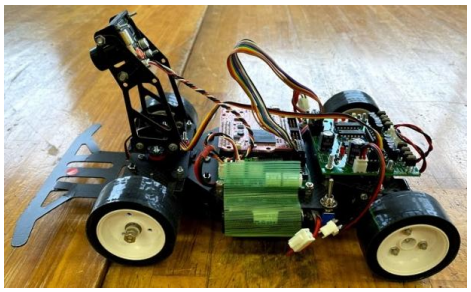
Camera クラスマイコンカーの安定化及び高速化について

令和 3 年 2 月 23 日

熊本県立熊本工業高等学校　中村　彰男

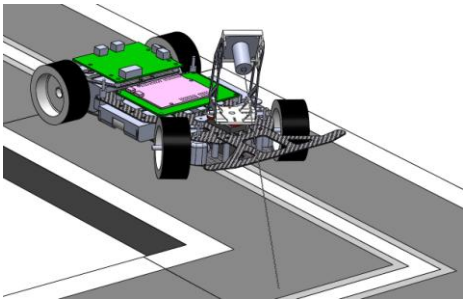
1　はじめに



マイコンカーラリー大会に，2020 年から Camera クラスが新設された。このクラスはカメラを搭載し，画像認識によりライントレースを行う技術を競うものである。より実車に近い制御となる。車体の制限は Basic クラスよりも厳格になり，車体を構成する様々なパーツはほとんどが指定のものを使用しなければならない。これまで以上にプログラムや制御方法に工夫を求められるものとなる。

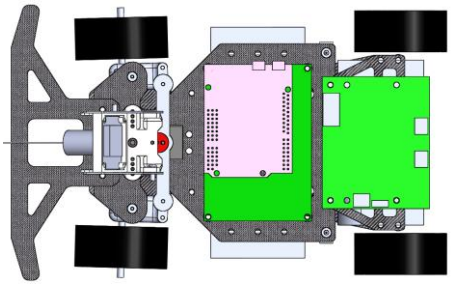
まだ新設されて間もないクラスのため，全国大会の完走率もまだ低く，これからが面白くなることが予想される。

2　車体について

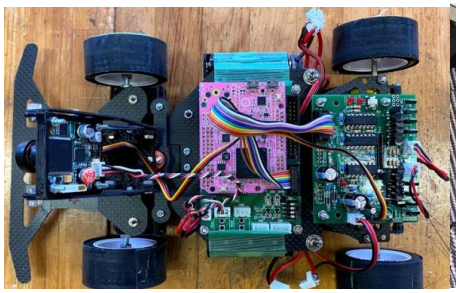


車体は SolidWorks を利用して設計を行った。今回初の試みとなるアッカーマン・ジャントー機構のステアリング機構とした。NC 切削と 3D プリンタを駆使することで，意外と思い通りの動きを実現できた。ハンドルの最大切れ角も CAD 上で確認できるので，非常に効率よい設計ができた。

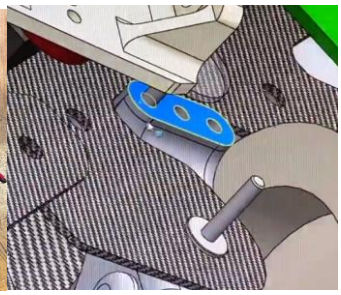
3D プリンタを使用した部品はカメラ固定用のパーツ 2 点と，ステアリング機構の心臓部と言える車軸を固定するパーツ 2 点。その他は全て 1mm 厚のカーボン板を NC 切削機械で切削した。



