標値をいったりきたりと振動してしまいます。

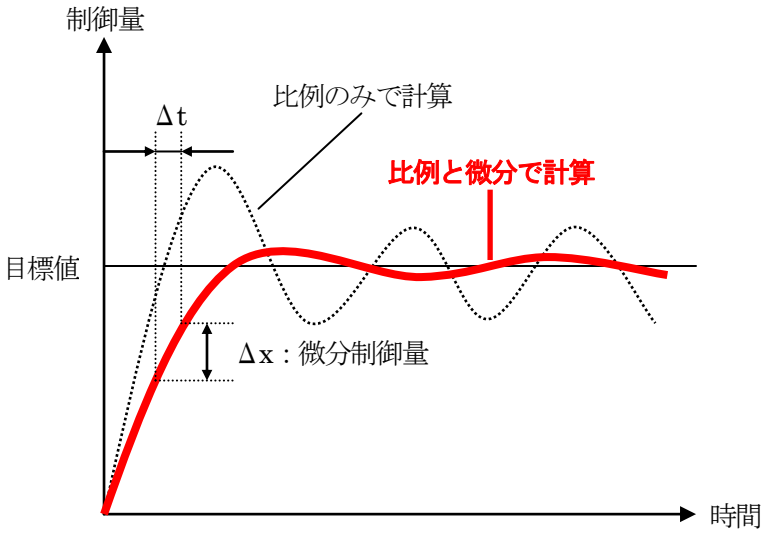


3.5.3. D（微分）制御を加える

微分制御とは、瞬間的な変化量を計算して**比例制御を押さえるような働き**をします。微分制御量 D を計算する式は下記のようになります。

制御量 D＝kd×d
kd　＝　定数
d　＝　過去のアナログセンサ値　－　現在のアナログセンサ値
　＝　iSensorBefore　－　getAnalogSensor()

過去のアナログセンサ値を iSensorBefore というグローバル変数に保存しておきます。
比例制御のみで振動していても、微分量を加えると振動を抑えることができます。ただし、比例制御を押さえる働きをしますので、目標値に近づく時間は長くなります。時間は数 ms～数十 ms のレベルです。それが、実際の走りに対して、どう影響するかは検証する必要があります。



3.5.4. 最終制御量

ステアリングモータに加える制御量を計算する式は下記のようになります。

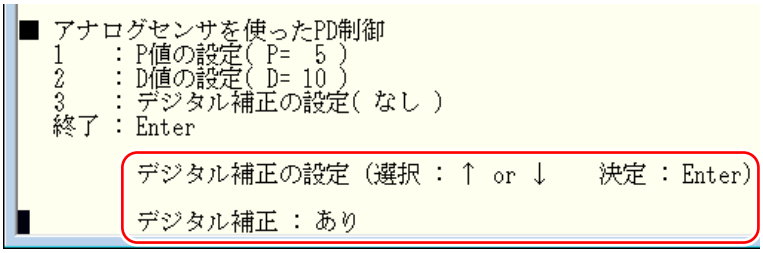
最終制御量　＝　P値　－　D値

プログラムでは、最終制御量に定数をかけて PWM 値に調整します。

ステアリングモータに大きい PWM を加えると、マクソンモータが壊れたり、ステアリング部のギヤが壊れたりしまうので、PWM の上限を設けます。サンプルプログラムは、90%以上にならないようにしています。

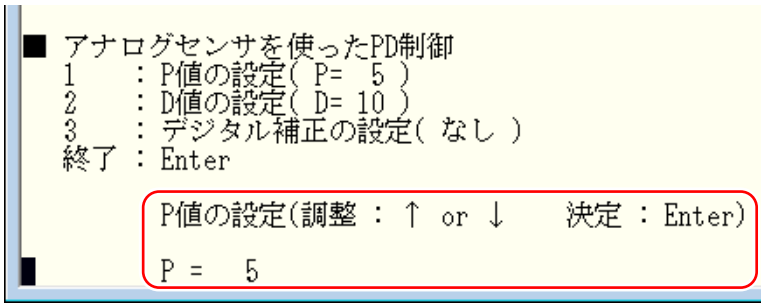
3.6. PD値の調整

PD 制御実験を行い、P 値、D 値の値を決めます。後ほど、走行プログラムに P 値、D 値を入力します。
まず、メニュー画面で「3」キーで「アナログセンサを使った PD 制御」を選択します。
「3」を選択し、デジタル補正「あり」に設定します。

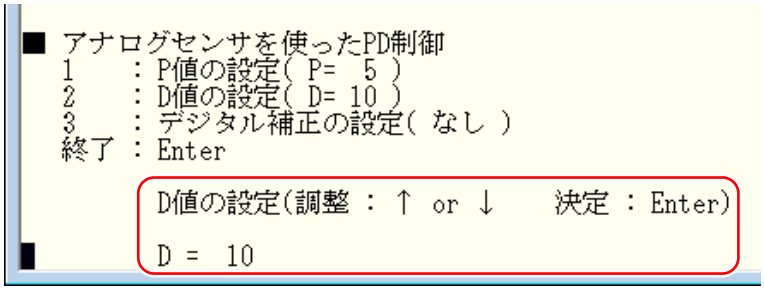


後は、下記①～③を繰り返し実行し、自分のマイコンカーにあった P 値、D 値を見つけてください。

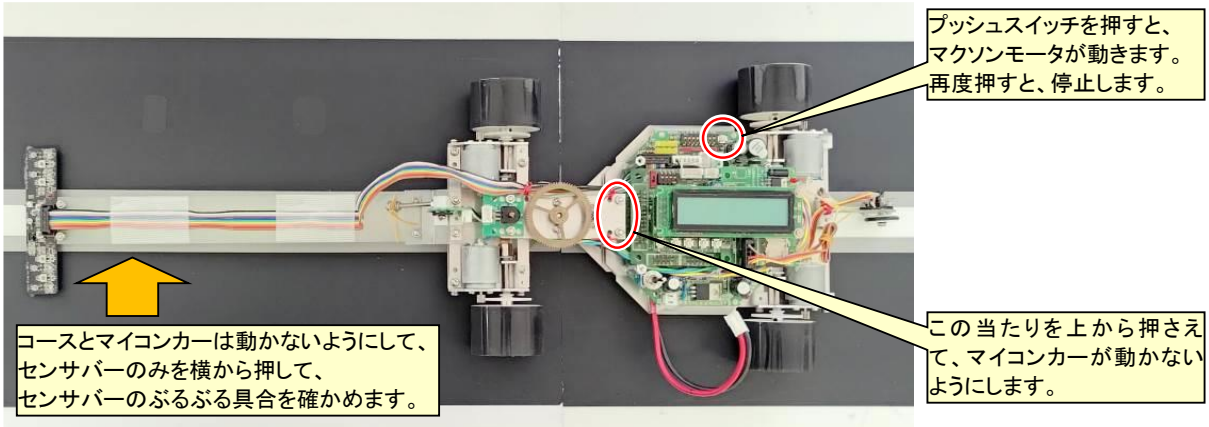
①メニュー画面で「1」キーを押し、P値（比例制御の定数）を設定します。



②メニュー画面で「2」キーを押し、D値（微分制御の定数）を設定します。



③モータドライブ基板 TypeS のプッシュスイッチを押すと、マクソンモータが動いて、PD 制御を行います。センサバーを横から押して、センサバーのぶるぶる具合を確かめます。コース中心に力強く戻り、かつぶるぶるが少ないのが理想です。おおむね D 値は、P 値の5～10倍がお勧めです。

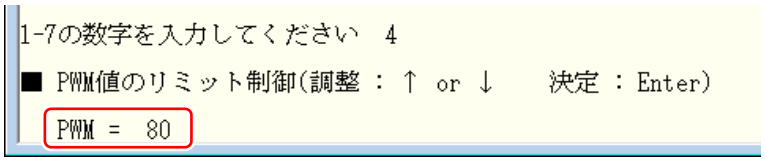


3.7. PWMのリミット制御

ステアリングモータに 100%の PWM を加えると、モータに高負荷がかかり、モータを傷めたりギヤを痛めたりします。例えば PD 制御で計算した PWM 値が±90%以上になった場合は、プログラムで上限を±90%にすることができます。リミット(限度)制御と呼びます。

リミット制御の実験をすることができます。メニュー画面で「4」キーを押し、ステアリングモータに加える PWM の上限を設定します。設定した値以上の PWM を加えなくなります。走行プログラムでは、90%にしています。

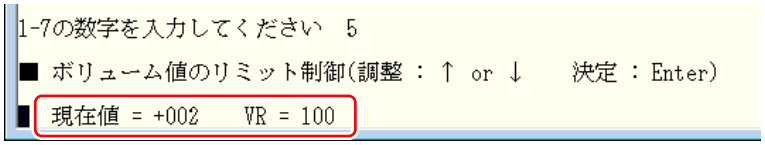
上限の PWM 値を設定してモータドライブ基板 TypeS のプッシュスイッチを押しステアリングモータを動作させます。手で無理矢理センサを中心からずらします。ステアリングモータの力のかかり方がどうなるか確かめてください。



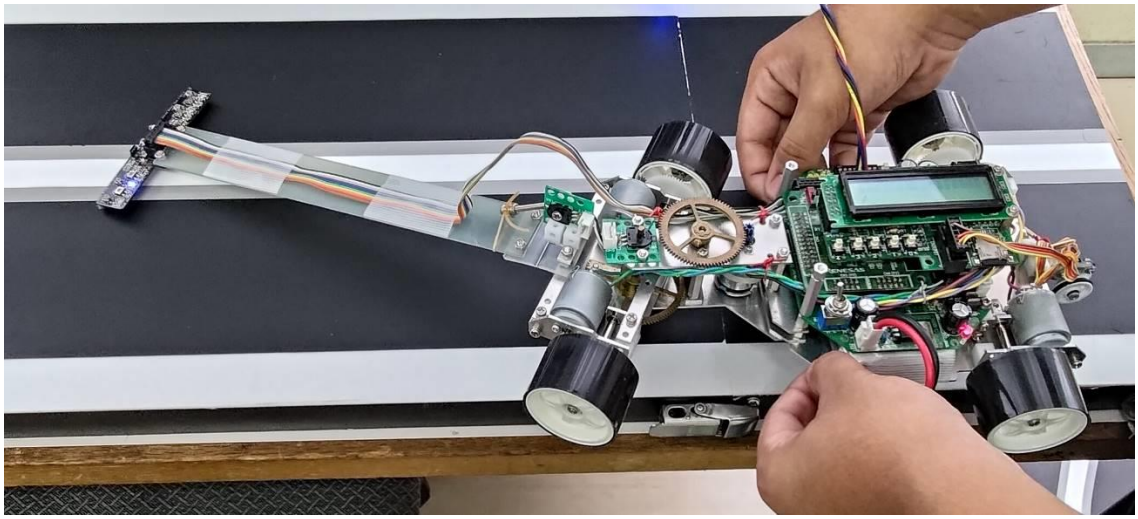
3.8. 角度のリミット制御

ハンドルを切りすぎるとフロント部分がシャーシにぶつかり止まってしまいます。止まってもステアリングモータを回し続けると、モータやギヤに無理がかかり、壊してしまうかもしれません。そのため、角度のポテンショメータ(ボリューム)の A/D 値を見て、設定以上の角度になると、PWM を下げてモータやギヤにあまり無理がかからないようにします。

メニュー画面で「5」キーを押し、ボリュームのリミット値を設定します。ボリュームの A/D 値が設定以上になると、ステアリングモータに加える PWM 値を 0%にします。



ボリューム値を設定して、モータドライブ基板 TypeS のプッシュスイッチを押しステアリングモータを動作させます。このとき、センサはコースに置いた状態で、マイコンカー本体のみを手で持ち、マイコンカーを左右に動かします。ハンドルの角度が大きくなっていくと、ステアリングモータの力のかかり方がどうなるか確かめてください。

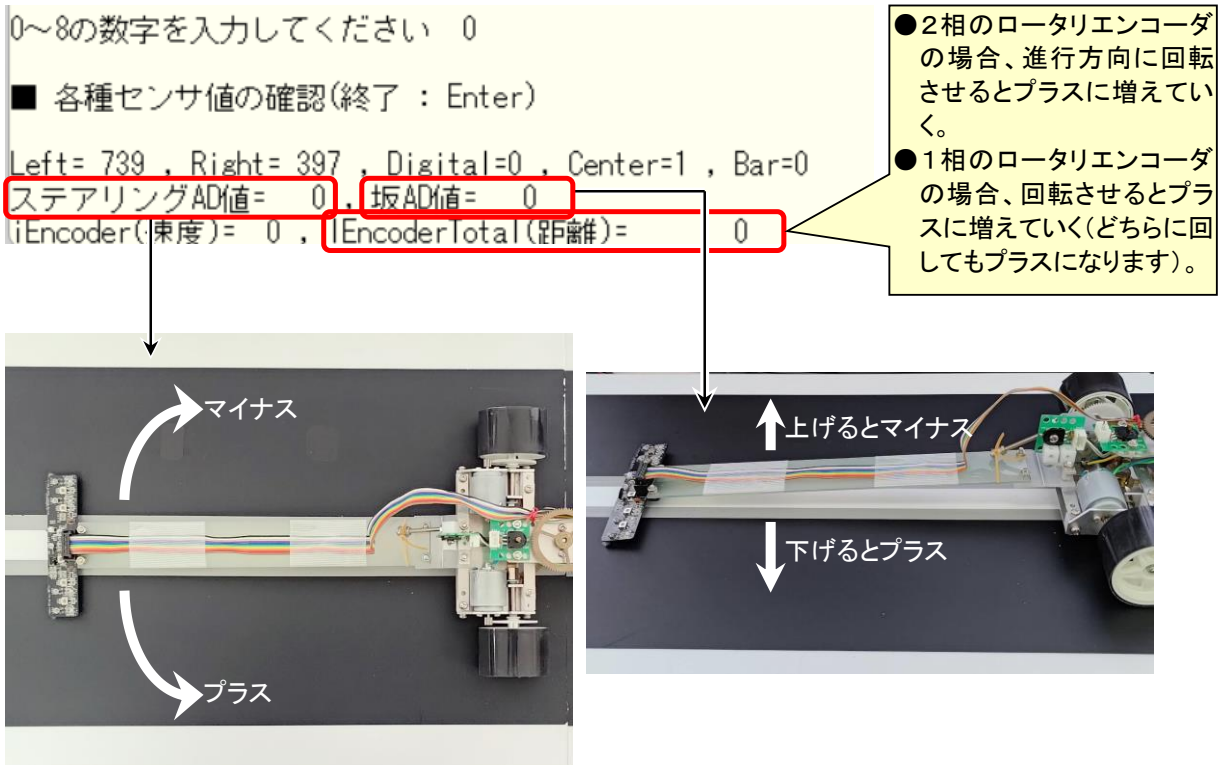


▲確認している様子

3.9. 角度検出用ポテンショメータ、坂道用ポテンショメータ、エンコーダ値の確認

※液晶・microSD 基板がある場合は、次の走行プログラムで確認できるので、ここでの確認は不要です。

マイコンカーのステアリングを0度にした状態で、電源を入れます。
メニュー画面で「0」キーを押し、下記のステアリング AD 値、坂 AD 値、lEncoderTotal(距離)を確認します。



●ステアリング AD 値のプラスマイナスが逆な場合
ステアリング角度検出用のポテンショメータ基板の1ピンと3ピンの配線を入れ替えます。

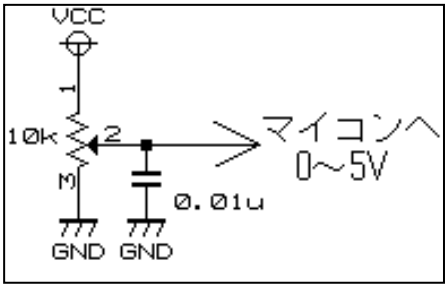
●坂 AD 値のプラスマイナスが逆な場合
坂道用のポテンショメータ基板の1ピンと3ピンの配線を入れ替えます。

測定した AD 値を記入しておきます。**※液晶・microSD 基板がある場合は、次の走行プログラムで確認できるので、ここでの確認は不要です。**

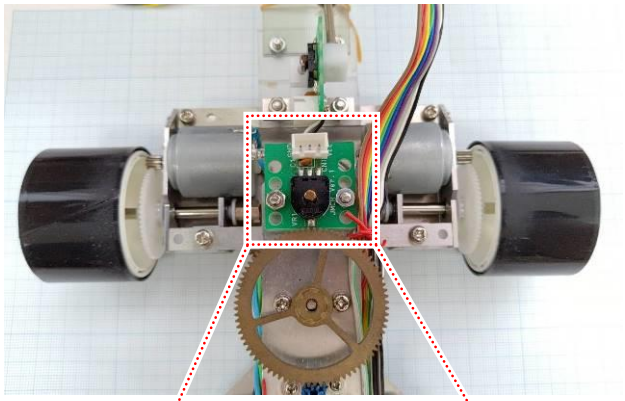
ハンドルを止まるまで左にしたときのステアリング AD 値	
ハンドルを止まるまで右にしたときのステアリング AD 値	
上り坂にさしかかり、センサバーが一番上がったときの AD 値	
上り坂が終わり、センサバーが一番下がったときの AD 値	

※角度検出用ポテンショメータの機構について

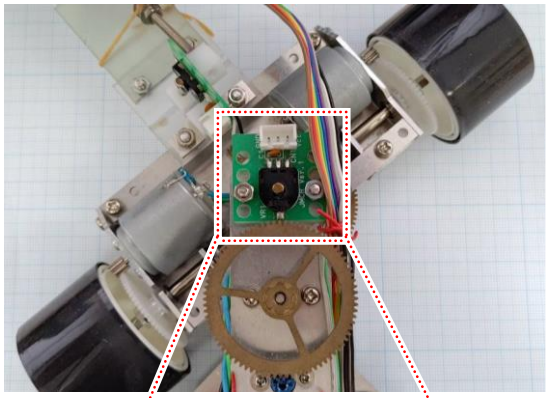
ステアリング角度検出用ポテンショメータとして、アルプス電気製の「RDC503013A」を使用しています。
ポテンショメータは可変抵抗で、右図の回路になっていて、中心部分が回転することにより 0～5V をマイコンへ出力し、マイコンの A/D 変換で 0～1023 に変換します。
ハンドルをまっすぐにしたときにポテンショメータの中心になるように合わせると約 2.5V が出力されます。ハンドルを左右に切ったときにポテンショメータの軸が回転し、出力電圧が変わります。

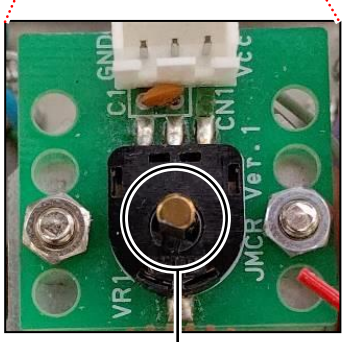


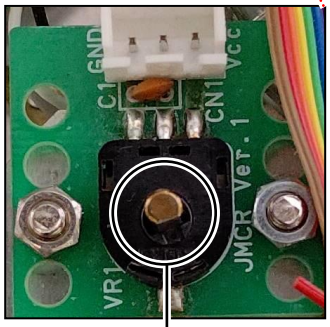
まっすぐのとき



左いっぱいに曲げたとき



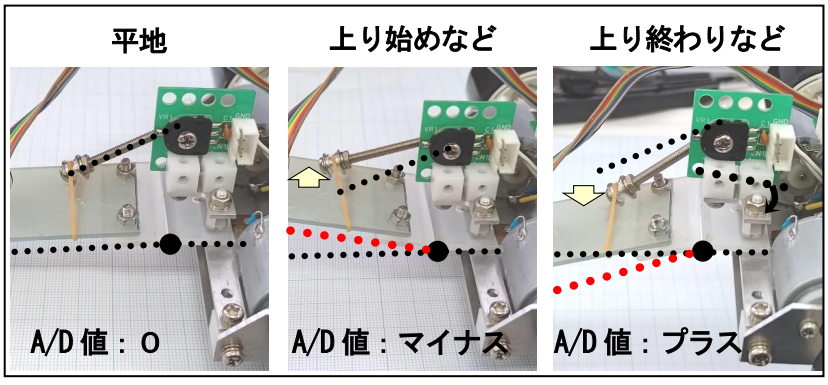




ポテンショメータの中心部分が回転して、出力電圧が変化する。
マイコンの A/D 変換器で電圧をデジタル値に変換し、角度を検出する。

※坂道用ポテンショメータの機構について

坂道を検出するためのポテンショメータです。センサバーが上下すると、ポテンショメータの軸が回転します。ポテンショメータから出力される電圧を、マイコンの A/D 変換器で 0～1023 に変換して、センサバーの状態を検出します。
回路は、ポテンショメータ基板（角度検出用）と同じです。