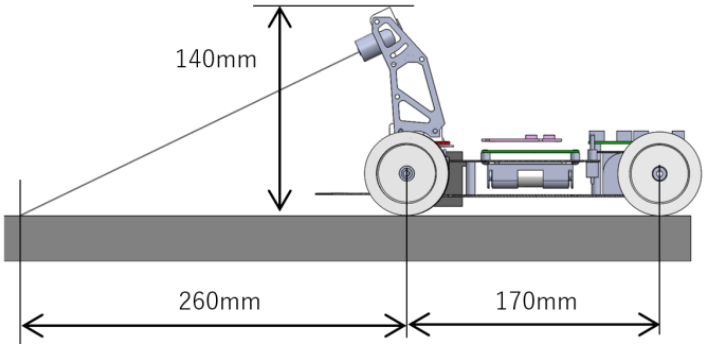
3D-CAD でモデリング



完成した実物



CAD で干渉チェック



3　2次元画像のログ記録について

Camera クラスは 2 次元画像を取り込み，その情報からいかに上手にライントレースさせるかを追求するものである。当然，どのように見えているかを正確に把握することがロボットを意のままに制御する第一歩となる。まず最初に取り組むべき事は，カメラで得た画像を可視化することである。

そこで，走行中のモータ出力やハンドル指定角に加え，カメラで得た 2 次元画像を SD カードに記録していくことにした。ログ記録間隔はカメラの情報を取得する毎の 16.67ms 毎とする。ただし，SD カードへの書き込み処理のオーバーヘッドが心配なため，走行中は全て RAM に記憶しておき，走行後に SD カードへ書き出すようにしている。これは指定マイコンである GR-Peach が膨大な RAM（10MB）を備えているため実現できたことである。現在は 3 分間の走行データを記録できるようメモリ確保をしている。

これによって，カメラが捉えた画像を解析することが可能となり，コースアウトした原因をはっきりと特定できたり，高速化への新たなアイデアを生み出す原動力となった。

（1）SD カードへのログ記録について

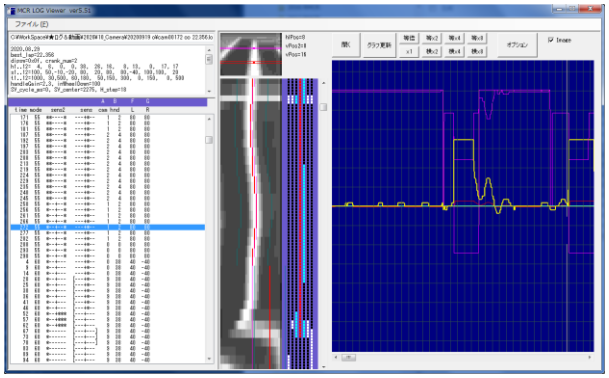
GR-Peach 基板にはマイクロ SD カードスロットが最初から付いているので，プログラムを作成するだけで SD カードを利用できる。プログラムについても，使いやすいライブラリが使用できるため，ネット上のサンプルプログラムを参考にするだけで簡単に作成することができた。

1 回の走行につき 1 つのファイルを作成し，走行後はマイクロ SD カードを PC に挿して読み出す。

（2）ログビューアについて

本校では以前よりログデータ解析用の Windows アプリを開発し，利用してきた。今回，Camera クラスの 2 次元画像を表示できるよう改良した。全てのソースコードと実行ファイルは github にて公開している。誰でも自己責任で使用でき，ソースコードの改変・再配布等も自由にできるものとする。

https://github.com/akio21r/MCRLogViewer



（3）サンプルプログラム

** ログ記録関連クラスの使い方 **

1. 他のファイル群と同じ場所に下記ファイルを追加
   210_log.h , 211_log.cpp
2. プロジェクトに下記ファイルを追加
   211_log.cpp
3. 最初に sdLog.Init(); を実行
4. スタート直後に sdLog.recStart(); を実行し、ログ記録処理を開始する。
5. intTimer() 関数内で sdLog.Rec(); を実行し、定期的にログデータを記録する。
6. 動作停止後に sdLog.print_data(); を実行し、内部データを SD カードに書きだす。

210_log.h（ログ関連プログラム　ヘッダファイル）

```
//================================================================================
//SD カードへのログ記録
//================================================================================
#pragma once

#defineLOG_VERSION          52                          //ログのバージョン 51
#define LOG_RECORD_BYTES    14                          //1 レコードのバイト数 12
#defineTXT_SECTORSIZE       4                           //TXT 領域のセクタ数
#defineLOG_RECORD_BYTES_A   (LOG_RECORD_BYTES + GASO_HW * GASO_VW / 2)
#defineLOG_GASO_BYTES       (2 + 16*24)                 //画素データ 2 + 16*24
#defineLOG_NUM              (unsigned long)7200         //約 2 分間（16.67ms 毎の記録）

//--------------------------------------------------------------------
//ログ記録関連
//--------------------------------------------------------------------
struct SDLOG_T{                                         //時系列記録ログ
       unsigned char        mode;                       //モード mode
       unsigned char        sens;                       //デジタルセンサ 00011000
       unsigned char        sens2;                      //遠方のセンサ   10011001
       unsigned int         cnt0;                       //時間
       signed char          handle;                     //ステア角
       signed char          powL, powR;                 //出力
       unsigned char        halfLine;                   //ハーフラインの検出状態
       signed char          center;                     //Cente 値  -16～0～16
       unsigned char        hl_cIndex;                  //halfLine | centerIndex;
       signed char          centerIndex2;               //centerIndex2
       unsigned char        ex1,ex2;                    //ext
       unsigned char        gaso[LOG_GASO_BYTES];       //画素データ 2 + 16*24
};

//================================================================================
//ログ記録関連クラス
//================================================================================
class SDLOG{
private:
       struct SDLOG_T *data;                            //ログデータ本体へのポインタ
       unsigned int  log_p;                             //ログデータの index
       void          writeLogData();                    //SD Card へのログ出力
       int           sdPrintf(const char *fmt, ...);    //SDCard への printf

public:
       bool          sdEnabled;                         //SD Card 使用可
       bool          recEnabled;                        //ログ記録許可
                     SDLOG(void);                       //コンストラクタ
       void          Init();                            //SD Card init
       void          Rec();                             //ログ記録
       void          recStart();                        //バイナリログ記録スタート
       void          recEnd();                          //バイナリログ記録終了
       void          print_data();                      //SD Card への設定値等出力
};
extern SDLOG sdLog;
```