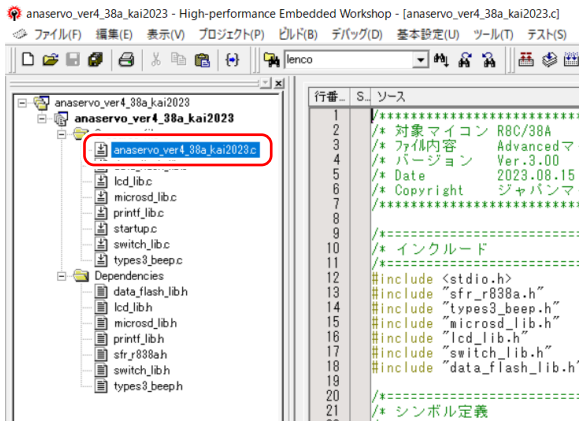


4. 走行プログラムの解説と調整

重要

本プログラムのロータリエンコーダの値(速度、距離)は、ロータリエンコーダ TypeS を基に作られています(秒速 1m/s のとき iEncoder=55、1m 進むと lEncoderTotal=5459)。それ以外のエンコーダの場合は、適宜、値を調整してください。

4.1. ワークスペースを開く



解凍したフォルダの「anaservo_ver4_38a_kai2023」フォルダの中にある「anaservo_ver4_38a_kai2023.hws」をダブルクリックすると開発環境が立ち上がります。
「anaservo_ver4_38a_kai2023.c」をダブルクリックして、プログラムを開きます。
プログラムの24行を開きます。

```
24 :  #define      ENCODER_1SOU     0
```

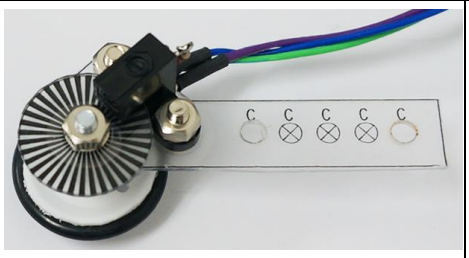
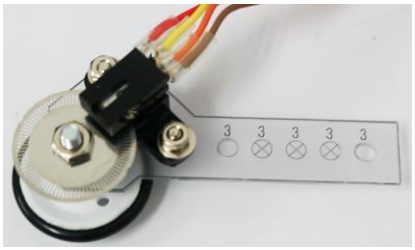
ロータリエンコーダがロータリエンコーダ Ver.2(1相)なら「**1**」に、ロータリエンコーダ TypeS(2 相)なら、「**0**」にしてください。
ビルドして、プログラムを書き込んでください。

4.2. ロータリエンコーダ値の調整

ロータリエンコーダは、走行しているときの速度と、スタートしてからの距離を測定することができます。直線やカーブで速度を一定にしたり、1周したあと自動的に停止させることができます。また、走行中に速度が0になったら、脱輪して止まっていると判断して、自動停止させることができます。

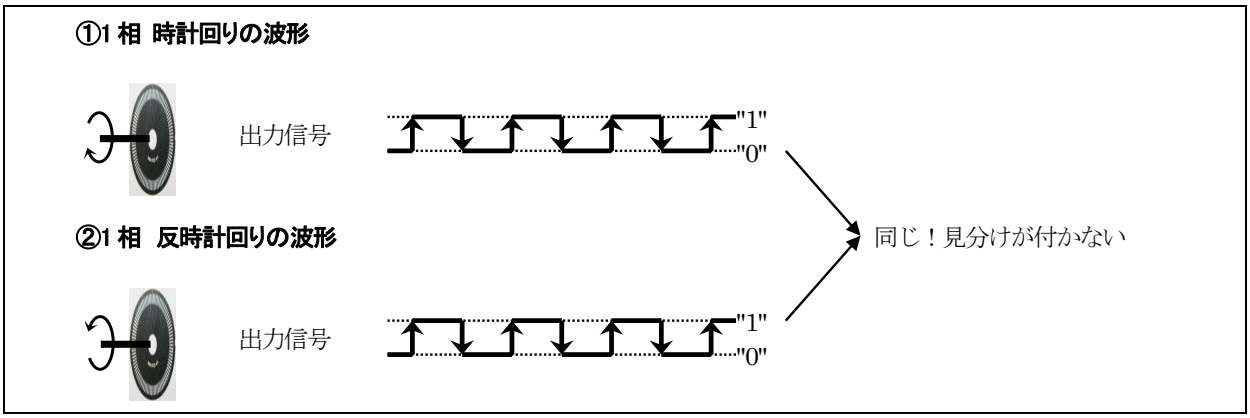
4.2.1. ロータリエンコーダの計算

Advanced マイコンカーで使用するロータリエンコーダ Ver.2 および TypeS の仕様を下記に示します。

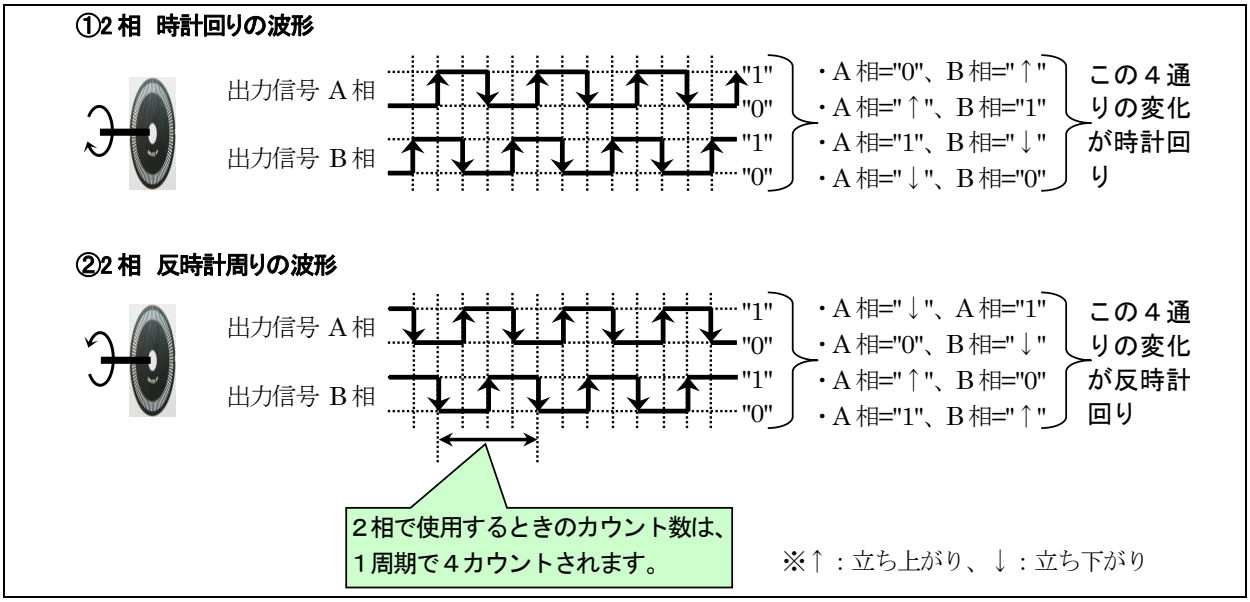
項目	ロータリエンコーダ Ver.2	ロータリエンコーダ TypeS
完成例		
相数	1相(A 相のみ)	2相(A 相、B 相)
ロータリエンコーダ 1 回転のパルス数	72 パルス／回転 ※スリットは36個ですが、立ち上がりと立下りを読み、2倍のカウントにしています。	360 パルス／回転 ※スリットは 90 個ですが、2 相なので 4 倍の 360 カウントになります。
タイヤの直径 (A)	21.0mm	21.0mm
円周 (B)	(A)× π ＝65.94mm	(A)× π ＝65.94mm
1 回転のパルス数 (C)	65.94mm 進むと 72 パルス	65.94mm 進むと 360 パルス
(C)より**1m進んだときのパルス数** (D)	65.94 : 72 = 1000 : x x = 約 **1092 パルス**	65.94 : 360 = 1000 : x x = 約 **5459 パルス**
秒速1m/s で進んだとき 1秒間のパルス数 (E)	約 1092 パルス／秒	約 5459 パルス／秒
秒速1m/sで進んだとき 10ms 間のパルス数 (F)	**約 11 パルス**	**55 パルス**

4.2.2. ロータリエンコーダの原理(1相と2相)

1 相出力のとき、ディスクが時計回り、反時計回りで回ったときの出力波形を下記に示します。1 相のときは、ディスクが時計回りか反時計回りか分かりませんが、マイコンカーは前進しかしませんので問題ありません。

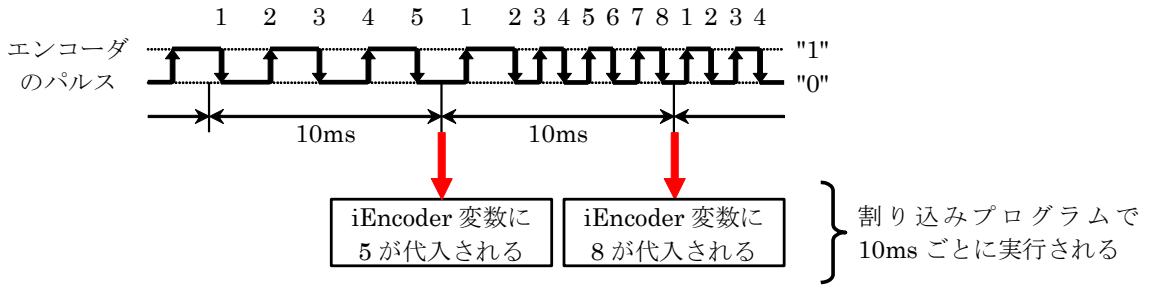


2 相出力のとき、ディスクが時計回り、反時計回りで回ったときの出力波形を下記に示します。1つのスリット(穴)で4カウントします。ロータリエンコーダ TypeS は90スリットですが、1回転 360 カウントします。



4.2.3. プログラムで速度を検出する

今回のプログラムでは、「**iEncoder**」という変数に 10ms 間のエンコーダパルス数が 10ms ごとに代入されます。1相の場合のパルスとカウント値を下記に示します。



例えば、秒速 2m/s 以上ならモータの PWM を 0%、それ未満なら PWM を 100%にする、というプログラムは下記のようになります。

```
if( 現在の速度 >= 2m/s ) {
    PWM を 0%にする
} else {
    PWM を 100%にする
}
```

「現在の速度」部分が、「iEncoder」になります。ロータリエンコーダ TypeS は秒速 1m/s で 55 パルス(前記表の(F)の計算)なので、

秒速 2m/s のパルス数 ＝ 秒速 1m/s のパルス数×2
＝ 55×2
＝ 110

となります。

プログラムは、下記のようになります。

```
    if( iEncoder >= 110 ) {  // ロータリエンコーダ Ver.2 なら 11×2＝22 になる
        motor2_f( 0, 0 );
        motor2_r( 0, 0 );
    } else {
        motor2_f( 70, 70 ); // ロータリエンコーダを使うときは、「motor2_f」を使う
        motor2_r( 70, 70 ); // ロータリエンコーダを使うときは、「motor2_r」を使う
    }
```

motor_f 関数、motor_r 関数はディップスイッチでスピードを調整した PWM をモータに出力します。今回は、ロータリエンコーダがスピードを調整しますので、ディップスイッチでスピードを調整する必要がありません。
よって、ディップスイッチでスピードを調整しない、「**motor2_f**」または「**motor2_r**」関数を使用します。

4.2.4. プログラムで距離を検出する

今回のプログラムは「**iEncoder**」変数の値を 10ms ごとに「**lEncoderTotal**」という変数に加算してきます。ロータリエンコーダ TypeS は1m進むとで 5459 パルス(前記表の(D)の計算)なので、lEncoderTotal 変数の値が 5459 になったら1m進んだということです。lEncoderTotal 変数の値をチェックすることにより、走行距離を知ることができます。

例えば、10m 進んだならモータの PWM を 0%、それ以下なら PWM を 100%にする、というプログラムは下記のようになります。

```
    if( 進んだ距離 >= 10m ) {
        PWM を 0%にする
        プログラム終了
    }
```

「進んだ距離」部分が、「lEncoderTotal」になります。
ロータリエンコーダ TypeS は 5459 パルス(前記表の(D)の計算)なので、
距離 10m のパルス数　＝ 1m のパルス数×10
　　＝ 5459×10
　　＝ 54590

プログラムは、下記のようになります。

```
    if( lEncoderTotal >= 54590L ) {  // ロータリエンコーダ Ver.2 なら 1092×10＝10920L になる
        motor_f( 0, 0 );
        motor_r( 0, 0 );
        while( 1 );  // 無限ループでプログラム終了
    }
```

※接尾語

54590**L** のように、数字の後ろに付いている"**L**"のことを接尾語と言い、定数の後ろに付加することで、その定数の型を指定することができます。

C 言語の型指定子を下記に示します。

型指定子		バイト長	表現範囲
キャラ char	文字型	1	JIS コードで表現される 1 文字、または-128 ～ +127 までの整数
イント int	整数型	2	-32 768 ～ +32 767 までの整数
ロング long	倍長整数型	4	-2 147 483 648 ～ +2 147 483 647 までの整数
フロート float	単精度実数型	4	約-10^{+38} ～ 約-10^{-38}，約 10^{-38} ～ 約 10^{+38} までの実数、および 0
ダブル double	倍精度実数型	8	約-10^{+308} ～ 約-10^{-308}，約 10^{-308} ～ 約 10^{+308} までの実数、および 0

int 型の上限である 32,767 を超える場合は、数字の末尾に「L」をつけておきます。詳しくは「接尾語　C 言語」などで検索して調べてみてください。

※本来は付けなくてもよい場合と、付けないといけない場合があります。実は下記は「L」を付けなくて問題ありません。

```
if( lEncoderTotal >= 54590 ) {    // 実は「L」がなくても問題ない
    プログラム
}
```

下記は、「L」を付けなければプログラムは正しく動きません。int 型同士の計算で「計算結果」が int 型の範囲を超えるときは必ず「L」を付けてください。

```
if( lEncoderTotal >= 5459L * 10L ) {    //  「L」がないと正しく動かない！
    プログラム
}
```

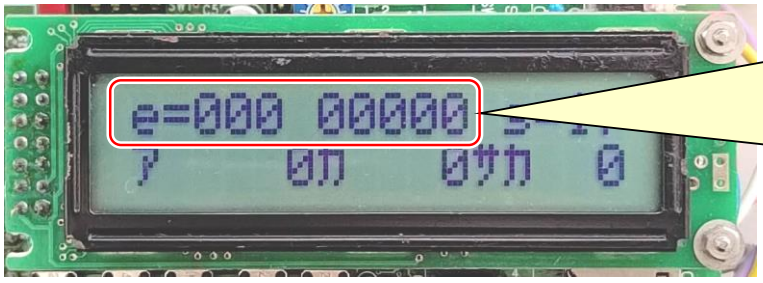
接尾語を付ける、付けないの判断を毎回するのは大変ですので、ロータリエンコーダの距離（int 型を超える値）のときは接尾語「L」を付けると覚えておくとよいでしょう。

4.2.5. ロータリエンコーダ値を実測して確認する

モータドライブ基板 TypeS のプッシュスイッチ(SW3)を押しながらマイコンカーの電源を入れ、ブザーが3回鳴ったらスイッチを離します。マイコンカーの状態確認画面が表示されます。

マイコンカーを手で 1m 進ませて TypeS なら 5000 以上、Ver.2 なら 950 以上あれば良いですが、それ以下の場合はフォトインタラプタとディスクの位置を調整してください。

※1m 進ませたときにほぼ 5459 カウントになるのが望ましいですが、タイヤがゴムなので直径コンマ数ミリの誤差があったり、ディスク部分の遊び（ガタ）で正確なカウントは難しいです。計算上の 9 割くらいの 5000 以上になれば、マイコンカー制御としては問題無いでしょう。



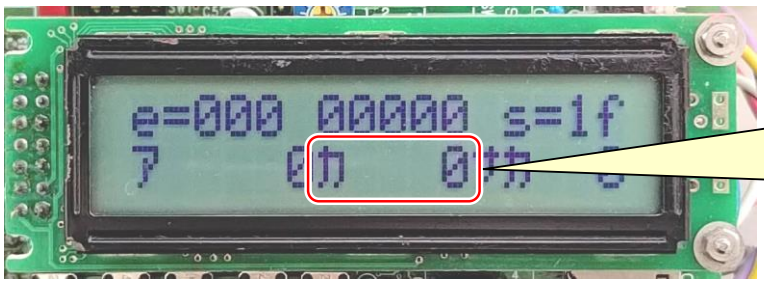
左側が iEncoder の値です。
10ms ごとの速度が表示されます。
TypeS は 1m/s で 55 です。

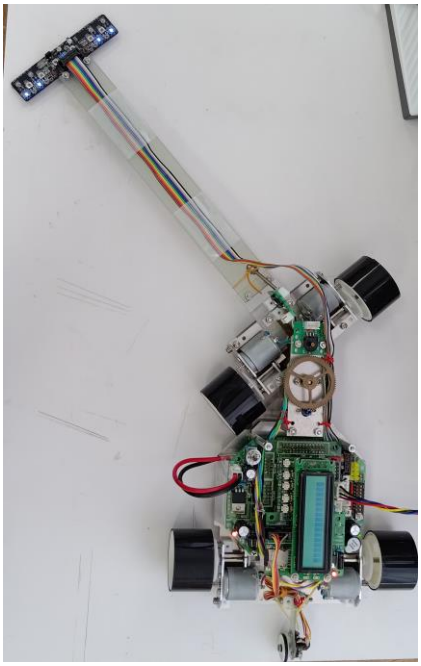
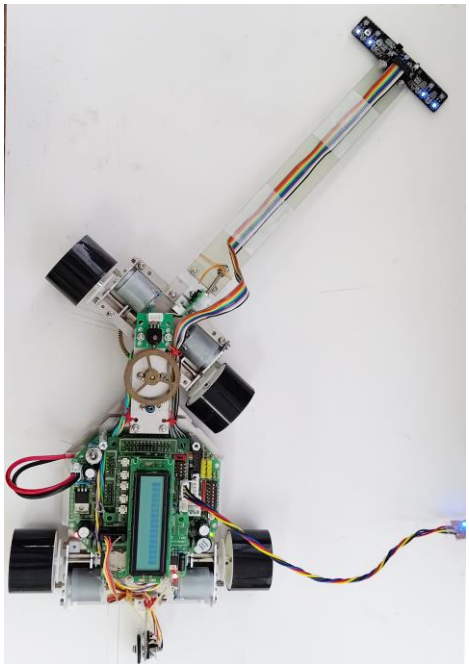
右側が lEncoderTotal の値です。TypeS は 1m 進むと 5459 なります。マイコンカーを手で 1m 進ませて 5000 以上あれば良いですが、それ以下の場合はフォトインタラプタとディスクの位置を調整してください。

4.3. 角度検出用ポテンショメータの確認

ステアリング角度をまっすぐにした状態で、モータドライブ基板 TypeS のプッシュスイッチ(SW3)を押しながらマイコンカーの電源を入れ、ブザーが3回鳴ったらスイッチを離します。マイコンカーの状態確認画面が表示されます。

下写真のように、左右にハンドルを止まるところまで動かして、そのときの AD 値をメモしてください。



左に止まるまでハンドルを切る	右に止まるまでハンドルを切る
	
このときの液晶の値＝＿＿＿＿＿＿＿＿ ※プラスの値になるように Vcc・GND 線を配線します。 参考:写真のマイコンカーは 130 でした。	このときの液晶の値＝＿＿＿＿＿＿＿＿ ※マイナスの値になるように Vcc・GND 線を配線します。 参考:写真のマイコンカーは-128 でした

※1度あたりの A/D 値について

Advanced マイコンカー製作マニュアルのとおりに製作すると、ハンドルは約±43度曲げることができます。
角度検出用ポテンショメータの値を±130 とすると、1度当たりの A/D 値は、