211_log.cpp（ログ関連プログラム本体）

```
//================================================================================
//SD カードへのログ記録
//================================================================================
#include "mbed.h"
#include "iodefine.h"
#include "DisplayBace.h"
#include "image_process.h"
#include "FATFileSystem.h"                      //FAT file system
#include "SDBlockDevice_GRBoard.h"              //SD Card
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include "000_setup.h"
#include "300_camera.h"                         //Camera クラス
#include "210_log.h"                            //Log クラス
```

```
//=====================================================================
//グローバル変数
//=====================================================================
struct SDLOG_T               LogData[LOG_NUM];     //ログデータ本体
FILE                         *fp;                  //FILE ポインタ

//=====================================================================
//インスタンス生成
//=====================================================================
SDLOG                        sdLog;                //Log クラスのインスタンス生成
FATFileSystem                fs("storage");        //FAT file system
SDBlockDevice_GRBoard sd;                          //SD Card

//=====================================================================
//コンストラクタ
//=====================================================================
SDLOG::SDLOG(void)
{
       data                  = LogData;            //ログデータ本体へのポインタ
       log_p                 = 0;                  //ログデータの index
       sdEnabled             = false;              //SD Card 使用不可
       recEnabled            = false;              //ログ記録不許可
}

//=====================================================================
//SD Card Init
//=====================================================================
void SDLOG::Init(void)
{
       if(sd.connect()){                           //SD に接続
              fs.mount(&sd);
              sdEnabled = true;
       }
       else{
              sdEnabled = false;
              led_rgb.setMode(LED_MODE_SDERROR);
       }
}

//=====================================================================
//バイナリログ記録　カメラ情報入力後に 16.67ms 毎ログ記録
//=====================================================================
void SDLOG::Rec()
{
       if(!sdLog.recEnabled)                       //ログ記録不許可の場合は帰る
              return;

       if(log_p < LOG_NUM){
              data[log_p].mode             = ms.mode;
              data[log_p].sens             = cam.getSens();
              data[log_p].sens2            = cam.getSens2();
              data[log_p].cnt0             = (unsigned int)ms.cnt1;
              data[log_p].handle           = (signed char)ms.angle;
              data[log_p].powL             = (signed char)ms.motL;
              data[log_p].powR             = (signed char)ms.motR;
              data[log_p].center           = (signed char)ms.center;
              data[log_p].hl_cIndex        = (cam.getHalfLine() << 6)
                                                  | cam.getCenterIndex();
              data[log_p].centerIndex2     = cam.getCenterIndex2();

              data[log_p].halfLine         = (cam.getHalfLine1() << 6)
                                                  | (cam.getHalfLine2() << 3)
                                                  | (cam.getHalfLine3());

              data[log_p].ex1              = (unsigned char)cam.ex1;
              data[log_p].ex2              = (unsigned char)cam.ex2;

              //画素データ記録（上位 4 ビットのみ記録し，２画素分を１バイトに収める）
              int n = 0;
              unsigned char d1, d2;
              for(int y=0; y<GASO_VW; y++){
                     for(int x=0; x<GASO_HW; x+=2){
                            d1 = ImageComp_B[GASO_HW * y + x  ] & 0xf0;
                            d2 = ImageComp_B[GASO_HW * y + x+1] >> 4;
                            data[log_p].gaso[n++] = d1 | d2;
                     }
```

```
		}

		if(++log_p >= LOG_NUM){
			sdLog.recEnd();				//バイナリログ記録終了
			led_out(1);
		}
	}
}

//=====================================================================
//SD Card へログデータ出力
//=====================================================================
void SDLOG::writeLogData(void)
{
	char logBuff[LOG_RECORD_BYTES];			//SD ログ記録用バッファ
	unsigned int d;

	if(!sdEnabled) return;

	//---------------------------------------------
	//ファイルポインタを TXT_SECTORSIZE セクタ分後方へ移動
	fseek(fp, 512L * TXT_SECTORSIZE, SEEK_SET);

	//---------------------------------------------
	for(unsigned int i=0; i< log_p; i++){
		logBuff[ 0] = data[i].mode;			//モード

		d = data[i].cnt0;				//走行時間[ms]
		logBuff[ 1] = (d >> 8) & 0xff;
		logBuff[ 2] = d & 0xff;

		logBuff[ 3] = data[i].sens;			//デジタルセンサ 00011000
		logBuff[ 4] = data[i].handle;			//指定ハンドル角
		logBuff[ 5] = data[i].powL;			//後左出力
		logBuff[ 6] = data[i].powR;			//後右出力
		logBuff[ 7] = data[i].center;			//カメラのセンター値
		logBuff[ 8] = data[i].hl_cIndex;		//halfLine | centerIndex
		logBuff[ 9] = data[i].sens2;			//遠方のセンサ   10011001
		logBuff[10] = data[i].centerIndex2;		//CenterIndex2(遠方)
		logBuff[11] = data[i].halfLine;			//halfLine
		logBuff[12] = data[i].ex1;			//ext
		logBuff[13] = data[i].ex2;			//ext

		fwrite(logBuff, 1, LOG_RECORD_BYTES, fp);		//log
		fwrite(data[i].gaso, 1, GASO_HW * GASO_VW / 2, fp);	//画素
	}

	//終了コード-1 で埋める
	for(int i=0; i<LOG_RECORD_BYTES; i++){
		logBuff[i] = -1;
	}
	fwrite(logBuff, 1, LOG_RECORD_BYTES, fp);

	for(int i=0; i<GASO_HW * GASO_VW / 2; i++){
		data[0].gaso[i] = -1;
	}
	fwrite(data[0].gaso, 1, GASO_HW * GASO_VW / 2, fp);
}

//=====================================================================
//ログ記録開始
//=====================================================================
void SDLOG::recStart(void)
{
	sdLog.recEnabled = true;
}

//=====================================================================
//ログ記録終了
//=====================================================================
void SDLOG::recEnd(void)
{
	sdLog.recEnabled = false;
}
```

```
//=====================================================================
//SD カードにテキストデータを printf で出力
//=====================================================================
int SDLOG::sdPrintf(const char *fmt, ...)
{
	va_list argptr;
	int		ret = 0;

	char lbuf[80];

	va_start(argptr, fmt);
	ret = vsprintf( lbuf, fmt, argptr );
	va_end(argptr);

	if( ret > 0 ){                                  //vsprintf が正常なら fprintf へ転送
		if(sdEnabled) fprintf(fp, lbuf);
	}
	return ret;
}

//=====================================================================
//走行終了後，SD Card へログデータ出力
//RAM に溜め込んだデータを SD カードへ吐き出す処理
//=====================================================================
void SDLOG::print_data(void)
{
	char cbuf[9];
	unsigned long cntA_int_tmp = ms.cntA_int;
	unsigned long cntA_alw_tmp = ms.cntA_alw;

	ms.flagA.bit.stop = 1;
	sdLog.recEnd();                                 //RAM へのログ記録終了

	trace_mode(0);
	motor2(0, 0);
	ms.mode = 7;
	led_rgb.setMode(LED_MODE_STOP);

	if(!sdEnabled){
		led_rgb.setMode(LED_MODE_SDERROR);
		while(1) always_process();                  //SD が使えなければここで動作停止
	}

	//-----------------------------------------------
	//ファイル名の決定（ディレクトリ中のファイル数 +1 ）
	char fname[32];                                 //ファイル名
	int  no=0;                                      //ファイル番号
	char DIRPATH[]="/storage";
	DIR *dir;
	struct dirent *entry;
	dir = opendir(DIRPATH);
	if ( dir != NULL ) {
		while ( (entry = readdir(dir)) != NULL ) {
			no++;
		}
	}
	closedir(dir);

	sprintf(fname, "/storage/cam%05d.log", no);

	//-----------------------------------------------
	//ファイルオープン
	if(fp = fopen(fname, "wb")){
		sdEnabled = true;
	}
	else{
		sdEnabled = false;
		led_rgb.setMode(LED_MODE_SDERROR);
		while(1) always_process();   //動作停止
	}

	//-----------------------------------------------
	led_rgb.setMode(LED_MODE_SD);
	led_out(3);

	//-----------------------------------------------
```

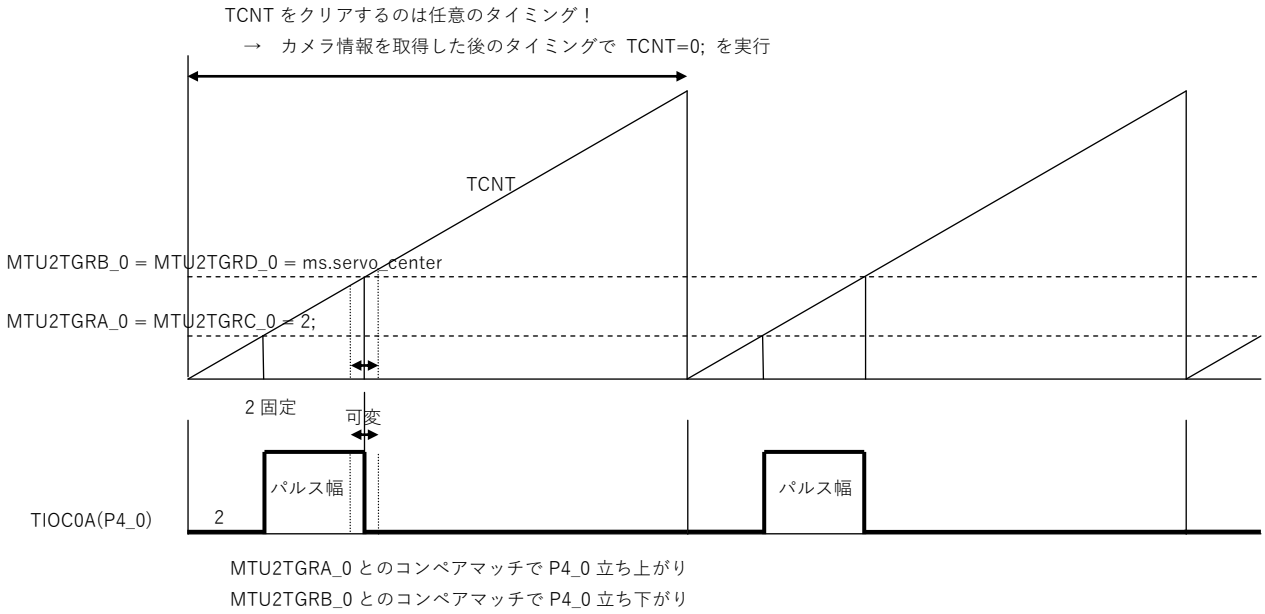
```
        //SD へデータ出力
        sdPrintf("#%03d¥r¥n", LOG_VERSION);         //バージョン記録
        sdPrintf("%03d¥r¥n", LOG_RECORD_BYTES);     //1 レコードのバイト数
        sdPrintf("%d¥r¥n", TXT_SECTORSIZE);         //TXT 領域のセクタサイズ
        sdPrintf("%s¥r¥n", PROGRAM_VERSION);        //プログラム ver.

        //---------------------------------------------
        if(ms.best_lap == 0xffffffff)
                sprintf(cbuf, "--.---");
        else
                sprintf(cbuf, "%lu.%03lu", ms.best_lap/1000UL, ms.best_lap%1000UL);
        sdPrintf("best_lap=%s¥r¥n", cbuf);

        sdPrintf("handleGain=%.1f, inWheelDown=%d, detect_HL,CR,LC=%d,%d,%d¥r¥n",
                ms.handleGain, ms.inWheelDown,
                ms.detect_HL, ms.detect_CR, ms.detect_LC);

                        //（出力したいデータをテキストとして出力する処理をここに記述）

        sdPrintf("¥r¥n<END>¥r¥n");

        //---------------------------------------------
        //各時刻毎のバイナリデータ及び 2D 画像データを SD Card に出力
        //---------------------------------------------
        writeLogData();

        //---------------------------------------------
        fclose(fp);
        led_out(0);
        led_rgb.setMode(LED_MODE_SDEND);
        while(1) always_process();                          //動作停止
}
```