130 ÷ 43 ＝ 3.02 ≒ 3

となります。A/D 値が3増減すると、約1度増減したことになります。

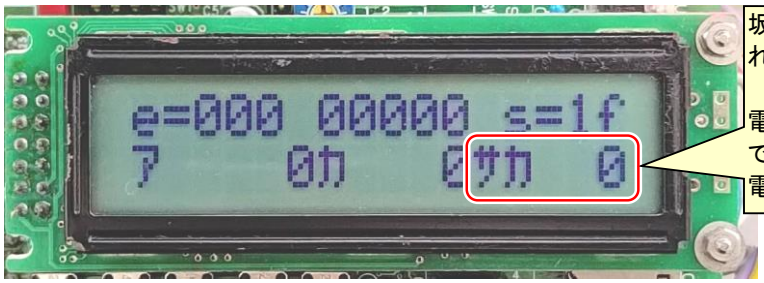
もし、**1 度あたりの A/D 値が3以外の場合**は、下記プログラムの**太字**部分を1度あたりの AD 値に変更します。

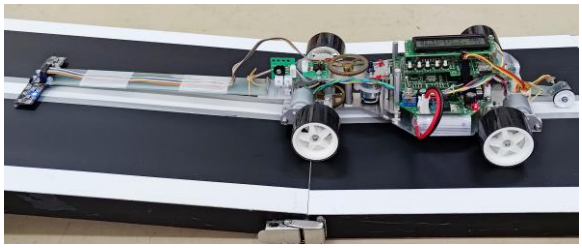
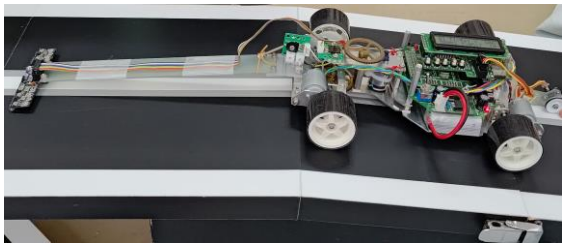
```
/************************************************************************/
/* 外輪のPWMから、内輪のPWMを割り出す　ハンドル角度は現在の値を使用　　*/
/* 引数　　外輪PWM                                                      */
/* 戻り値　内輪PWM                                                      */
/************************************************************************/
int diff( int pwm )
{
    int i, ret;

    i  = getServoAngle() / 3;           /* 1度あたりの増分で割る        */
    if( i <  0 ) i = -i;
    if( i > 45 ) i = 45;
    ret = revolution_difference[i] * pwm / 100;

    return ret;
}
```

4.4. 坂道用ポテンショメータ

コースの上など水平に置いた状態で、モータドライブ基板 TypeS のプッシュスイッチ(SW3)を押しながらマイコンカーの電源を入れ、ブザーが3回鳴ったら、スイッチを離します。マイコンカーの状態確認画面が表示されます。

下写真のように上り坂に入りセンサバーがいちばん上がったところ、上り坂終わりでセンサバーがいちばん下がったところの坂 AD 値をメモしてください。

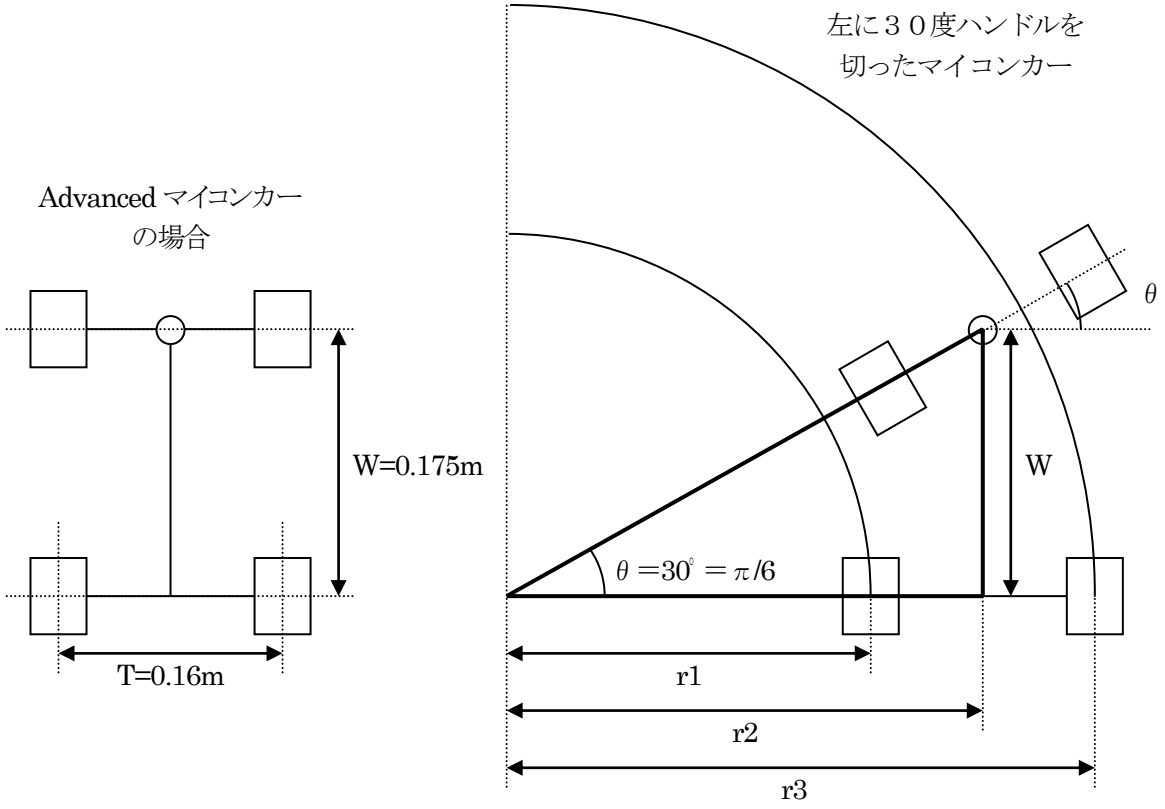


上り坂に入りセンサバーがいちばん上がったところ	上り坂終わりでセンサバーがいちばん下がったところ
	
このときの液晶の値＝＿＿＿＿＿＿＿＿ ※マイナスの値になるように Vcc・GND 線を配線します。 参考：写真のマイコンカーは-20 でした。	このときの液晶の値＝＿＿＿＿＿＿＿＿ ※プラスの値になるように Vcc・GND 線を配線します。 参考：写真のマイコンカーは 18 でした

4.5. 内輪差の計算

4.5.1. 内輪差について

ハンドルを切ったとき、内輪側と外輪側ではタイヤの回転数が違います。その計算方法を下記に示します。



W＝ホイールベース…前輪と後輪の間隔　0.175[m]です。
T＝トレッド…左右輪の中心線の距離　0.16[m]です。

図のように、底辺 r2、高さ W、角度 θ の三角形の関係は次のようです。

$\tan\theta = W / r2$

角度 θ 、W が分かっていますので、r2 が分かります。

$r2 = W / \tan\theta = 0.175 / \tan(\pi/6) = 0.303[m]$

内輪の半径 r1 は、

$r1 = r2 - T/2 = 0.303 - 0.08 = 0.223$

外輪の半径 r3 は、

$r3 = r2 + T/2 = 0.303 + 0.08 = 0.383$

よって、外輪を 100 とすると内輪の回転数は、

$r1 / r3 \times 100 = 0.223 / 0.383 \times 100 = 58$

となります。

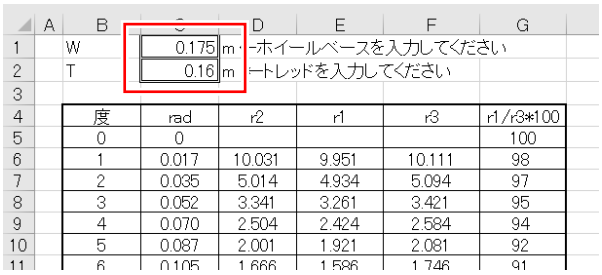
左に 30° ハンドルを切ったとき、右タイヤ 100 に対して、左タイヤ 58 の回転となる。

ハンドル角度が左－30度のとき、内輪と外輪のロスのない回転ができます。

```
motor_r( 58,  100 );
```

4.5.2. エクセルシートを使った内輪の計算

Advanced マイコンカーに合わせた内輪を計算し、「anaservo_ver4_38a_kai2023.c」に設定してみましょう。



	A	B	C	D	E	F	G
1		W	0.175	m	←ホイールベースを入力してください		
2		T	0.16	m	←トレッドを入力してください		
3							
4		度	rad	r2	r1	r3	r1/r3*100
5		0	0				100
6		1	0.017	10.031	9.951	10.111	98
7		2	0.035	5.014	4.934	5.094	97
8		3	0.052	3.341	3.261	3.421	95
9		4	0.070	2.504	2.424	2.584	94
10		5	0.087	2.001	1.921	2.081	92
11		6	0.105	1.666	1.586	1.746	91

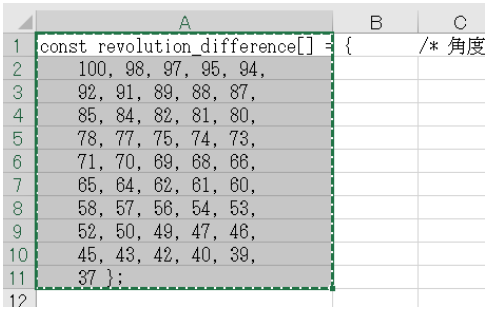
「角度計算.xls」を開きます。
□で囲んだ部分に、ホイールベースとトレッドを入力します。
Advanced マイコンカーのホイールベース(W)とトレッド(T)は下記の通りです。
W：0.175m
T：0.16m （後輪）

30	25	0.436	0.376	0.296	0.456	65
31	26	0.454	0.359	0.279	0.439	64
32	27	0.471	0.344	0.264	0.424	62
33	28	0.488	0.329	0.249	0.409	61
34	29	0.506	0.316	0.236	0.396	60
35	30	0.523	0.303	0.223	0.383	58
36	31	0.541	0.291	0.211	0.371	57
37	32	0.558	0.280	0.200	0.360	56

表　コピーして貼り付け用　＋

準備完了　アクセシビリティ: 利用不可

シート「コピーして貼り付け」を選択します。



	A	B	C
1	const revolution_difference[] = {		/* 角度
2	100, 98, 97, 95, 94,		
3	92, 91, 89, 88, 87,		
4	85, 84, 82, 81, 80,		
5	78, 77, 75, 74, 73,		
6	71, 70, 69, 68, 66,		
7	65, 64, 62, 61, 60,		
8	58, 57, 56, 54, 53,		
9	52, 50, 49, 47, 46,		
10	45, 43, 42, 40, 39,		
11	37 };		
12			

A 列 1 行～A 列 11 行までを選択し、コピーします。

「anaservo_ver4_38a_kai2023.c」ファイルの 117 行目～127 行目の**太字**部分を選択し、貼り付けます。

```
112 :  /* モータドライブ基板 TypeS 上の LED、ディップスイッチ制御 */
113 :  unsigned char   types_led;              /* LED 値設定                  */
114 :  unsigned char   types_dipsw;            /* ディップスイッチ値保存      */
115 :
116 :  /* 内輪差値計算用　各マイコンカーに合わせて再計算して下さい */
117 :  const revolution_difference[] = {       /* 角度から内輪、外輪回転差計算 */
118 :      100,  98,  97,  95,  94,
119 :       92,  91,  89,  88,  87,
120 :       85,  84,  82,  81,  80,
121 :       78,  77,  75,  74,  73,
122 :       71,  70,  69,  68,  66,
123 :       65,  64,  62,  61,  60,
124 :       58,  57,  56,  54,  53,
125 :       52,  50,  49,  47,  46,
126 :       45,  43,  42,  40,  39,
127 :       37 };
```

プログラムでは、内輪側を diff 関数を使って記載します。diff 関数の引数(カッコの中の数字)は、外輪と同じ PWM 値にします。

例えば、左にカーブしているとき、外輪(右)が 100%なら内輪(左)は「diff(100)」と記載します。記述例を下記に示します。

```
motor_r( diff(100), 100 );
```

もし、ハンドルを左に 30 度曲げていたら「diff(100)=58」と自動で計算し、次のようになります。

```
motor_r( 58, 100 );       // diff(100)部分が角度に応じて自動で変わる
```

外輪が 80%なら次のようにします。

```
motor_r( diff(80), 80 );
```

もし、ハンドルを左に 30 度曲げていたら「diff(80)=46」(58 の 80%)と自動で計算し、次のようになります。

```
motor_r( 46, 80 );        // diff(80)部分が角度と外輪の PWM に応じて自動で変わる
```

ただし、内輪・外輪の判断は自動では行いません。getServoAngle 関数で、ハンドルを左右のどちら側に曲げているか判断して、diff 関数を使ってください。

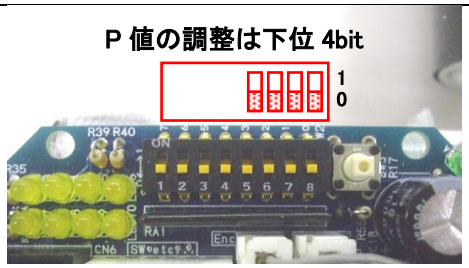
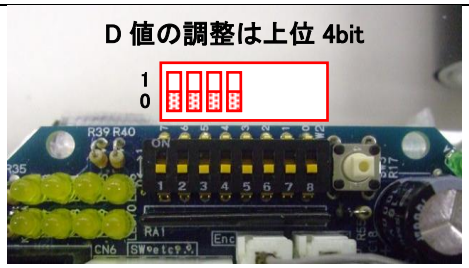
使用例を下記に示します。

```
    i = getServoAngle();              // 角度の A/D 変換値を取得 左で＋　右でー
    if( i >= 9 ) {                    // 角度が左 9(約 3 度)以上なら
        motor2_f( diff(80),  80 );    // 左：内輪　右：外輪となる
        motor2_r( diff(80),  80 );
    } else if( i <= -9 ) {            // 角度が右 9(約 3 度)以上なら
        motor2_f( 80, diff(80) );     // 左：外輪　右：内輪となる
        motor2_f( 80, diff(80) );
    } else {                          // それ以外 (角度が左右±9(約 3 度)未満)なら
        motor2_f( 80, 80 );           // ほぼ直進なので、左右とも 80%で回転
        motor2_f( 80, 80 );
    }

※motor2_f( 左前モータの PWM , 右前モータの PWM )、
  motor2_r( 左後モータの PWM , 右後モータの PWM )です。
  f= front(前)、r= rear(後ろ)の意味です。
```

なお、前輪、後輪でトレッドが違いますが、計算が複雑になるため、後輪の計算を前輪でも使用しています。

4.6. PD 値の調整

P 値、D 値は、モータドライブ基板 TypeS のディップスイッチ 8bit で調整することができます。

	P 値は、ディップスイッチの下位 4bit を使って調整します。 例えば、「0101」なら、P 値は5となります。
	D 値は、ディップスイッチの上位 4bit を使って調整します。 また、設定値×5倍の値になります。 例えば、「0110」なら、D 値は6×5＝30となります。

何度も走らせて、P値とD値が決まったら、プログラムに反映させます。
下記の**太字**部分を削除し、調整した値を入力します D 値は、ディップスイッチ値を 5 倍した値を入力します。

```
/*************************************************************************/
/* ステアリングモータ PD 制御                                            */
/* 引数    なし                                                          */
/* 戻り値 グローバル変数 iServoPwm に代入                                */
/*************************************************************************/
void servoControl( void )
{
    int i, iRet, iP, iD;
    int kp, kd;

    i = getAnalogSensor();              /* センサ値取得                  */
    kp = dipsw_get2() & 0x0f;           /* 調整できたら P,D 値は固定値に  */
    kd = (dipsw_get2() >> 4) * 5;       /* してください                  */

    /* ステアリングモータ用 PWM 値計算 */
    iP = kp * i;                        /* 比例                          */
    iD = kd * (iSensorBefore - i );     /* 微分(目安 P の 5～10 倍)       */
    iRet = iP - iD;
    iRet /= 64;

    /* PWM の上限、下限の設定 */
    if( iRet >  90 ) {
        iRet =  90;
    }
    if( iRet < -90 ) {
        iRet = -90;
    }
    iServoPwm = iRet;
    iSensorBefore = i;                  /* 次回はこの値が 1ms 前の値となる*/
}
```

P 値は、ディップスイッチ(下位 4bit)の値を入力します。

D 値は、ディップスイッチ(上位 4bit)の値を 5 倍した値を入力します。

4.7. 角度保持について

ステアリングモータを制御するPD制御は、アナログセンサのA/D値を使ってコース中心からのずれを検出して、コース中心に来るようにステアリングモータを制御します。レーンチェンジは中心線がありません。レーンチェンジはラジコンサーボのように、右に何度曲げる、というような処理をします。

4.7.1. 角度指定の PD 制御

コース中心にくるようにステアリングモータを PD 制御したとき、アナログセンサの「左－右」が値を使いました。角度制御は、角度検出用ポテンショメータの A/D 値にします。

	アナログセンサの値にするとき	**角度の値にするとき**
比例制御	制御量 P＝kp×p kp　＝定数 p　＝現在のアナログセンサ値－目標のアナログセンサ値 ＝getAnalogSensor()－0 ＝getAnalogSensor()	制御量 P＝kp×p kp　＝定数 p　**＝現在の角度　－目標の角度** **＝getServoAngle ()－iSetAngle** ※目標の角度を iSetAngle 変数に入れます
微分制御	制御量 D＝kd×d kd　＝定数 d　＝過去のアナログセンサ値－現在のアナログセンサ値 ＝iSensorBefore－getAnalogSensor()	制御量 D＝kd×d kd　＝定数 d　**＝過去の角度　－現在の角度** **＝iAngleBefore2－getServoAngle()**

4.7.2. 角度指定の PD 制御の計算プログラム

ステアリングモータの角度指定用 PD 制御は、servoControl2 関数で行います。intTRB 関数内に入れて、1ms ごとに実行するようにします。

サンプルプログラムは**比例定数 20、微分定数 100、計算後の調整値は 1/2 にしています**。この値は、各自調整してください。

```
/*****************************************************************************/
/* ステアリングモータ 角度のPD制御                                           */
/* 引数　なし                                                                */
/* 戻り値 グローバル変数 iServoPwm2 に代入                                   */
/*****************************************************************************/
void servoControl2( void )
{
    int i, j, iRet, iP, iD;
    int kp, kd;

    i = iSetAngle;
    j = getServoAngle();

    /* ステアリングモータ用PWM値計算 */
    iP = 20 * ( j - i );                /* 比例                              */
    iD = 100 * ( iAngleBefore2 - j );   /* 微分(目安はPの5～10倍)            */
    iRet = iP - iD;
    iRet /= 2;

    /* PWMの上限、下限の設定 */
    if( iRet >  90 ) {
        iRet =  90;
    }
    if( iRet < -90 ) {
        iRet = -90;
    }
    iServoPwm2 = iRet;

    iAngleBefore2 = j;                  /* 次回はこの値が1ms前の値となる*/
}
```

4.7.3. 角度指定するプログラム

main 関数などで実際に角度指定にするプログラムを説明します。

①iSetAngle 変数に、ステアリングモータで角度を指定したい A/D 値を代入します。
②プログラムは、「**servoPwmOut(iServoPwm2);**」を実行します。「servoPwmOut(**iServoPwm**);」とすると、センサ基板がコースの中心になるようなステアリング制御になります。2 が付くか付かないかの違いです。

下記に、main プログラムで角度指定を実験するプログラムを示します。
このプログラムできちんと角度指定できているか iSetAngle 変数の値を変えて実験してみましょう。Advanced マイコンカーは、1度当たりの A/D 値が 3 なので、45 に設定すると約 15 度左へ曲げます。0度は電源を入れたときの角度になります。

※電源を入れるとすぐに動くので、手でもって電源を入れてください。

#if の値を1に変えてプログラムを書き込み、電源を入れると iSetAngle 変数にセットした角度の A/D 値にハンドルを切ったまま、ほとんど動きません。このように、レーンチェンジで中心線がなくなると、角度指定でステアリングモータを制御します。
もし、保持が強すぎてぶるぶるしたり弱すぎる場合は、P 値＝20、D 値＝100を調整してください。

```
200 :  #if 0 ←1にすると、201～205 行が有効になる
201 :      // ステアリングモータの角度制御実験用
202 :      iSetAngle = +45;                    /* +で左 -で右にハンドルを切ります     */
203 :      while( 1 ) {
204 :          servoPwmOut( iServoPwm2 );
205 :      }
206 :  #endif
```

実験が終わったら、「#if 0」にして無効にしておきましょう。

4.8. 走行プログラムのパターン番号

マイコンカーの状態は pattern 変数で管理しています。通称、パターン処理と呼ぶことにします。pattern が 0 でスタート待ち、11 で通常トレースなど、それぞれの状態に応じてパターンを変えて処理内容を変えていきます。

現在のモード (pattern)	状態	pattern 変数が変わる条件
0	プッシュスイッチ押下待ち	・プッシュスイッチを押したら 1 へ
1	スタートバー開待ち	・スタートバーが開いたら 11 へ
11	通常トレース	・クロスラインを検出したら 21 へ ・右ハーフラインを検出したら 51 へ ・左ハーフラインを検出したら 61 へ
21	クロスライン通過処理	・50ms たったら 22 へ
22	クロスライン後のトレース、直角検出処理	・右クランクを見つけたら 31 へ ・左クランクを見つけたら 41 へ
31	右クランク処理	・曲げ終わりを検出すると 32 へ
32	少し時間がたつまで待つ	・100ms たったら 11 へ
41	左クランク処理	・曲げ終わりを検出すると 42 へ
42	少し時間がたつまで待つ	・100ms たったら 11 へ
51	右ハーフライン検出したら	・駆動モータの PWM や変数を設定したら 52 へ
52	右ハーフラインを通り過ぎるまで待つ	・50ms たったら 53 へ ・クロスラインを検出したら 21 へ
53	中心線がなくなるまで進む	・中心線が無くなったら 54 へ
54	新しい中心線が見つかるまで右に曲げる	・新しい中心線が見つかったら 55 へ
55	少し時間がたつまで待つ	・100ms たったら 11 へ
61	左ハーフライン検出したら	・駆動モータの PWM や変数を設定したら 62 へ
62	左ハーフラインを通り過ぎるまで待つ	・50ms たったら 63 へ ・クロスラインを検出したら 21 へ
63	中心線がなくなるまで進む	・中心線が無くなったら 64 へ
64	新しい中心線が見つかるまで右に曲げる	・新しい中心線が見つかったら 65 へ
65	少し時間がたつまで待つ	・100ms たったら 11 へ
その他	―	・0 へ

4.9. パターン 0～1:スタート待ち

パターン 0 は、プッシュスイッチ押下待ちです。プッシュスイッチを押すまでの間、LED8 個中 2 個を点滅させプッシュスイッチが押されるまで待ちます。また、スタートバー検出センサが反応するとさらに 2 個(合計 4 個)点滅させ、スタートバー閉を検出していることを、選手に分かりやすく知らせます。

プッシュスイッチを押すと、ブザーを1回鳴らします。microSD 処理などが終わってパターン1に移る前に、ブザーを2回鳴らして、準備完了を知らせます。

```
269 :      case 0:
270 :          /* プッシュスイッチ押下待ち */
271 :          servoPwmOut( 0 );
272 :          if( pushsw_get() == 1 ) {          プッシュスイッチを押したら
273 :              setBeepPatternS( 0x8000 );     ブザーを１回鳴らす
中略
293 :              if( msdError == 0 ) {
294 :                  /* ファイルのタイムスタンプセット */
295 :                  setDateStamp( getCompileYear( C_DATE ),
296 :                      getCompileMonth( C_DATE ),  getCompileDay( C_DATE ) );
297 :                  setTimeStamp( getCompileHour( C_TIME ),
298 :                      getCompilerMinute( C_TIME ), getCompilerSecond( C_TIME ) );
299 :
300 :                  /* 書き込みファイル名作成 */
301 :                  // 書き込みしたい時間[ms] : x = 10[ms] : 64バイト
302 :                  // 60000msなら、x = 60000 * 64 / 10 = 384000
303 :                  // 結果は512の倍数になるように繰り上げする。
304 :                  ret = writeFile( fileName, 384000 );
305 :                  if( ret != 0x00 ) msdError = 11;
306 :
307 :                  // microSD書き込み
308 :                  msdPrintf( "[Your Car Name] Log Data¥n" );
309 :                  while( checkMsdPrintf() );  // msdPrintf処理完了待ち
310 :                  msdPrintf( "Compile Date:" );
311 :                  while( checkMsdPrintf() );  // msdPrintf処理完了待ち
312 :                  msdPrintf( C_DATE );
313 :                  while( checkMsdPrintf() );  // msdPrintf処理完了待ち
314 :                  msdPrintf( " Time:" );
315 :                  while( checkMsdPrintf() );  // msdPrintf処理完了待ち
316 :                  msdPrintf( C_TIME );
317 :                  while( checkMsdPrintf() );  // msdPrintf処理完了待ち
318 :                  msdPrintf( "¥n¥nLineNo,Pattern,Sensor,Center,"
319 :                         "Analog,Angle,Encoder,左前,右前,左後,右後,坂AD,坂flag+坂pattern¥n" );
320 :                  while( checkMsdPrintf() );  // msdPrintf処理完了待ち
321 :              }
322 :              writeDataFlashParameter();     パラメータを保存
323 :              iAngle0  = ad2;                今の状態を０度とする
324 :              saka0_ad = ad4;                今の状態を平地とする
325 :              cnt1 = 0;
326 :              setBeepPatternS( 0x8800 );     パターン１に移る前にブザーを２回鳴らす
327 :              pattern = 1;
328 :              break;
329 :          }
330 :          i =  (cnt1/200) % 2 + 1;
331 :          if( startbar_get() == 1 ) {
332 :              i |= ((cnt1/100 ) % 2 + 1) << 2;
333 :          }
334 :          i |= (p3_2 << 5) | (p3_0 << 4);
335 :          led_out( i );                   /* LED点滅処理                  */
336 :          break;
```

パターン 1 は、スタートバーが開かれるのを待っている状態です。
340 行でステアリングモータ制御を行っています。iServoPwm 変数が PD 制御でステアリングモータに加える PWM 値です。割り込みプログラム内で 1ms ごとに自動更新されます。スタート時、センサがばたばたしないように、PWM 値を 1/5 にしています。弱かったり強すぎたりする場合は、適宜調整してください。

```
338 :      case 1:
339 :          /* スタートバー開待ち */
340 :          servoPwmOut( iServoPwm / 5 );
341 :          if( startbar_get() == 0 ) {          スタートバーの反応が無くなったら（開いたら）
342 :              led_out( 0x0 );
343 :              if( msdError == 0 ) msdFlag = 1;     /* データ記録開始        */
344 :              cnt1 = 0;
345 :              check_sen_cnt = 0;
346 :              check_enc_cnt = 0;
347 :              lEncoderTotal = 0;                ここを０ｍとしてスタート
348 :              digital_sensor_hosei_flag = 1;   デジタルセンサでアナログセンサの値を補正 1:する 0:しない
349 :              pattern = 11;
350 :              break;
351 :          }
352 :          led_out( 1 << (cnt1/50) % 4 );
353 :          break;
3
```

4.10. パターン 11:通常トレース

パターン11は通常走行です。
358 行で、変数 i に角度の A/D 変換値を代入します。ステアリングが左だとプラス、右だとマイナスです。
360 行で、saka_flag 変数(坂道フラグ)が1なら上り坂と判断して、秒速 1m/s 以上なら PWM を 0%、未満なら 95%にして、秒速 1m/s で走行するようにします。

```
355 :      case 11:
356 :          /* 通常トレース */
357 :          servoPwmOut( iServoPwm );
358 :          i = getServoAngle();              角度のA/D変換値を取得 左で＋　右でー
359 :
360 :          if( saka_flag == 1 ) {            坂道フラグがONなら減速
361 :              if( iEncoder >= 55 ) {        ロータリエンコーダTypeSは1m/s=55
                                                  数字が適宜調整してください
362 :                  motor2_f(   0,    0 );
363 :                  motor2_r(   0,    0 );
364 :              } else {
365 :                  motor2_f(  95,   95 );
366 :                  motor2_r(  95,   95 );
367 :              }
368 :              break;  // case 11はここで終了
369 :          }
```

次に、角度やスピードに応じて、4輪の PWM 値を調整します。

```
371 :          if( i >= 50 ) {                  角度が左50(約15度)以上で
372 :              if( iEncoder >= 110 ) {      速度が秒速2.0m/s以上なら
373 :                  motor2_f(   0,    0 );   0%にする
374 :                  motor2_r(   0,    0 );
375 :              } else {                     それ以外（秒速2.0m/s未満)なら
376 :                  motor2_f(  50,   80 );   モータを回転させて走行させる
377 :                  motor2_r(   0,   80 );
378 :              }
379 :          } else if( i <= -50 ) {          角度が右50(約15度)以上で
380 :              if( iEncoder >= 110 ) {      速度が秒速2.0m/s以上なら
381 :                  motor2_f(   0,    0 );   0%にする
382 :                  motor2_r(   0,    0 );
383 :              } else {                     それ以外（秒速2.0m/s未満)なら
384 :                  motor2_f(  80,   50 );   モータを回転させて走行させる
385 :                  motor2_r(  80,    0 );
386 :              }
387 :          } else if( i >= data_buff[DF_P11_ANG1] ) {  角度が左15(約5度)以上で
388 :              if( iEncoder >= 137 ) {                 速度が秒速2.5m/s以上なら
389 :                  motor2_f(   0,    0 );              0%にする
390 :                  motor2_r(   0,    0 );
391 :              } else if( iEncoder >= 126 ) {          速度が2.5m/s未満2.3m/s以上なら
392 :                  motor2_f( 50, 80 );                 モータを回転させて走行させる
393 :                  motor2_r(  0, 80 );
394 :              } else {                                それ以外（秒速2.3m/s未満)なら
395 :                  motor2_f( diff(80), 80 );           モータを回転させて走行させる
396 :                  motor2_r( diff(80), 80 );
397 :              }
398 :          } else if( i <= -data_buff[DF_P11_ANG1] ) { 角度が右15(約5度)以上なら
399 :              if( iEncoder >= 137 ) {                 速度が秒速2.5m/s以上なら
400 :                  motor2_f(  0,    0 );               0%にする
401 :                  motor2_r(  0,    0 );
402 :              } else if( iEncoder >= 126 ) {          速度が2.5m/s未満2.3m/s以上なら
403 :                  motor2_f( 80, 50 );                 モータを回転させて走行させる
404 :                  motor2_r( 80,  0 );
405 :              } else {                                それ以外（秒速2.3m/s未満)なら
406 :                  motor2_f( 80, diff(80) );           モータを回転させて走行させる
407 :                  motor2_r( 80, diff(80) );
408 :              }
409 :          } else {                                それ以外(角度が左右約5度未満なら、ほぼ直線と判断し
410 :              if( iEncoder >= data_buff[DF_P11_ENC]*5 ) {  速度が2.5m/s以上なら
411 :                  motor2_f( 0, 0 );                          0%にする
412 :                  motor2_r( 0, 0 );
413 :              } else {                                       それ以外（秒速2.5m/s未満)なら
414 :                  motor2_f( 100, 100 );                      モータを100%で走行させる
415 :                  motor2_r( 100, 100 );
416 :              }
417 :          }
```

初期値 15 です

初期値 15 です

初期値 27 です

最後に、クロスライン、右ハーフライン、左ハーフラインのチェックです。

クロスラインを見つけたらパターン21へ移ります。右ハーフラインを見つけたらパターン51へ移ります。左ハーフラインを見つけたらパターン61へ移ります。

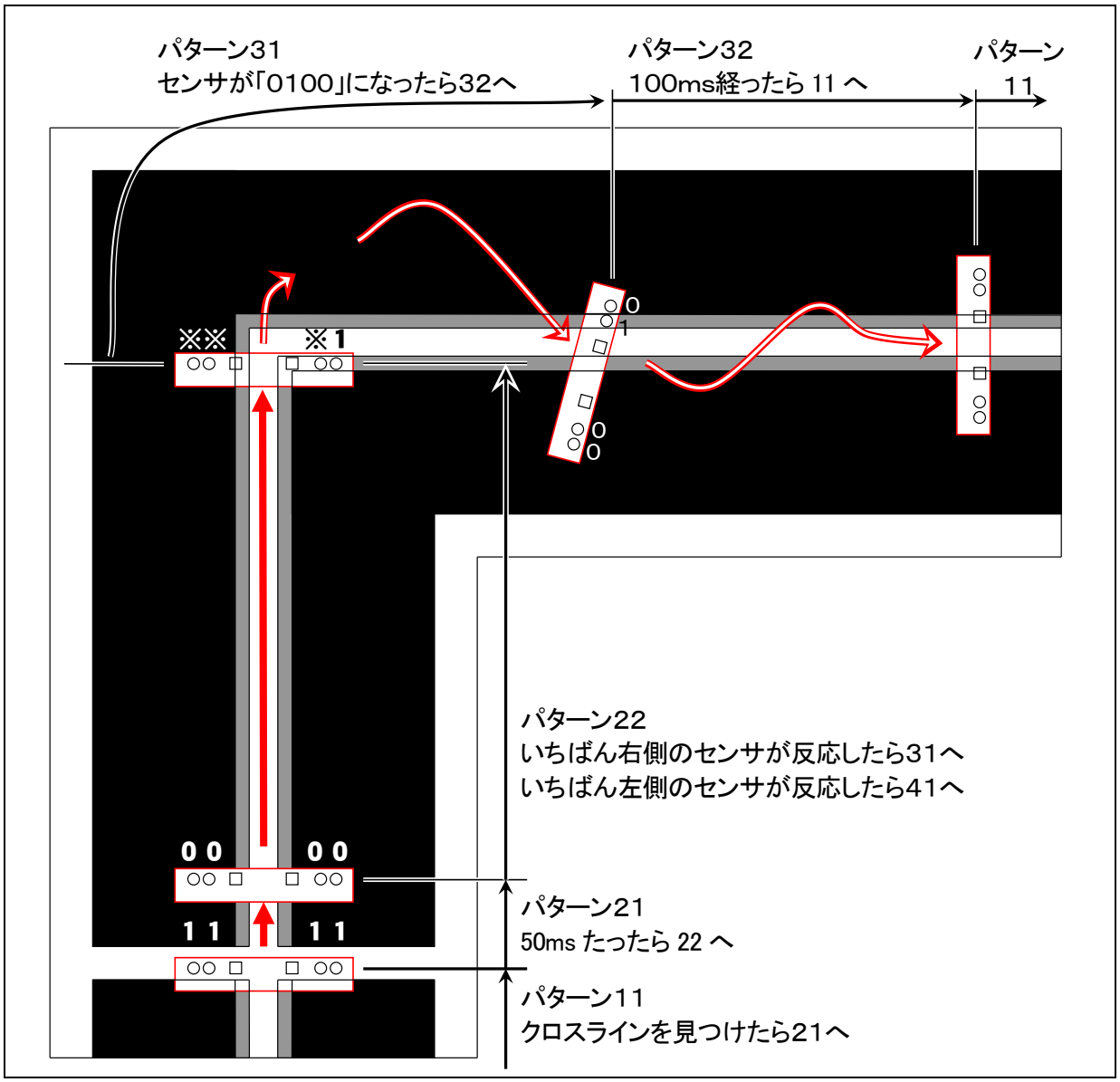
digital_sensor_hosei_flag は、デジタルセンサでアナログセンサの値を補正するかしないかのフラグです。クロスラインやハーフラインでデジタルセンサが反応しているので、デジタルセンサ値で補正すると、アナログセンサの値が強制的に 700 や 800 になるので digital_sensor_hosei_flag の値を 0 にして、デジタルセンサの補正処理をしないようにしています。

```
418 :          if( check_crossline() == 1 ) {       /* クロスラインチェック          */
419 :              cnt1 = 0;
420 :              digital_sensor_hosei_flag = 0;
421 :              pattern = 21;
422 :          }
423 :          if( check_rightline() == 1 ) {       /* 右ハーフラインチェック        */
424 :              cnt1 = 0;
425 :              digital_sensor_hosei_flag = 0;
426 :              pattern = 51;
427 :          }
428 :          if( check_leftline() == 1 ) {       /* 右ハーフラインチェック        */
429 :              cnt1 = 0;
430 :              digital_sensor_hosei_flag = 0;
431 :              pattern = 61;
432 :          }
433 :          break;
```

4.11. パターン 21～42:クロスライン処理

クロスラインを見つけてから左右のクランク(直角)を見つけるまでの処理が、パターン 21、22 です。
右クランクを見つけるとパターン 31、32 を実行します。32 の次は通常トレースの 11 へ戻ります。
左クランクを見つけるとパターン 41、42 を実行します。42 の次は通常トレースの 11 へ戻ります。

クロスラインから右クランクを進んで、パターン11へ戻るまでの概要を下図に示します。



4.11.1. パターン 21:クロスライン通過処理

```
435 :      case 21:
436 :          /* クロスライン通過処理 */
437 :          servoPwmOut( iServoPwm );
438 :          led_out( 0x3 );
439 :          motor_f( 0, 0 );
440 :          motor_r( 0, 0 );
441 :          if( cnt1 >= 50 ) {       cnt1変数は1msごとに＋１する変数 50以上ということは
442 :              cnt1 = 0;                                    50ms以上なら、という意味になる
443 :              pattern = 22;        50ms後、パターン22へ
444 :          }
445 :          break;
```

モータ PWM を 0%にしてブレーキをかけます。50ms の間にクロスラインを通過させ、50ms 後にはパターン 22 へ移行します。クロスラインを通過しきる前にパターン 22 に移ってしまうと、クロスラインを直角と見間違って脱輪してしまいます。cnt1 変数は 1ms ごとに＋1する変数です。

4.11.2. パターン 22:クロスライン後のトレース、直角検出処理

```
447 :      case 22:
448 :          /* クロスライン後のトレース、直角検出処理 */
449 :          servoPwmOut( iServoPwm );
450 :          if( iEncoder >= 115 ) {          速度が約2.1m/s以上なら
451 :              motor2_f( -80, -80 );        -80%で逆転ブレーキ
452 :              motor2_r( -80, -80 );
453 :          } else if( iEncoder >= 110 ) {   速度が約2.0m/s以上なら
454 :              motor2_f( 0, 0 );            0%でモータを回さない
455 :              motor2_r( 0, 0 );
456 :          } else {                         それ以外(速度が約2.0m/s未満)なら
457 :              motor2_f( 70, 70 );          70%でモータを回転させ秒速2.0mになるようにする
458 :              motor2_r( 70, 70 );
459 :          }
460 :
461 :          if( (sensor_inp() & 0x01) == 0x01 ) { /* 右クランク？              */
462 :              led_out( 0x1 );
463 :              cnt1 = 0;
464 :              pattern = 31;
465 :              break;
466 :          }
467 :          if( (sensor_inp() & 0x08) == 0x08 ) {  /* 左クランク？              */
468 :              led_out( 0x2 );
469 :              cnt1 = 0;
470 :              pattern = 41;
471 :              break;
472 :          }
473 :          break;
```

クロスライン通過後の処理を行います。450～459 行でロータリエンコーダによる速度制御を行っています。秒速 2m/s で走行するように制御しています。

461 行目で、いちばん右のデジタルセンサのみをチェック、反応すれば右クランクと判断しパターン 31 へ移ります。例えば、sensor_inp()関数で左2個、右2個のデジタルセンサの値を読み込んだときに、「0111」とします。このとき、次のように 0x01 で AND 演算を行います。

sensor_inp()関数で取得したセンサの値	0111
AND演算する値	&) 0001
結果	0001

結果が 0x01(2 進数で 0001)なら、右クランクと判断します。同様に 467 行目で、いちばん左のセンサのみをチェック、反応すれば左クランクと判断しパターン 41 へ移ります。

4.11.3. パターン 31・32:右クランク処理

```
475 :       case 31:
476 :           /* 右クランク処理 */
477 :           servoPwmOut( 90 );                  ステアリングモータを90%で右に回転させる(PD制御はしません)
478 :           motor_f( 60,  0 );
479 :           motor_r( 49, 22 );
480 :           if( sensor_inp() == 0x04 ) {        デジタルセンサが「0100」になったら中心線に戻ったと判断
481 :               cnt1 = 0;
482 :               iSensorPattern = 0;
483 :               digital_sensor_hosei_flag = 1;  デジタルセンサ補正を有効にする
484 :               pattern = 32;
485 :           }
486 :           break;
487 :
488 :       case 32:
489 :           /* 少し時間が経つまで待つ */
490 :           servoPwmOut( iServoPwm );
491 :           motor2_r( 80, 80 );
492 :           motor2_f( 80, 80 );
493 :           if( cnt1 >= 100 ) {                 100ms経ったらパターン11へ戻る
494 :               led_out( 0x0 );
495 :               pattern = 11;
496 :           }
497 :           break;
```

パターン 31 は、右にハンドルを曲げて、曲げ終わりかどうかチェックしている状態です。ステアリングモータは、センサ状態に関係なく右に PWM90%で回転させています。デジタルセンサが「0100」になったら中心線に戻ったと判断して、パターン 32 へ移ります。