4　カメラとサーボの同期について

カメラの信号は約 16.67ms 毎に取得している。サンプルプログラムでは 16ms 毎に自動でサーボへのパルスが発生するように設定されており，この周期が非同期のままだと，最悪のタイミングの場合で，カメラに光が到達してサーボが反応するまで約 32ms かかる。カメラマシンの最大のネックがここにある。スピードを出していくとハンドルを切るのが間に合わずにコースアウトしたりしなかったりする現象に悩まされてしまう。

しかし，これを同期させることで，常に 16.67ms で反応することができる。カメラマシンの最大のネックであるカメラの遅さは事実上問題がなくなり，Basic マシンとタメを張る速さを手に入れることができる！Basic マシンは高速にラインを読み取る事ができるものの，ハンドルを切るタイミングは結局サーボ PWM 周期に拘束されてしまうので，カメラマシンと基本的には同条件になる。Basic マシンに大きく劣る部分としては，タイヤの径の自由度の無さと電池ボックスの抵抗くらいか。

カメラの周期を早めることは困難なので（？），サーボの周期をカメラに合わせて 16.67ms とする。サーボにパルスを送出するタイミングは自動ではなく，カメラの信号を取り込み終えた適切なタイミングでプログラムからパルスを出すようにする。（下記プログラムの※１～４箇所のみの変更）



・プログラム中で TCNT=0;を実行したすぐ後に，MTU2TGRD_0 に設定してあったバッファの値が MTU2TGRB_0 へ転送され，2 カウント後に MTU2TGRD_0 で指定された幅のパルスが P4_0 から出る。

【サーボ PWM の初期設定】

```
//-----------------------------------------------------------------------//
//Initialize MTU2 PWM functions
//-----------------------------------------------------------------------//
//MTU2_0
//PWM mode 1
//TIOC0A(P4_0) :Servo-motor
//-----------------------------------------------------------------------//
void init_MTU2_PWM_Servo( void )
{
       //Port setting for S/W I/O Control
       //alternative mode

       //MTU2_0 (P4_0)
       GPIOPBDC4     = 0x0000;            //Bidirection mode disabled
       GPIOPFCAE4    &= 0xfffe;           //The alternative function of a pin
       GPIOPFCE4     &= 0xfffe;           //The alternative function of a pin
       GPIOPFC4      |= 0x0001;           //The alternative function of a pin
                                          //2nd alternative function/output
```

```
        GPIOP4          &= 0xfffe;              //
        GPIOPM4         &= 0xfffe;              //p4_0:output
        GPIOPMC4        |= 0x0001;              //P4_0:double

        //Module stop 33(MTU2) canceling
        CPGSTBCR3       &= 0xf7;

        //MTU2_0 (Motor PWM)
        MTU2TCR_0       = 0x02;    //0x22;     //TCNT No Clear, P0fÓ/16                ※1
        MTU2TIORH_0     = 0x52;                 //TGRA L>H, TGRB H>L
        MTU2TMDR_0      = 0x32;                 //TGRC and TGRD = Buff-mode
                                                //PWM-mode1
        MTU2TBTM_0      = 0x03;                 //バッファ動作転送は TCNT クリア時
        MTU2TCNT_0      = 0;                    //TCNT0 Set 0
        MTU2TGRA_0      = MTU2TGRC_0 = 2;                   //SV パルス立ち上がり       ※2
        MTU2TGRB_0      = MTU2TGRD_0 = ms.servo_center;     //SV パルス立ち下がり       ※3
        MTU2TSTR        |= 0x01;                //TCNT_0 Start, SV 動作開始
}
```

※1　MTU2TCR_0 を 0x02 に設定すると，カウンタは自動でクリアされない。つまり，任意のタイミングでクリアすることができるようになる。後は，intTimer( )の中で，カメラの画像取得が終わり，プログラムのループを一回り回してサーボの角度指定が終わった後に，MTU2TCNT_0 = 0;を実行すると，カウンタがクリアされる。その 2 カウント後にパルスが発生し，ハンドルが指定角度に切れる。