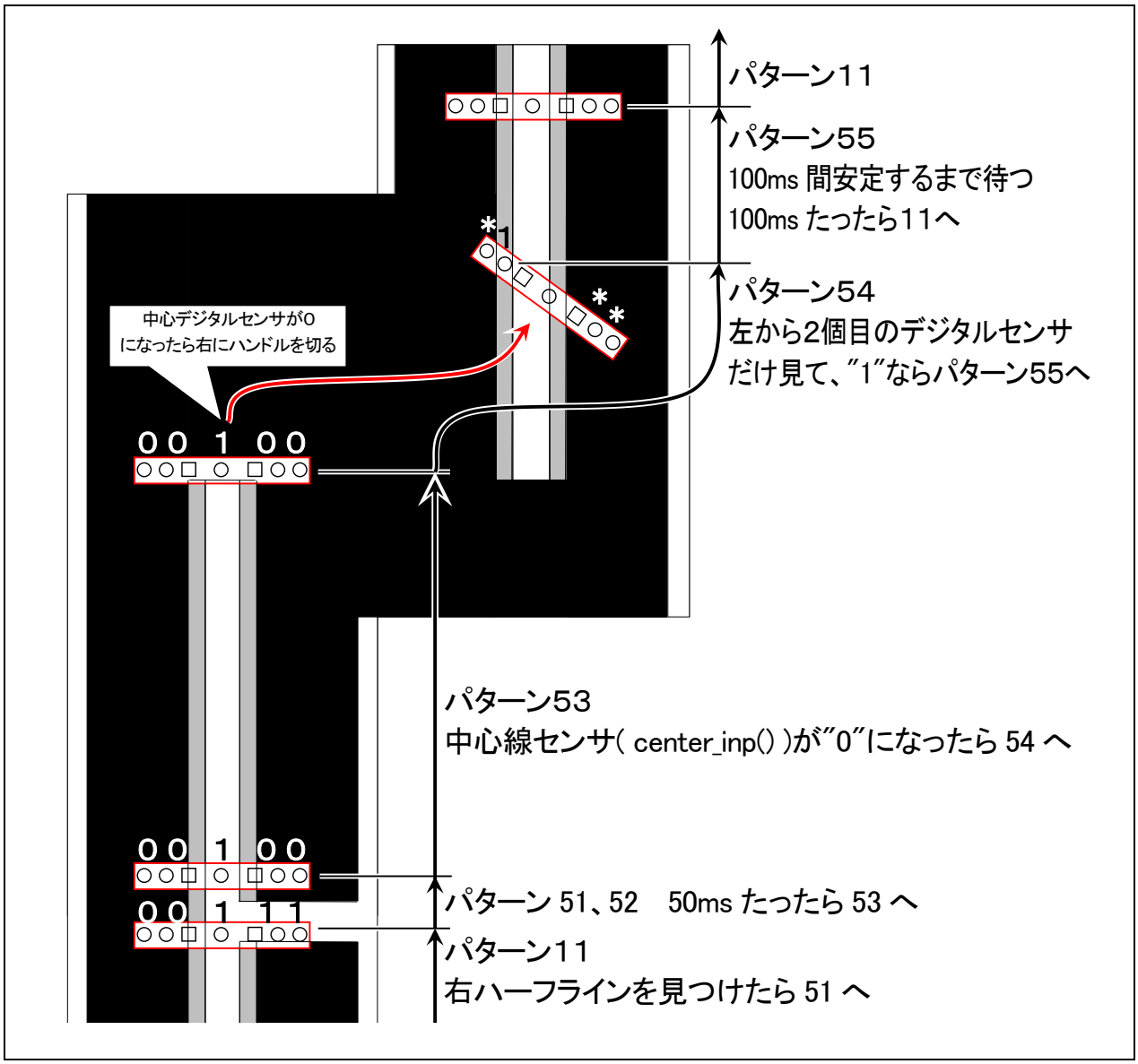
パターン 32 は、中心線に戻ったばかりでセンサがばたばたするので、100ms はばたばたが収まるまで PWM80%で進めます。100ms 経ったらばたばたが収まったものと判断し、パターン 11 の通常走行へ戻します。

4.11.4. パターン 41・42:左クランク処理

パターン 41・42 は、パターン 31・32 の左右逆になる動きになります。

4.12. パターン 51～55：右レーンチェンジプログラム

右ハーフラインを見つけてからレーンチェンジ後の通常トレースに戻るまで処理が、パターン 51～55 です。
右ハーフラインからパターン11へ戻るまでの概要を下図に示します。



4.12.1. パターン 51:右ハーフラインの検出

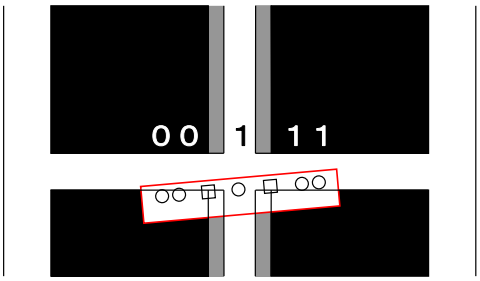
```
523 :       case 51:
524 :           /* 右レーンチェンジ　右ハーフライン検出したら */
525 :           servoPwmOut( iServoPwm );
526 :           motor2_f( 50, 50 );
527 :           motor2_r( 50, 50 );
528 :           led_out( 0xc0 );
529 :           pattern = 52;
530 :           cnt1 = 0;
531 :           break;
```

右ハーフラインを検出したら PWM を 50%、cnt1 変数を 0 にしてパターン 52 へ移行します。パターン 52 で cnt1 を使いますが、cnt1 を 0 にしたここからの時間になります。

4.12.2. パターン 52:右ハーフラインを通り過ぎるまで待つ

```
533 :       case 52:
534 :           /* 右レーンチェンジ　右ハーフラインを通り過ぎるまで待つ */
535 :           servoPwmOut( iServoPwm );
536 :           if( cnt1 > 50 ) {
537 :               pattern = 53;
538 :           }
539 :           if( check_crossline() == 1 ) {       /* 右ハーフラインチェック       */
540 :               cnt1 = 0;
541 :               pattern = 21;
542 :           }
543 :           break;
```

50ms の間に右ハーフラインを通過させ、50ms 後にパターン 53 へ移行します。
このとき、クロスラインがどうかチェックしてクロスラインなら、本当はクロスラインだったのに誤って右ハーフラインを検出してしまったので、パターン 21 のクロスライン処理へ移行します。なぜ誤検出するかというと、下図のようにセンサが斜めにクロスラインに入ってしまったとき、右ハーフライン検出と同じセンサ状態になるためです。



4.12.3. パターン 53:右ハーフラインの検出

```
545 :       case 53:
546 :           /* 右レーンチェンジ　中心線がなくなるまで進む */
547 :           servoPwmOut( iServoPwm );
548 :           if( iEncoder >= 110 ) {          速度が約2.0m/s以上なら
549 :               motor2_f( 0, 0 );            0%でブレーキ
550 :               motor2_r( 0, 0 );
551 :           } else {                         それ以外(速度が約2.0m/s未満)なら
552 :               motor2_f( 70, 70 );          70%でモータを回転させ秒速2.0mになるようにする
553 :               motor2_r( 70, 70 );
554 :           }
555 :           if( center_inp() == 0 ) {        中心線が無くなったら
556 :               pattern = 54;                パターン54へ移行する
557 :               break;
558 :           }
559 :           break;
```

中心線がなくなるまで中心線にそって走ります。中心線が無くなったら右に急激に曲げなければいけないので、安定して曲げられるようにスピードを落として走行します。

555 行で中心センサをチェックして中心の白線がなくなったら右レーンチェンジ開始と判断してパターン54へ移行します。

4.12.4. パターン 54:新しい中心線が見つかるまで右に曲げる

```
561 :       case 54:
562 :           /* 右レーンチェンジ　新しい中心線が見つかるまで右に曲げる */
563 :           iSetAngle = -45;                 右に角度AD値で45（角度だと約１５度)に指定
564 :           servoPwmOut( iServoPwm2 );       iServoPwm2は角度指定
565 :           motor2_f( 50, 0 );
566 :           motor2_r( 50, 20 );
567 :           if( (sensor_inp()&0x04) == 0x04 ) {  センサ値を0x04でANDして、結果が0x04なら
568 :               pattern = 55;                    パターン55へ移行
569 :               led_out( 0x00 );
570 :               iSensorPattern = 0;
571 :               digital_sensor_hosei_flag = 1;   デジタルセンサ補正処理をONにする
572 :               cnt1 = 0;
573 :               break;
574 :           }
575 :           break;
```

中心線がなくなるので、ステアリングモータをアナログセンサ値で PD 制御することができません。ここでは角度用ポテンショメータを使って、右に 45(角度だと約15度)曲げるようにしています。

デジタルセンサの左から2個目だけチェックして、"1"なら中心線を見つけたと判断して、パターン 55 へ移行します。

4.12.5. パターン 55:新しい中心線をトレース

```
577 :       case 55:
578 :           /* 右レーンチェンジ　新しい中心線をトレース */
579 :           servoPwmOut( iServoPwm );  角度指定は止めて中心線トレースで進んでいく
580 :           motor2_f(  0, 50 );
581 :           motor2_r( 20, 50 );
582 :           if( cnt1 > 100 ) {         100msたったらトレースが安定したと判断して
583 :               pattern = 11;          パターン11に戻る
584 :               cnt1 = 0;
585 :           }
586 :           break;
```

新しい中心線を見つけましたが右にハンドルを切っている状態なので、左前モータ0%、右前モータ50%程度にして、中心線トレースが安定するまで 100ms 間トレースします。100ms たったら安定したと判断して、パターン 11 に戻り通常トレースを再開します。

4.13. パターン 61～65：左レーンチェンジプログラム

パターン 61～65 は、パターン 51～55 の左右逆になる動きになります。

4.14. microSD に走行データを記録して走行データを解析する

Advnaced マイコンカーは、走行中の走行データ(ログ(log)といいます)を microSD に保存して、走行後にパソコンのエクセルなどで解析することができます。
詳しい説明は、「液晶・microSD 基板(Ver.2) kit12_38a プログラム解説マニュアル データ解析(microSD)編(R8C/38A 版)」を参照してください。

4.14.1. ログ保存ファイルの容量

microSD に保存する容量は下記の行で決めています。

```
304 :                   ret = writeFile( fileName, 384000 );
```

ログは microSD への書き込み時間の関係で 10ms ごとに 64 バイト書き込みができます。今回は、60 秒保存できるようにします。 microSD に確保する容量は、次のようになります。

容量＝記録したい時間[ms]÷記録する間隔[ms]×1 回に記録するバイト数

よって、容量は次のとおりです。

＝60000÷10×64
＝384000

値は、512 の倍数にしなければいけません。512 の倍数かどうか確かめます。

384000÷512＝750 余り 0

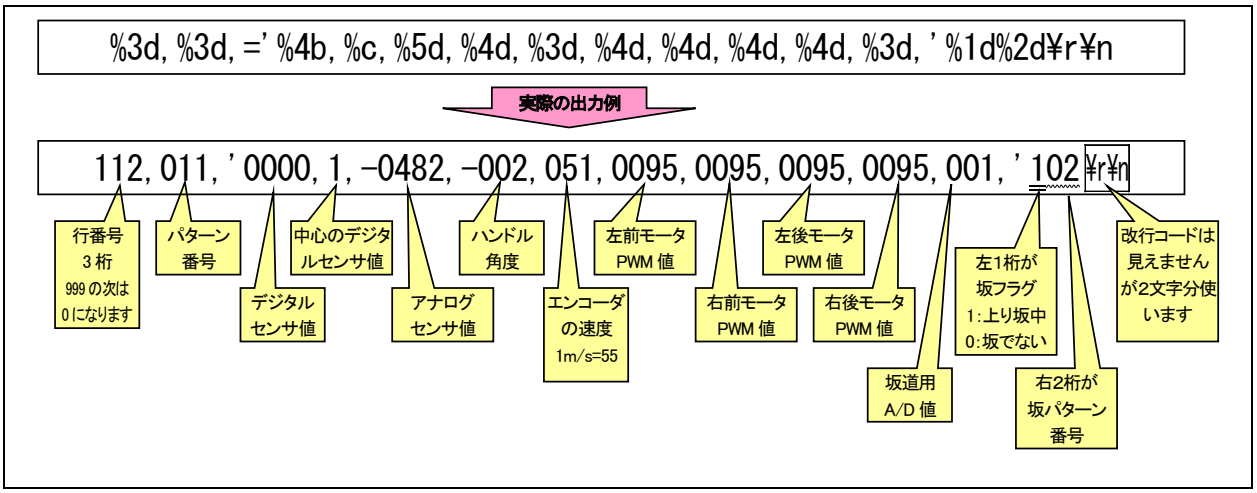
よって、384000 バイト、記録容量を確保することとします。

4.14.2. ログの保存方法

ログの保存は、1ms ごとの割り込みの中で 10 回に 1 回(10ms ごと)に microSD に保存します。
実際に保存している部分は下記のプログラム部分です。
「%○」を変換指定子といいます。色分けしているように変数の値が変換指定子の書式に従って、数値などに変換されます。

```
906 :       /* microSD記録処理 */
907 :       if( msdFlag == 1 ) {
908 :           msdPrintf( "%3d,%3d,'%4b,%c,%5d,%4d,%3d,%4d,%4d,%4d,%4d,%3d,'%1d%2d¥r¥n",
909 :               line_no,                // 行番号
910 :               pattern,                // パターン番号
911 :               sensor_inp(),           // デジタルセンサ (4bit)
912 :               center_inp() + '0',     // デジタルセンサ (中心)
913 :               getAnalogSensor(),      // アナログセンサ
914 :               getServoAngle(),        // ボリューム (ステアリング角度)
915 :               iEncoder,               // ロータリエンコーダ
916 :               motorLF,                // 左前モータ
917 :               motorRF,                // 右前モータ
918 :               motorLR,                // 左後モータ
919 :               motorRR,                // 右後モータ
920 :               getSakaAngle(),         // 坂A/D値
921 :               saka_flag,              // 坂フラグ
922 :               saka_pattern            // 坂道処理用パターン番号
923 :           );
924 :
925 :           if( ++line_no >= 1000 ) line_no = 0;
926 :       }
927 :       break;
```

microSD へ記録する書式と出力例を下記に示します。



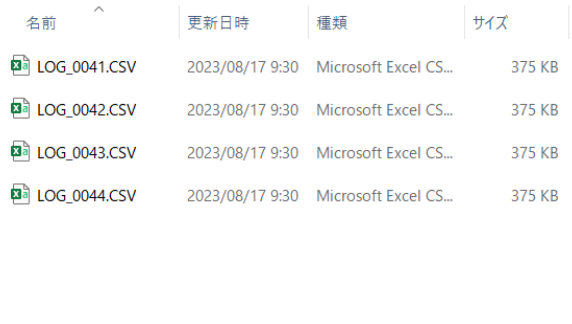
※「%b」は msdPrintf 関数で使用できる変換指定子で2進数に変換できます。通常の C 言語にはありません。

ポイントは、改行コードの2文字を含めて 64 文字以内でなければいけません。今回は下記のように 62 文字分記録しています。 ※「¥r」で1文字分、「¥n」で1文字分になります。
1行 64 文字を超えるとログをうまく保存できないことがありますので、64 文字以内にしてください。また、microSD によっては 64 文字以下でもうまく保存できないことがあります。このときは、1行の文字数を減らしてください。

```
                                1111111111222222222233333333334444444444555555555666
    文字数    12345678901234567890123456789012345678901234567890123456789012456789012
ログ出力例：112, 011, ' 0000, 1, -0482, -002, 051, 0095, 0095, 0095, 0095, 001, ' 102¥r¥n
```

4.14.3. ログをエクセル（表計算ソフト）で確認する

※csv ファイルが読み込める表計算ソフトであれば、フリーソフトで構いません。



ログを保存した microSD をパソコンで読み込みます。
ファイル名は「LOG_xxxx.csv」というファイル名で「xxxx」には連番で数字が入ります。
ログファイルは追加されていきますが、ファイルが増えていくとスタートするときに microSD の空部分を探すのに時間がかかるので、適宜パソコンに移動させてください。
なお、ファイルの日付は、ビルドした日付になります。
CSV ファイルをダブルクリックします。

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	[Your Car Name] Log Data												
2	Compile Date:Aug 17 2023 Time:09:30:20												
3													
4	LineNo	Pattern	Sensor	Center	Analog	Angle	Encoder	左前	右前	左後	右後	坂AD	坂flag+坂pattern
5	0	11	0000	1	194	-1	0	100	100	100	100	-1	'000
6	1	11	0000	1	-182	-2	0	100	100	100	100	1	'000
7	2	11	0000	1	506	0	5	100	100	100	100	-1	'000
8	3	11	0000	1	-533	-4	11	100	100	100	100	1	'000

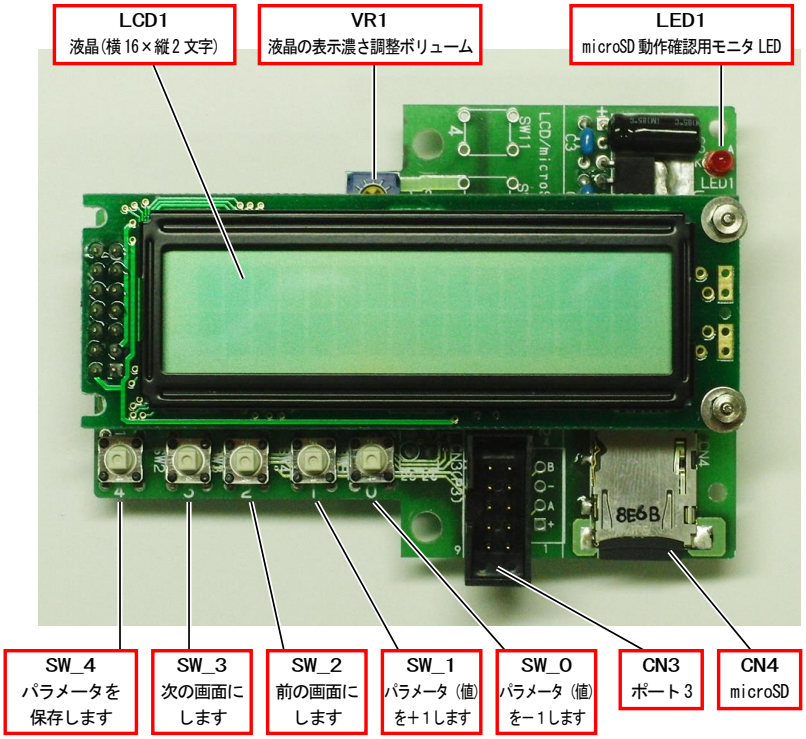
セルに分かれて走行ログが表示されます。1行が 10ms になりますので 100 行なら 1000ms の時間がかかっていることになります。**走行ログを解析して、脱輪原因を解析しましょう！**

4.15. 液晶とスイッチでパラメータを設定する

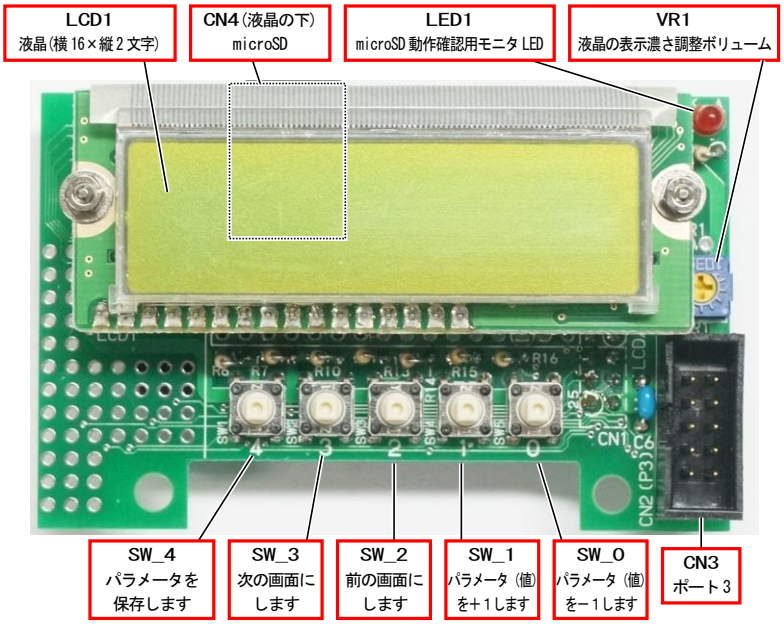
詳しくは「液晶・microSD 基板(Ver.2) kit12_38a プログラム 解説マニュアル 液晶編(R8C/38A 版)」を参照してください。ここで説明する内容は、このマニュアルの抜粋になります。

4.15.1. 液晶・microSD 基板の液晶・スイッチ部分について

液晶にパラメータを表示して、各種パラメータの設定を行うことができます。確認用にセンサの値や、ロータリエンコーダの値、A/D 値を表示させるモードもあります。外観を下記に示します。



▲液晶・microSD 基板(Ver.1)の外観



▲液晶・microSD 基板 Ver.2 の外観

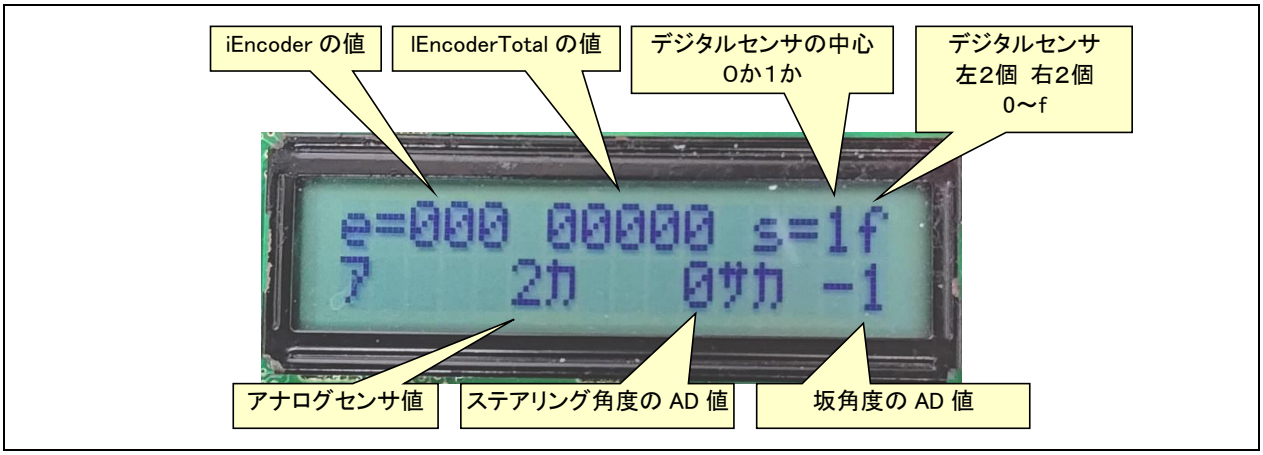
スイッチは5個あり、次のような用途で使用します。

SW_4	設定したパラメータを保存します。モータドライブ基板 TypeS のプッシュスイッチを押しても保存されます。
SW_3	パラメータ表示画面を、次の画面にします。
SW_2	パラメータ表示画面を、1つ前の画面にします。
SW_1	パラメータを＋1します。
SW_0	パラメータを－1します。

4.15.2. 設定内容

(1) 確認画面

モータドライブ基板 TypeS のプッシュスイッチ(SW3)を押しながらマイコンカーの電源スイッチを入れ、ブザーが3回鳴ったらスイッチを離します。マイコンカーの状態確認画面になります。



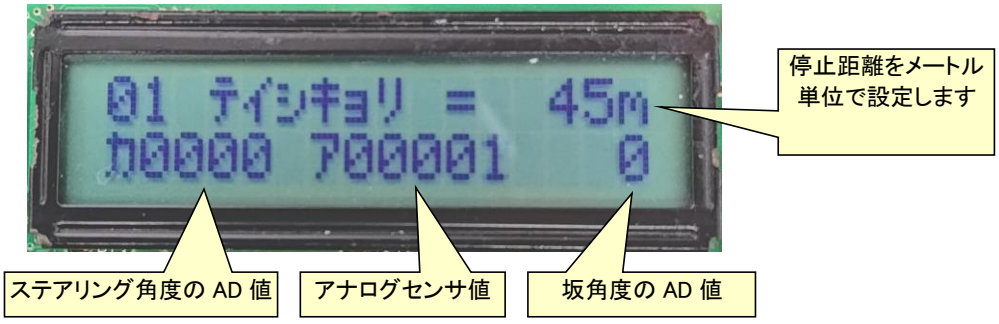
表示プログラムを下記に示します。

```
        lcdPosition( 0, 0 );
                /* 0123456789abcde f 1行16文字 */
        lcdPrintf( "p=%3d e=%3d s=%x%x",
                    pattern, iEncoder, center_inp(), sensor_inp() );
                /* 01.567 abcdef 1行16文字 */
        lcdPrintf( "ｱ%5dｶ%4dｻｶ%3d",
                    getAnalogSensor(), getServoAngle(), getSakaAngle() );
```

(2) 停止距離(画面1)

モータドライブ基板 TypeS のプッシュスイッチを押さずに電源スイッチを入れると、通常モードでマイコンカーが立ち上がり、画面1～3でパラメータ設定をすることができます。

画面1は、停止距離を設定します。メートル単位で設定します。
SW_1、SW_0 で停止距離を設定します。確認用として、ステアリング角度、アナログセンサ値、坂角度を表示します。

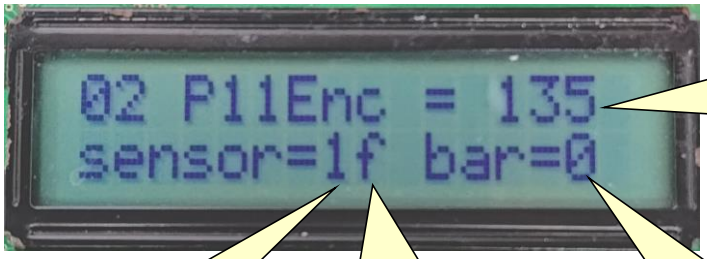


表示プログラムを下記に示します。

```
    case 1:
        /* 距離設定 */
        i = data_buff[DF_KYORI];
        if( getSwFlag(SW_1) ) {
            i++;
            if( i > 100 ) i = 100;
        }
        if( getSwFlag(SW_0) ) {
            i--;
            if( i < 1 ) i = 1;
        }
        data_buff[DF_KYORI] = i;

        /* LCD処理 */
        lcdPosition( 0, 0 );
                /* 0123456789abcdef 1行16文字 */
        lcdPrintf( "01 ﾃｲｼｷｮﾘ = %3dm", i );
                /* 01.4567.bcdef 1行16文字 */
        lcdPrintf( "ｶ%04d ｱ%05d %3d",
                    getServoAngle(), getAnalogSensor(), getSakaAngle() );

        break;
```

(3) パターン11直線のエンコーダ値(画面2)

画面2は、パターン11の直線のスピード(エンコーダ値)を設定します。ロータリエンコーダ TypeS の場合は、1m/s が 55 です。初期値は 135 で、秒速 2.5m/s の設定です。
SW_1、SW_0 でエンコーダ値を設定します。確認用として、デジタルセンサの中心、左2個右2個の16進数の値、スタートバーセンサの値を表示します。



パターン11の直線のスピード(ロータリエンコーダ値)を設定します。
1m/s で 55、2m/s で 110 です。

デジタルセンサ
中心1個 0 または 1

デジタルセンサ
左2個、右2個の値 0~f

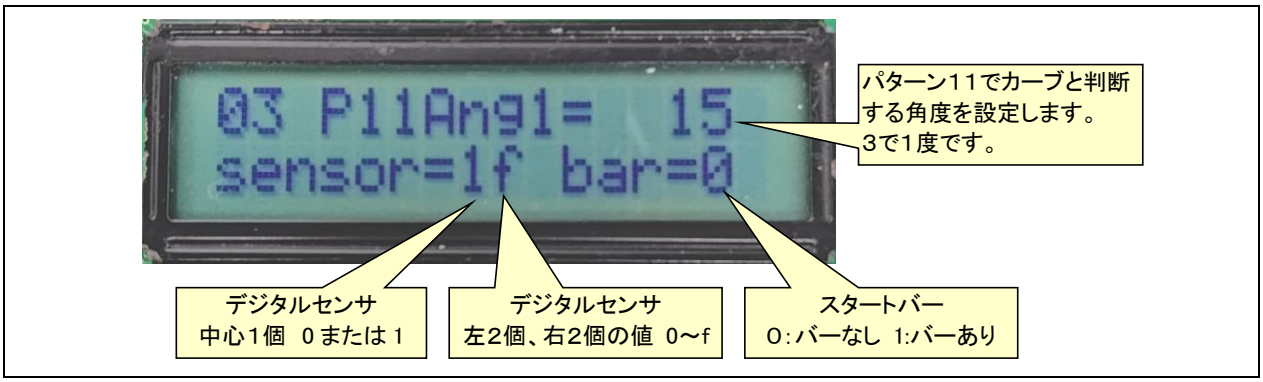
スタートバー
0:バーなし 1:バーあり

表示プログラムを下記に示します。

```
    case 2:
        /* パターン１１のエンコーダ値設定 */
        i = data_buff[DF_P11_ENC];
        if( getSwFlag(SW_1) ) {
            i++;
            if( i > 127 ) i = 127;
        }
        if( getSwFlag(SW_0) ) {
            i--;
            if( i < 0 ) i = 0;
        }
        data_buff[DF_P11_ENC] = i;

        /* LCD処理 */
        lcdPosition( 0, 0 );
                 /* 0123456789abcdef 1行16文字 */
        lcdPrintf( "02 P11Enc = %3d ", i*5 );
                 /* 01234567 8 9abcd ef 1行16文字 */
        lcdPrintf( "sensor=%x%x bar=%d ",
                    center_inp(), sensor_inp(), startbar_get() );
        break;
```

(4) パターン11の角度の値(画面3)

画面3は、パターン11でカーブと判断する角度を設定します。3で1度です。初期値は15で約5度です。5度以上になったらパターン11のカーブ処理として内外輪差をつけてカーブで曲がりやすくします。
SW_1、SW_0 で角度を設定します。確認用として、デジタルセンサの中心、左2個右2個の16進数の値、スタートバーセンサの値を表示します。



表示プログラムを下記に示します。

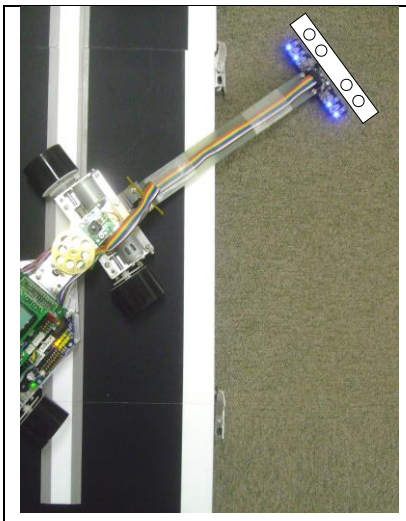
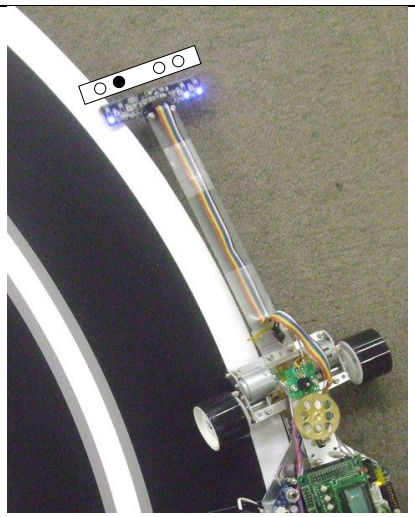
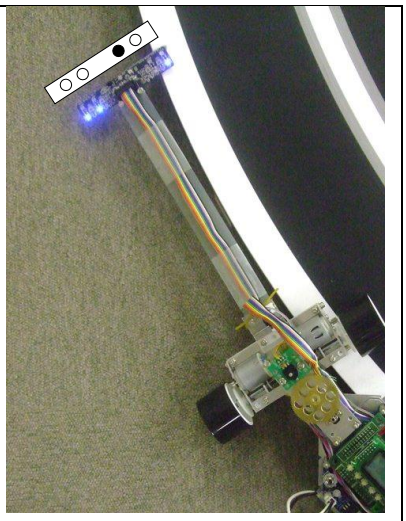
```
    case 3:
        /* パターン１１の角度１　設定 */
        i = data_buff[DF_P11_ANG1];
        if( getSwFlag(SW_1) ) {
            i++;
            if( i > 127 ) i = 127;
        }
        if( getSwFlag(SW_0) ) {
            i--;
            if( i < 0 ) i = 0;
        }
        data_buff[DF_P11_ANG1] = i;

        /* LCD処理 */
        lcdPosition( 0, 0 );
                 /* 0123456789abcdef 1行16文字 */
        lcdPrintf( "03 P11Ang1= %3d ", i );
                 /* 01234567 89abcd ef 1行16文字 */
        lcdPrintf( "sensor=%x%x bar=%d ",
                    center_inp(), sensor_inp(), startbar_get() );
        break;
```

4.16. 自動停止プログラム

マイコンカーがコースアウトしたあと、マイコンカーが暴走して壁などにぶつかり部品が壊れないようにコースアウト後、マイコンカーを停止させるプログラムを入れています。プログラム作成のポイントは、自動停止プログラムが誤動作して、順調に走行しているにもかかわらず停止しないようにすることです。

4.16.1. デジタルセンサで自動停止

マイコンカーがコースアウトしたときのデジタルセンサの状態を考えてみます。

		
0x0f	0x0d	0x0b

マイコンカーがコースアウトしたときのデジタルセンサの状態は、「0x0f」、「0x0d」、「0x0b」の状態になることが想定されます。デジタルセンサがこの状態になったら停止させます。

ただ、これらのセンサ状態は、クロスライン検出と同じです。このままでは、クロスラインを検出したのか、コースアウトしたのか、判別することができません。

実際にクロスラインを通過する時間は、マイコンカーの速度にもよりますが、数十 ms 程度です。100ms 以上反応があれば、コースアウトと判断することができます。

これらの内容をプログラム化してみます。

```
247 :          /* 脱輪時の停止処理 (デジタルセンサ)  */
248 :          b = sensor_inp();
249 :          if( b == 0x0f || b == 0x0d || b == 0x0b ) {  デジタルセンサが 0x0f または 0x0d または 0x0b で
250 :              if( check_sen_cnt >= 100 ) {             このセンサ状態が 100ms 以上続いたら
251 :                  pattern = 101;                       パターン 101(自動停止処理)へ
252 :              }
253 :          } else {
254 :              check_sen_cnt = 0;
255 :          }
```

4.16.2. ロータリエンコーダで自動停止

マイコンカーがコースアウトしたあと、コース検出センサの状態が自動停止の状態にならないと停止しません。例えばコースを広げている部屋の床の色が白やグレーの場合、デジタルセンサは自動停止の状態になりません。しかし、ロータリエンコーダは脱輪してコースの段差に引っかかっていたりすると回転しません。このとき、脱輪したと判断して、停止するようプログラムを追加してみましょう。

速度は iEncoder 変数をチェックすることで検出することができます。iEncoder 変数は、10ms 間のエンコーダパルス数が 10ms ごとに代入されます。Advanced マイコンカーに取り付けられているロータリエンコーダの 1m/s あたりの 10ms間のパルス数は 55 パルスです。今回は iEncoder 変数をチェックして、5 パルス以下(秒速約 0.1m/s 以下)であれば、マイコンカーが停止していると判断し、マイコンカーの停止処理をするようにします。ただしマイコンカー走行中に急ブレーキを掛けた瞬間、この状態になるかもしれないので、300ms 以上、5 パルス以下になったら、パターン 101 の自動停止に移行するようにします。

```
256 :  #if 1   // 駆動モータのコネクタを抜いて、手押しで確認するときは0にして無効にする
257 :          /* 脱輪時の停止処理 (ロータリエンコーダ)  */
258 :          if( iEncoder <= 5 ) {              秒速 0.1m/s(TypeS は 1m/s=55)以下なら
259 :              if( check_enc_cnt >= 300 ) {   このセンサ状態が 100ms 以上続いたら
260 :                  pattern = 101;             パターン 101(自動停止処理)へ
261 :              }
262 :          } else {
263 :              check_enc_cnt = 0;
264 :          }
265 :  #endif
```

※駆動モータのコネクタを抜いて、手押しでマイコンカー状態を確認したいとき

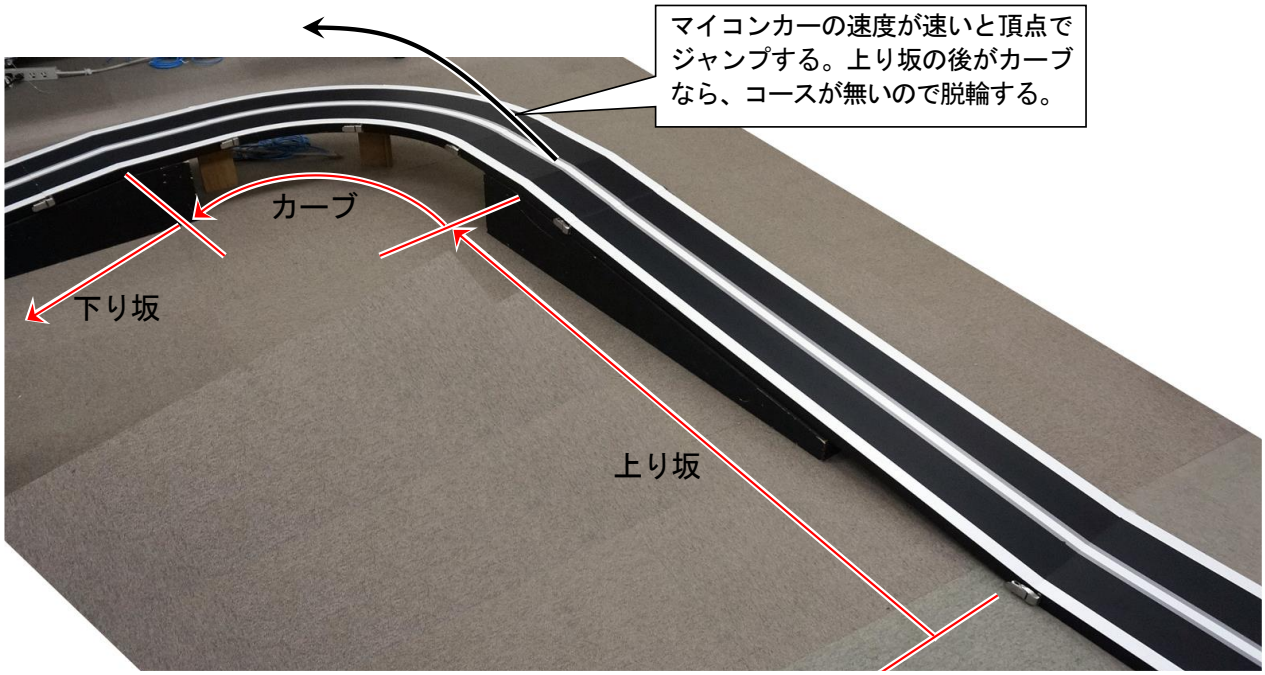
マイコンカーのスピードが速くて確認ができないので、駆動モータのコネクタを抜いて手押しでマイコンカー状態を確認したいとき、スピードが遅くてロータリエンコーダで自動停止なってしまいます。

このとき、256 行の「1」を「0」にすると、ロータリエンコーダの自動停止を無効にすることができます。「0」にした状態でプログラムを書き込んでください。確認ができたら「1」に戻して、ロータリエンコーダの自動停止を有効に戻してください。

4.17. 坂道検出用ポテンショメータを使った坂道制御

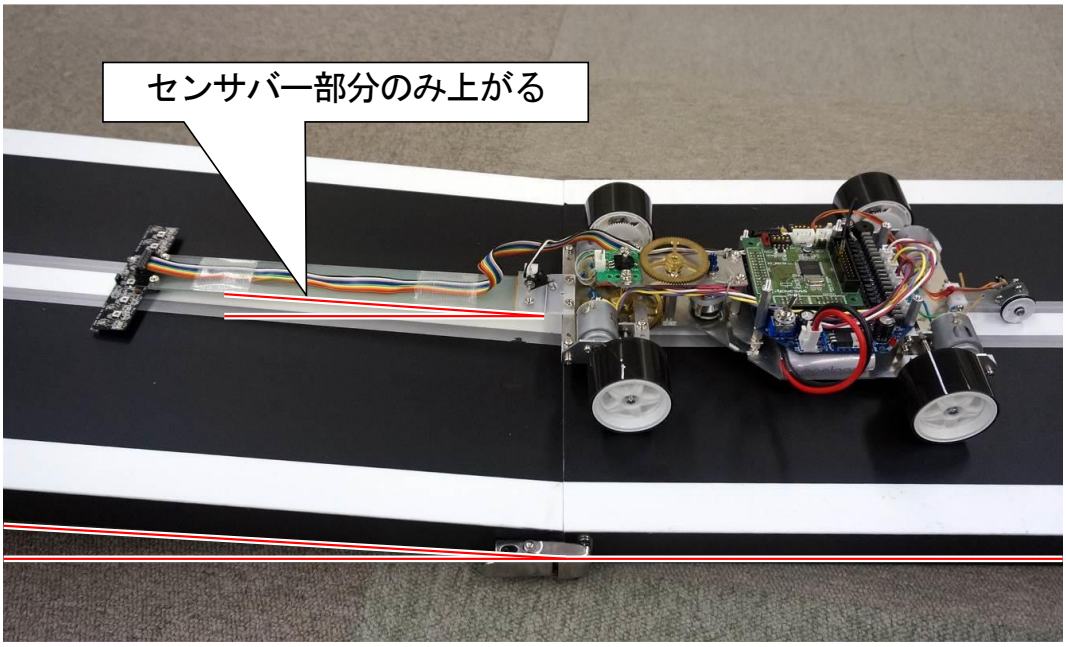
4.17.1. 概要

マイコンカーの速度が約秒速 3m/s 以上くらいになると、コースの上り坂の頂点でマイコンカーはジャンプして落ちてしまいます。坂道用ポテンショメータで上り坂を検出したら、スピードを落とすプログラムの解説をします。

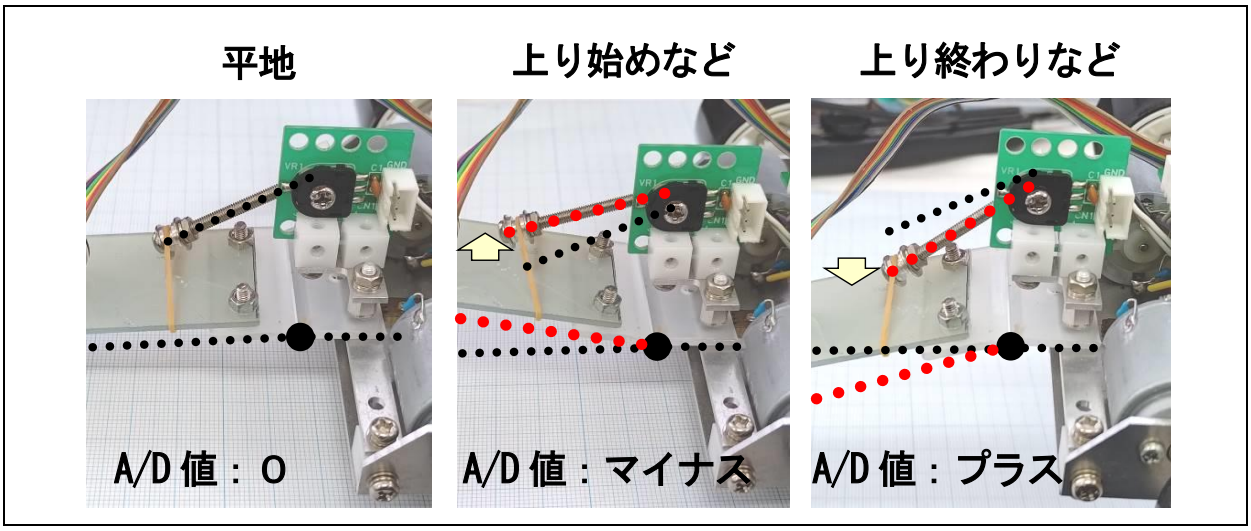


4.17.2. 上り坂制御の考え方

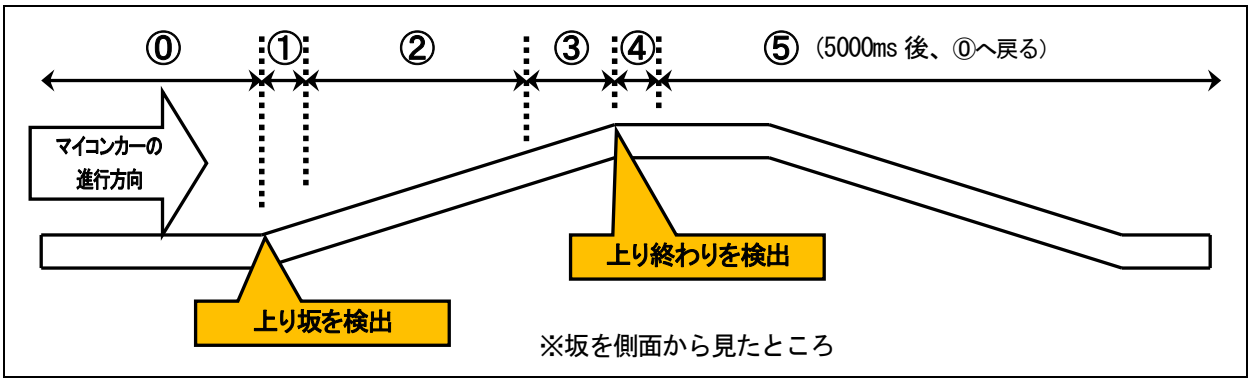
車体本体がまだ上り坂部分でなく、センサバー部分のみ上り坂にさしかかると、坂道の角度分、センサバーが上がります(下写真)。



Advanced マイコンカーには、センサバー根本に坂道用ポテンショメータを取り付けています。下記写真のように、センサバーの上下によって、ポテンショメータの値が変わります。坂道用 A/D 値が平地の値より大きくなったり小さくなったりしたら、上り坂や上り坂終わりと判断して減速するプログラムを作ります。



今回の考え方を下図に示します。



プログラム	内容
⓪	坂道検出用ポテンショメータが、上り坂を検出したかチェックします。上り坂を検出したなら、プログラム①に移ります。
①	プログラム⓪で上り坂を検出しましたが誤動作かもしれないので、10ms たってから再度上り坂を検出しているか確認します。 上り坂を検出しているならプログラム②へ移ります。検出していないなら誤動作と判断してプログラム⓪に戻ります。
②	上り坂検出後、saka_flag 変数を1にして、上り坂を走行中であることを知らせます。 すぐには上り坂の頂点にはならないので 500ms 間、上り終わり検出はしません。500ms たったらプログラム③へ移ります。
③	坂道検出用ポテンショメータが、上り坂の終わりを検出したかチェックします。上り坂の終わりを検出したなら、プログラム④に移ります。
④	プログラム③で上り坂の終わりを検出しましたが誤動作かもしれないので、10ms たってから再度、上り坂の終わりを検出しているか確認します。 上り坂の終わりを検出しているならプログラム⑤へ移ります。検出していないなら誤動作と判断してプログラム③に戻ります。
⑤	上り坂はしばらくないと判断して、5000ms 間、上り坂モードにしないために何もせずに待ちます。 5000ms たったならプログラム⓪へ戻ります。

4.17.3. プログラム

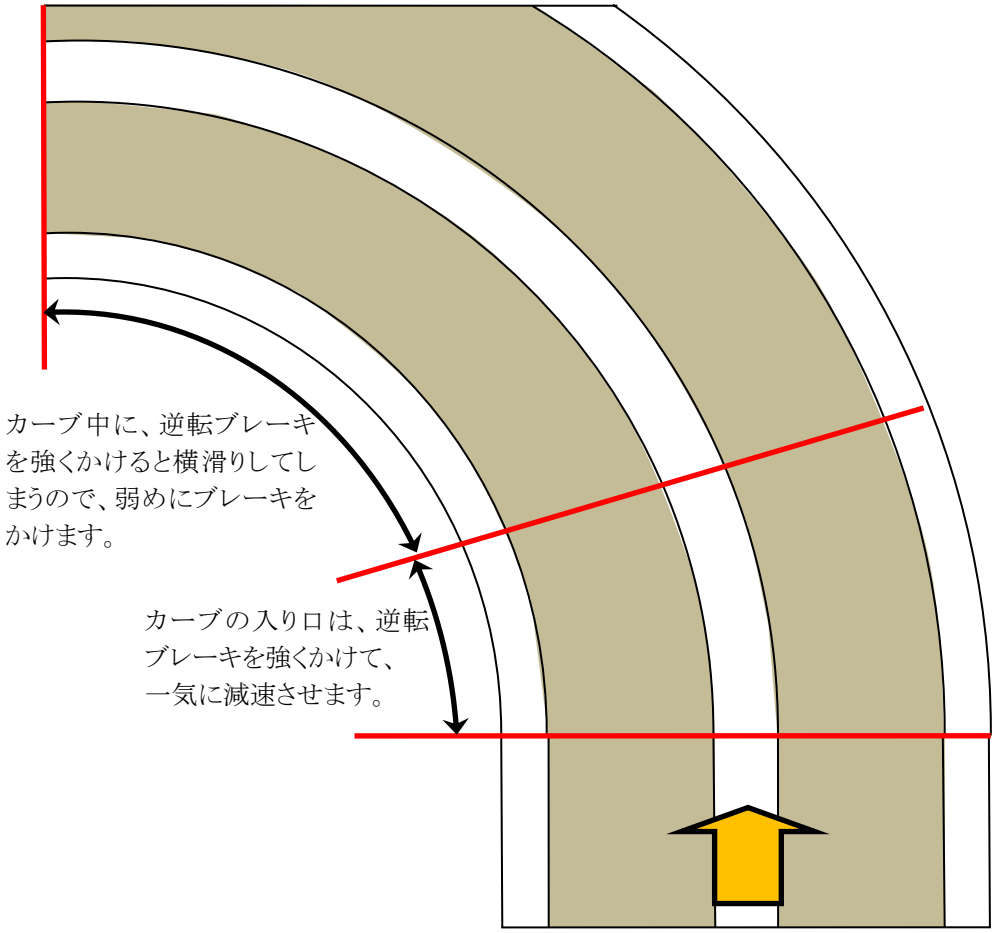
坂道チェック処理は、sakaSyori 関数で行います。この関数は main 関数の while ループの中に入れて、走行中は常に実行します。 この関数内で上り坂なら saka_flag を 1 に、上り坂でないなら 0 にします。パターン 11 のプログラムで saka_flag をチェックして、1 なら上り坂中と判断してスピードを落として走行させます。

```
1676 :  /***************************************************************************/
1677 :  /* 坂道チェック                                                            */
1678 :  /* 引数    なし                                                            */
1679 :  /* 戻り値 なし                                                             */
1680 :  /* メモ    坂道と判断するとsaka_flag = 1 坂道でないなら 0                  */
1681 :  /***************************************************************************/
1682 :  void sakaSyori( void )
1683 :  {
1684 :      // 坂の確認中はled_outでパターンを表示、確認が終わったらコメントにする
1685 :  #if 0
1686 :      led_out( ((saka_pattern/10)<<4) | (saka_pattern%10) );
1687 :  #endif
1688 :
1689 :      switch( saka_pattern ) {
1690 :      case 0:                                    図の⓪部分のプログラムです
1691 :          // 上り坂、下り坂のチェック
1692 :          if( getSakaAngle() <= -15 ) {  // 上りはA/D値が小さくなる
1693 :              cnt_saka = 0;
1694 :              saka_pattern = 1;               // 上り坂処理
1695 :          }
1696 :          break;
1697 :      case 1:                                    図の①部分のプログラムです
1698 :          // 上り坂  少し時間をおいて、再度チェック
1699 :          if( cnt_saka >= 10 ) {
1700 :              if( getSakaAngle() <= -15 ) {  // 反応あり
1701 :                  setBeepPatternS( 0xcc00 );
1702 :                  saka_flag = 1;                  // 坂フラグをON!
1703 :                  cnt_saka = 0;
1704 :                  saka_pattern = 2;
1705 :              } else {
1706 :                  // 反応なしなら誤動作と判断して戻る
1707 :                  saka_pattern = 0;
1708 :              }
1709 :          }
1710 :          break;
1711 :      case 2:                                    図の②部分のプログラムです
1712 :          // 上り坂  頂点の中間くらいから上り坂終わりのチェック
1713 :          if( cnt_saka >= 500 ) {
1714 :              cnt_saka = 0;
1715 :              saka_pattern = 3;
1716 :          }
1717 :          break;
1718 :      case 3:                                    図の③部分のプログラムです
1719 :          // 上り坂  上りの頂点をチェック
1720 :          if( getSakaAngle() >= 15 ) {
1721 :              cnt_saka = 0;
1722 :              saka_pattern = 4;
1723 :          }
1724 :          break;
1725 :      case 4:                                    図の④部分のプログラムです
1726 :          // 上り坂  少し時間をおいて、再度チェック
1727 :          if( cnt_saka >= 10 ) {
1728 :              if( getSakaAngle() >= 15 ) {   // 反応あり
1729 :                  setBeepPatternS( 0xff00 );
1730 :                  saka_flag = 0;                  // 坂フラグをOFF!
1731 :                  cnt_saka = 0;
1732 :                  saka_pattern = 5;
1733 :              } else {
1734 :                  // 反応なしなら誤動作と判断して戻る
1735 :                  saka_pattern = 3;
1736 :              }
1737 :          }
1738 :          break;
1739 :      case 5:                                    図の⑤部分のプログラムです
1740 :          // 上り坂終わり  しばらく上り坂モードには入らない
1741 :          if( cnt_saka >= 5000 ) {
1742 :              cnt_saka = 0;
1743 :              saka_pattern = 0;
1744 :          }
1745 :          break;
1746 :      }
1747 :  }
```

5. プログラムを改造してみよう

5.1. カーブ進入時の逆転ブレーキ

マイコンカーは、秒速 3.0～3.5m/s を超えるとカーブを曲がることが大変難しくなります。そこで、カーブを曲がるときの速度が秒速 3.5m/s 以上のとき、逆転ブレーキを掛けるようにします。ただし、ハンドル角度が左右 3 度以内のときは、直線と判断し、ブレーキは掛けないようにします。



太字部分が追加・変更した部分です。実際に走らせて、逆転の PWM 値や角度を調整してみてください。

```
    case 11:
        /* 通常トレース */
        servoPwmOut( iServoPwm );
        i = getServoAngle();

        if( saka_flag == 1 ) {          // 坂道フラグがONなら減速
            if( iEncoder >= 55 ) {
                motor2_f(   0,   0 );
                motor2_r(   0,   0 );
            } else {
                motor2_f(  95,  95 );
                motor2_r(  95,  95 );
            }
            break;  // case 11はここで終了
        }

        if( i >= 50 ) {
            if( iEncoder >= 110 ) {
                motor2_f(   0,   0 );
                motor2_r(   0,   0 );
            } else {
                motor2_f(  50,  80 );
                motor2_r(   0,  80 );
            }
        } else if( i <= -50 ) {
            if( iEncoder >= 110 ) {
                motor2_f(   0,   0 );
                motor2_r(   0,   0 );
            } else {
                motor2_f(  80,  50 );
                motor2_r(  80,   0 );
            }
         } else if( i >= data_buff[DF_P11_ANG1] ) {  // 初期値15  左5度以上で
            if( iEncoder >= 190 ) {                  // 3.5m/s以上なら
                motor2_f( -50, -50 );                // カーブ入り口で速いと判断して逆転ブレーキ
                motor2_r( -30, -30 );
            } else if( iEncoder >= 150 ) {
                motor2_f(   0,   0 );
                motor2_r(   0,   0 );
            } else if( iEncoder >= 109 ) {
                motor2_f( 50, 80 );
                motor2_r(  0, 80 );
            } else {
                motor2_f( diff(80), 80 );
                motor2_r( diff(80), 80 );
            }
        } else if( i <= -data_buff[DF_P11_ANG1] ) {  // 初期値-15  右5度以上で
            if( iEncoder >= 190 ) {                  // 3.5m/s以上なら
                motor2_f( -50, -50 );                // カーブ入り口で速いと判断して逆転ブレーキ
                motor2_r( -30, -30 );
            } else if( iEncoder >= 150 ) {
                motor2_f(  0,   0 );
                motor2_r(  0,   0 );
            } else if( iEncoder >= 109 ) {
                motor2_f( 80, 50 );
                motor2_r( 80,  0 );
            } else {
                motor2_f( 80, diff(80) );
                motor2_r( 80, diff(80) );
            }
        } else {
            if( iEncoder >= data_buff[DF_P11_ENC]*5 ) {
                motor2_f( 0, 0 );
                motor2_r( 0, 0 );
            } else {
                motor2_f( 100, 100 );
                motor2_r( 100, 100 );
            }
        }
```

5.2. クロスライン検出後のブレーキ

速い速度でクロスラインを検出したとき、強いブレーキを掛けないと減速しきれずクランクに突っ込んでしまい脱輪してしまいます。また、遅い速度でクロスラインを検出したとき、強いブレーキを掛けすぎるとタイムロスになってしまいます。

そこで、クロスラインを検出したときのマイコンカーの速度を検出し、その速度に応じたブレーキを掛けるようにします。

速度は、「iEncoder」変数で確認することができます。今回は、2.2m/s 以上とき、2.0m/s 以上とき、2.0m/s 未満のときの 3 つの状態に応じて、ブレーキを掛ける強さを変えるようにします。

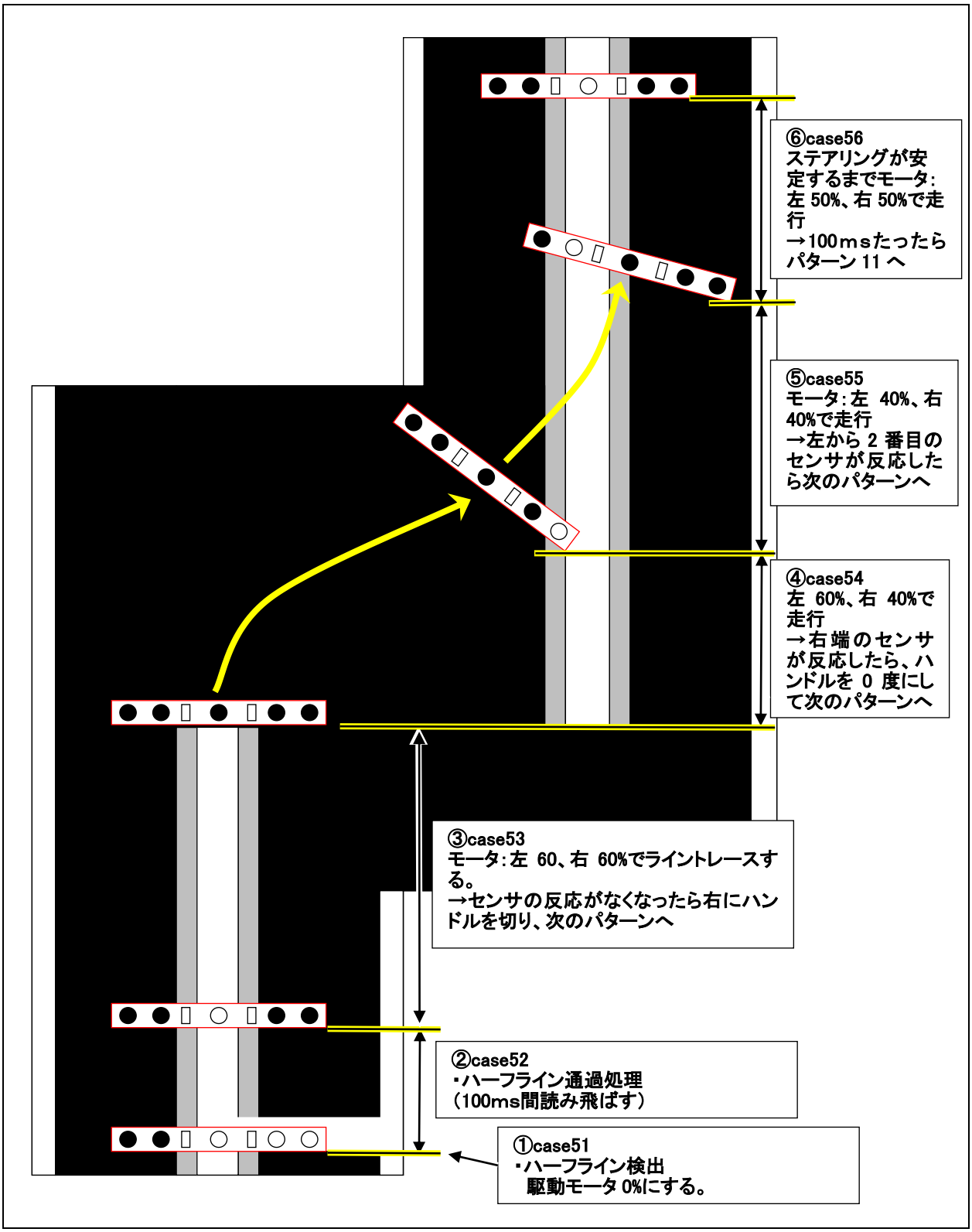
プログラム化すると、下記のようになります。**太字**部分を追加し、調整してみてください。

```
    case 21:
        /* クロスライン通過処理 */
        servoPwmOut( iServoPwm );
        led_out( 0x3 );
        if( iEncoder >= 120 ) {         // 2.2m/s 以上なら
            motor2_f( -80, -80 );
            motor2_r( -80, -80 );
        } else if( iEncoder >= 110 ) {  // 2.0m/s 以上なら
            motor2_f( 0, 0 );
            motor2_r( 0, 0 );
        } else {
            motor2_f( 80, 80 );
            motor2_r( 80, 80 );
        }
        if( cnt1 >= 50 ) {
            cnt1 = 0;
            pattern = 22;
        }
        break;
```

5.3. レーンチェンジの改良（右レーンチェンジの例）

レーンチェンジで速度を速くすると、新しい中心線を見つけたときの復帰処理がうまくいきません（脱輪するか、ハンドルがばたばたしてしまいます）。

安定して走行する方法を考えてみます。下図④のように、中心線を見つけたら0度にして進ませ、その状態でデジタルセンサの左から2番目が"1"になったら次のパターンに移行するように、もう一段階、追加してみましょう。プログラムは記載しませんので、各自で考えてみてください。

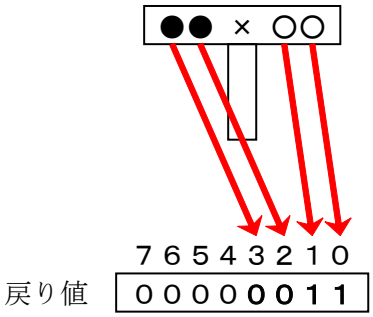


6. 走行プログラムで使用されている主な関数の説明

6.1. アナログセンサ基板 TypeS のデジタルセンサ値読み込み

```
/*************************************************************************/
/* アナログセンサ基板TypeSのデジタルセンサ値読み込み                      */
/* 引数    なし                                                          */
/* 戻り値 左端、左中、右中、右端のデジタルセンサ 0:黒 1:白                */
/*************************************************************************/
unsigned char sensor_inp( void )
{
    unsigned char sensor;

    sensor = ~p0 & 0x0f;

    return sensor;
}
```

アナログセンサ基板 TypeS のデジタルセンサ 4 個を読み込む関数です。デジタルセンサは黒で"1"、白で"0"なのでポートから読み込むときに"～"(チルダ)をつけて反転させます。

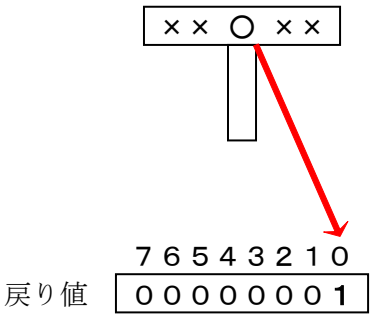


※中心のデジタルセンサ値を読み込む関数は、center_inp 関数です。

6.2. アナログセンサ基板 TypeS の中心デジタルセンサ読み込み

```
/************************************************************************/
/* アナログセンサ基板TypeSの中心デジタルセンサ読み込み                  */
/* 引数    なし                                                         */
/* 戻り値 中心デジタルセンサ 0:黒 1:白                                  */
/************************************************************************/
unsigned char center_inp( void )
{
    unsigned char sensor;

    sensor = ~p1_7 & 0x01;

    return sensor;
}
```

アナログセンサ基板 TypeS の中心デジタルセンサ 1 個を読み込む関数です。

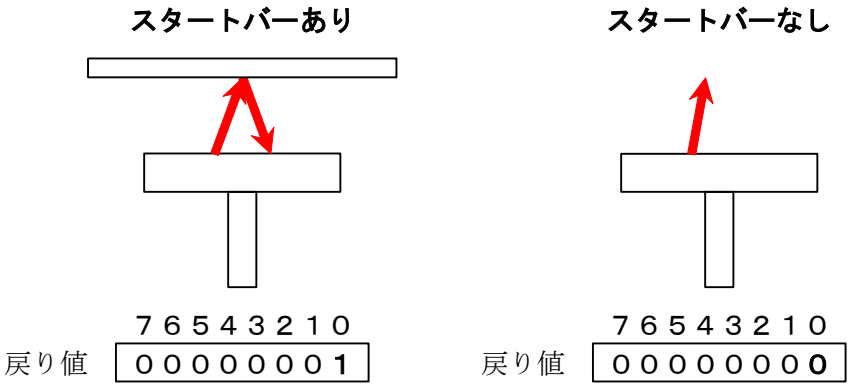


※中心以外のデジタルセンサ値を読み込む関数は、sensor_inp 関数です。

6.3. アナログセンサ基板 TypeS のスタートバー検出センサ読み込み

```
/************************************************************************/
/* アナログセンサ基板TypeSのスタートバー検出センサ読み込み               */
/* 引数    なし                                                         */
/* 戻り値 0:スタートバーなし 1:スタートバーあり                          */
/************************************************************************/
unsigned char startbar_get( void )
{
    unsigned char sensor;

    sensor = ~p1_6 & 0x01;

    return sensor;
}
```

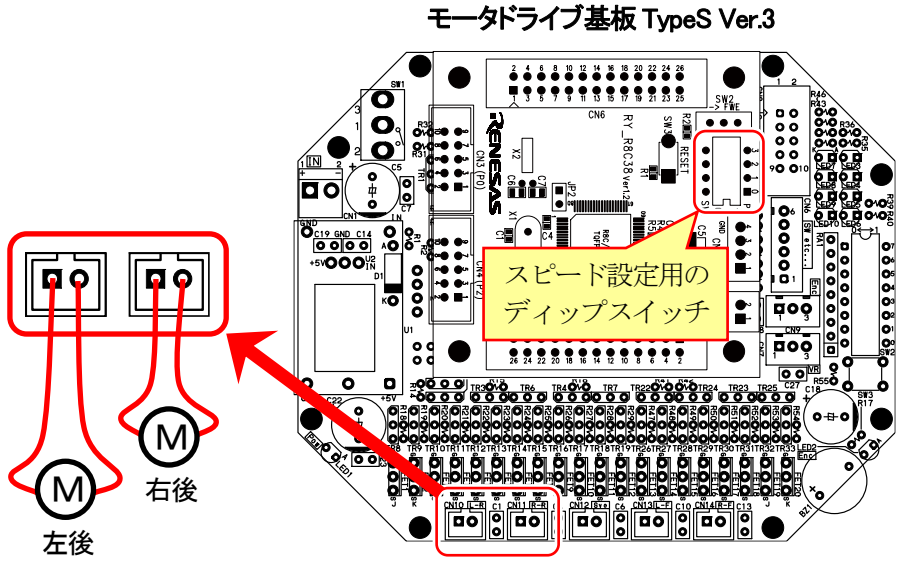
アナログセンサ基板 TypeS のスタートバー検出センサの状態を読み込む関数です。



6.4. 後輪の速度制御

```
/************************************************************************/
/* 後輪の速度制御(ディップスイッチで減速あり)                           */
/* 引数    左モータ:-100～100 , 右モータ:-100～100                      */
/*         0で停止、100で正転100%、-100で逆転100%                       */
/* 戻り値 なし                                                          */
/************************************************************************/
void motor_r( int accele_l, int accele_r )
{
    int sw_data;

    sw_data  = dipsw_get() + 5;          /* ディップスイッチ読み込み      */
    accele_l = accele_l * sw_data / 20;
    accele_r = accele_r * sw_data / 20;

    motorLR = accele_l;
    motorRR = accele_r;

    /* 左後モータ */
    if( accele_l >= 0 ) {
        p2_1 = 0;
        trdgrd0 = (long)( TRD_MOTOR_CYCLE - 2 ) * accele_l / 100;
    } else {
        p2_1 = 1;
        trdgrd0 = (long)( TRD_MOTOR_CYCLE - 2 ) * ( -accele_l ) / 100;
    }

    /* 右後モータ */
    if( accele_r >= 0 ) {
        p2_3 = 0;
        trdgrc1 = (long)( TRD_MOTOR_CYCLE - 2 ) * accele_r / 100;
    } else {
        p2_3 = 1;
        trdgrc1 = (long)( TRD_MOTOR_CYCLE - 2 ) * ( -accele_r ) / 100;
    }
}
```

モータドライブ基板 TypeS の後輪モータ 2 個を制御する関数です。



使い方を下記に示します。

```
motor_r ( 左後モータの PWM, 右後モータの PWM ) ;
```

左後モータの PWM、右後モータの PWM の値は、下記を設定することができます。

0… 停止
1～ 100… 正転の割合　100 が一番速い
-1～-100… 逆転の割合　-100 が一番速い

モータへの出力は、motor_r 関数で設定した割合がそのままモータに出力されるのではなく、下記の計算結果がモータに出力されます。

実際のモータに出力される割合＝引数×(マイコンボードのディップスイッチの値＋5)÷20

例えば、ディップスイッチの値が 10 で下記プログラムを実行したとします。

```
motor_r( 50, 100 );
```

左後モータに出力される割合　＝プログラムの割合×(ディップスイッチの値＋5)÷20
＝50×(10＋5)÷20
＝37.5
≒**37**　(小数点以下は切り捨てです)

右後モータに出力される割合　＝プログラムの割合×(ディップスイッチの値＋5)÷20
＝100×(10＋5)÷20
＝**75**

「motor_r(100, 100);」を実行したとき、ディップスイッチの値と実際に出力される PWM の関係を、下記に示します。

ディップスイッチ				10 進数	計算	実際に出力される PWM の割合
P1_3	P1_2	P1_1	P1_0			
0	0	0	0	0	5/20	25%
0	0	0	1	1	6/20	30%
0	0	1	0	2	7/20	35%
0	0	1	1	3	8/20	40%
0	1	0	0	4	9/20	45%
0	1	0	1	5	10/20	50%
0	1	1	0	6	11/20	55%
0	1	1	1	7	12/20	60%
1	0	0	0	8	13/20	65%
1	0	0	1	9	14/20	70%
1	0	1	0	10	15/20	75%
1	0	1	1	11	16/20	80%
1	1	0	0	12	17/20	85%
1	1	0	1	13	18/20	90%
1	1	1	0	14	19/20	95%
1	1	1	1	15	20/20	100%

6.5. 後輪の速度制御 2 ディップスイッチには関係しない motor 関数

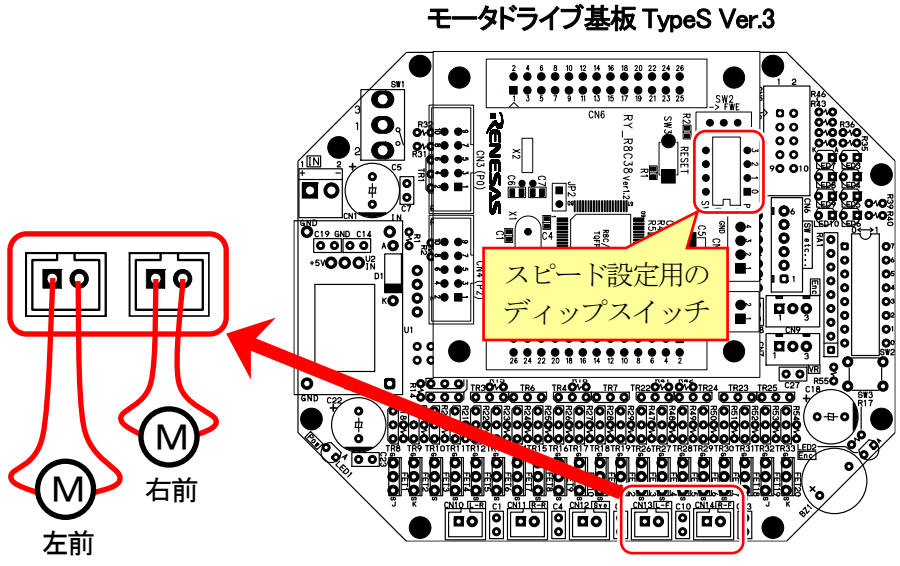
```
/**************************************************************************/
/* 後輪の速度制御2 ディップスイッチには関係しないmotor関数                */
/* 引数    左モータ:-100～100 , 右モータ:-100～100                        */
/*         0で停止、100で正転100%、-100で逆転100%                         */
/* 戻り値 なし                                                            */
/**************************************************************************/
void motor2_r( int accele_l, int accele_r )
{
    motorLR = accele_l;
    motorRR = accele_r;

    /* 左後モータ */
    if( accele_l >= 0 ) {
        p2_1 = 0;
        trdgrd0 = (long)( TRD_MOTOR_CYCLE - 2 ) * accele_l / 100;
    } else {
        p2_1 = 1;
        trdgrd0 = (long)( TRD_MOTOR_CYCLE - 2 ) * ( -accele_l ) / 100;
    }

    /* 右後モータ */
    if( accele_r >= 0 ) {
        p2_3 = 0;
        trdgrc1 = (long)( TRD_MOTOR_CYCLE - 2 ) * accele_r / 100;
    } else {
        p2_3 = 1;
        trdgrc1 = (long)( TRD_MOTOR_CYCLE - 2 ) * ( -accele_r ) / 100;
    }
}
```

motor_r 関数は、ディップスイッチの割合でモータに出力する割合をさらに落としましたが、motor2_r 関数は、プログラムの引数どおりの割合をモータに出力します。

6.6. 前輪の速度制御

```
/************************************************************************/
/* 前輪の速度制御(ディップスイッチで減速あり)                           */
/* 引数    左モータ:-100～100 , 右モータ:-100～100                      */
/*         0で停止、100で正転100%、-100で逆転100%                       */
/* 戻り値 なし                                                          */
/************************************************************************/
void motor_f( int accele_l, int accele_r )
{
    int sw_data;

    sw_data  = dipsw_get() + 5;         /* ディップスイッチ読み込み     */
    accele_l = accele_l * sw_data / 20;
    accele_r = accele_r * sw_data / 20;

    motorLF = accele_l;
    motorRF = accele_r;

    /* 左前モータ */
    if( accele_l >= 0 ) {
        p2_0 = 0;
    } else {
        p2_0 = 1;
        accele_l = -accele_l;
    }
    if( accele_l <= 5 ) {
        trcgrb = trcgrb_buff = trcgra;
    } else {
        trcgrb_buff = (unsigned long)(TRC_MOTOR_CYCLE-2) * accele_l / 100;
    }

    /* 右前モータ */
    if( accele_r >= 0 ) {
        p2_7 = 0;
    } else {
        p2_7 = 1;
        accele_r = -accele_r;
    }
    if( accele_r <= 5 ) {
        trcgrd = trcgrd_buff = trcgra;
    } else {
        trcgrd_buff = (unsigned long)(TRC_MOTOR_CYCLE-2) * accele_r / 100;
    }
}
```

モータドライブ基板 TypeS の前輪モータ 2 個を制御する関数です。



使い方を下記に示します。

```
motor_f ( 左前モータの PWM, 右前モータの PWM ) ;
```

左前モータの PWM、右前モータの PWM の値は、下記を設定することができます。

```
      0…  停止
 1～ 100…  正転の割合   100 が一番速い
-1～-100…  逆転の割合  -100 が一番速い
```

モータへの出力は、motor_f 関数で設定した割合がそのままモータに出力されるのではなく、下記の計算結果がモータに出力されます。

実際のモータに出力される割合＝引数×(マイコンボードのディップスイッチの値＋5)÷20

例えば、ディップスイッチの値が 12 で下記プログラムを実行したとします。

```
motor_f( 50, 100 );
```

左前モータに出力される割合　＝プログラムの割合×(ディップスイッチの値＋5)÷20
＝50×(12＋5)÷20
＝42.5
≒**42**　(小数点以下は切り捨てです)

右前モータに出力される割合　＝プログラムの割合×(ディップスイッチの値＋5)÷20
＝100×(12＋5)÷20
＝**85**

「motor_f(100, 100);」を実行したとき、ディップスイッチの値と実際に出力される PWM の関係を、下記に示します。

ディップスイッチ				10 進数	計算	実際に出力される PWM の割合
P1_3	P1_2	P1_1	P1_0			
0	0	0	0	0	5/20	25%
0	0	0	1	1	6/20	30%
0	0	1	0	2	7/20	35%
0	0	1	1	3	8/20	40%
0	1	0	0	4	9/20	45%
0	1	0	1	5	10/20	50%
0	1	1	0	6	11/20	55%
0	1	1	1	7	12/20	60%
1	0	0	0	8	13/20	65%
1	0	0	1	9	14/20	70%
1	0	1	0	10	15/20	75%
1	0	1	1	11	16/20	80%
1	1	0	0	12	17/20	85%
1	1	0	1	13	18/20	90%
1	1	1	0	14	19/20	95%
1	1	1	1	15	20/20	100%

6.7. 前輪の速度制御 2 ディップスイッチには関係しない motor 関数

```
/************************************************************************/
/* 前輪の速度制御2 ディップスイッチには関係しないmotor関数               */
/* 引数    左モータ:-100～100 , 右モータ:-100～100                      */
/*         0で停止、100で正転100%、-100で逆転100%                       */
/* 戻り値 なし                                                          */
/************************************************************************/
void motor2_f( int accele_l, int accele_r )
{
    motorLF = accele_l;
    motorRF = accele_r;

    /* 左前モータ */
    if( accele_l >= 0 ) {
        p2_0 = 0;
    } else {
        p2_0 = 1;
        accele_l = -accele_l;
    }
    if( accele_l <= 5 ) {
        trcgrb = trcgrb_buff = trcgra;
    } else {
        trcgrb_buff = (unsigned long)(TRC_MOTOR_CYCLE-2) * accele_l / 100;
    }

    /* 右前モータ */
    if( accele_r >= 0 ) {
        p2_7 = 0;
    } else {
        p2_7 = 1;
        accele_r = -accele_r;
    }
    if( accele_r <= 5 ) {
        trcgrd = trcgrd_buff = trcgra;
    } else {
        trcgrd_buff = (unsigned long)(TRC_MOTOR_CYCLE-2) * accele_r / 100;
    }
}
```

motor_f 関数は、ディップスイッチの割合でモータに出力する割合をさらに落としましたが、motor2_f 関数は、プログラムの引数どおりの割合をモータに出力します。

6.8. ディップスイッチの値とモータ出力

「motor_r(100, 100);」、または「motor_f(100, 100);」を実行したとき、ディップスイッチの値と実際に出力される PWM の関係を、下記に示します。

ディップスイッチ				10 進数	計算	実際に出力される PWM の割合
P1_3	P1_2	P1_1	P1_0			
0	0	0	0	0	5/20	25%
0	0	0	1	1	6/20	30%
0	0	1	0	2	7/20	35%
0	0	1	1	3	8/20	40%
0	1	0	0	4	9/20	45%
0	1	0	1	5	10/20	50%
0	1	1	0	6	11/20	55%
0	1	1	1	7	12/20	60%
1	0	0	0	8	13/20	65%
1	0	0	1	9	14/20	70%
1	0	1	0	10	15/20	75%
1	0	1	1	11	16/20	80%
1	1	0	0	12	17/20	85%
1	1	0	1	13	18/20	90%
1	1	1	0	14	19/20	95%
1	1	1	1	15	20/20	100%

6.9. ステアリングモータ角度の取得

```
/*************************************************************************/
/* ステアリング角度取得                                                  */
/* 引数    なし                                                          */
/* 戻り値 ハンドル角度のA/D値                                            */
/*************************************************************************/
int getServoAngle( void )
{
    return( ad2 - iAngle0 );
}
```

ステアリングモータの角度は、AN14(P7_2)端子に接続されているポテンショメータの A/D 値で分かります。iAngle0 は、0 度のときの A/D 変換値を入れておきます。例えば、角度が 0 度のとき A/D 変換値が 454 なら、iAngle0 変数に 454 を代入すると、戻り値は下記のようになります。

戻り値 ＝ A/D 変換値 － 0 度のときの A/D 変換値
＝ 454 － 454
＝ 0

今回のマイコンカーは左右 43 度ずつハンドルが切れました。中心と左右最大にハンドルを切ったときの電圧を測ります。テスタでポテンショメータの電圧を計った結果、

左いっぱい…2.84V　中心…2.22V　右いっぱい…1.59V

となりました(下左図)。5.00V が 1023 なので、それぞれの電圧を A/D 値に変換すると、

左いっぱい…2.84/5*1023=581　中心…2.22/5*1023=454　右いっぱい…1.59/5*1023=325

となります(下中図)。
796 行の iAngle0 変数には、0 度のときの A/D 値を入れておきます。iAngle0 変数に 454 の値を入れると、

左いっぱい…+127　中心…0　右いっぱい…−129

となります(下右図)。