※2　カウンタをクリアして 2 カウント後にパルス立ち上がり

※3　最初にパルス立ち下りをセンター値に合わせておくと，電源投入時にサーボが暴れない。

【ハードウェアタイマによる 1ms 毎の呼び出し関数】

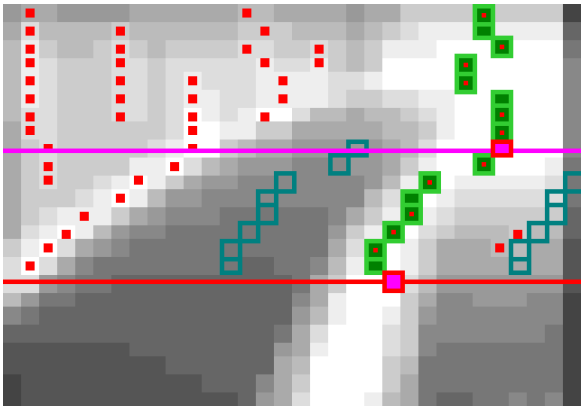
```
//---------------------------------------------------------------------//
//Interrupt function( intTimer )            1ms
//  カメラからの信号は，16.67ms 毎にしか更新されない！
//---------------------------------------------------------------------//
void intTimer( void )
{
        static int      counter = 0;  //Only variable for image process

        ms.cnt0++;      ms.cnt1++;      ms.cntA_int++;
        ms.cntS++;      ms.cntC++;      ms.lap++;

        ms.flagA.bit.timerA = 1;      //1ms 周期で 1 になるフラグを ON
        led_rgb.process();            //LED(rgb) on the GR-peach board

        //field check
        if( vfield_count2 != vfield_count2_buff ) {
                vfield_count2_buff = vfield_count2;
                counter = 0;
        }

        //Top field / bottom field
        switch( counter++ ) {
                //-----------------------------------------------------------
                //カメラからの画素入力 160 x 120
                //-----------------------------------------------------------
                case 0:
                        ImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW,
                                        ImageData_A, vfield_count2 );
                        break;
                case 1:
                        ImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW,
                                        ImageData_A, vfield_count2 );
                        break;
                //-----------------------------------------------------------
                //輝度調整処理？
                //-----------------------------------------------------------
                case 2:
                        Extraction_Brightness( ImageData_A, PIXEL_HW, PIXEL_VW,
                                        ImageData_B, vfield_count2 );
                        break;
                case 3:
                        Extraction_Brightness( ImageData_A, PIXEL_HW, PIXEL_VW,
                                        ImageData_B, vfield_count2 );
                        break;
```

```
            //-------------------------------------------------------
            //画像データを 160 x 120 -> 32 x 24 に縮小する処理
            //-------------------------------------------------------
            case 4:
                ImageReduction( ImageData_B, PIXEL_HW, PIXEL_VW,
                                ImageComp_B, Rate );
                break;
            case 5:
                ImageReduction( ImageData_B, PIXEL_HW, PIXEL_VW,
                                ImageComp_B, Rate );
                break;                                          （ここまでは kit18 同様）

            //-------------------------------------------------------
            //32 x 24 に縮小した画素から中央線検出処理
            //-------------------------------------------------------
            case 6:
                cam.getLine();                  //中央線検出処理
                break;

            //-------------------------------------------------------
            //カメラ画像を取得したところで一呼吸置く
            //ハンドルやモータ出力計算が終わったところで各値を記録するため
            //-------------------------------------------------------
            case 7:
                break;

            //-------------------------------------------------------
            //サーボ PWM 信号発生
            //ハーフラインの検出 type_3             //00112233
            //ログ記録
            //-------------------------------------------------------
            case 8:
                MTU2TCNT_0 = 0;                 //カウンタクリア（→サーボ PWM 信号発生）    ※4

                cam.ex1 = cam.ex2 = 0;          //ハーフライン検出処理 3
                if(cam.detectHalfLine( cam.hlPos)) cam.halfLine |= 0x01;
                if(cam.detectHalfLine(-cam.hlPos)) cam.halfLine |= 0x02;

                sdLog.Rec();                    //ログ記録
                break;

        }
}
```

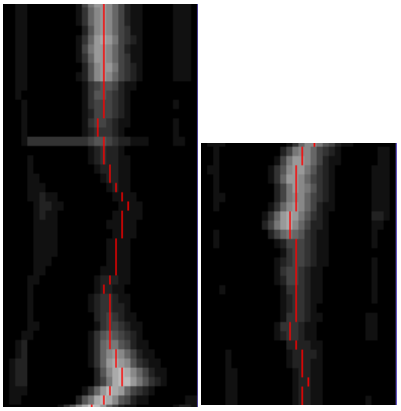
※4　MTU2TCNT_0=0; とすると，カウンタがリセットされる。※2 で MTU2TGRA_0=MTU2TGRC_0=2; と設定しているので，この 2 カウント後にパルスが発生する。

5　濃淡画素データを利用した中央線の検出方法について

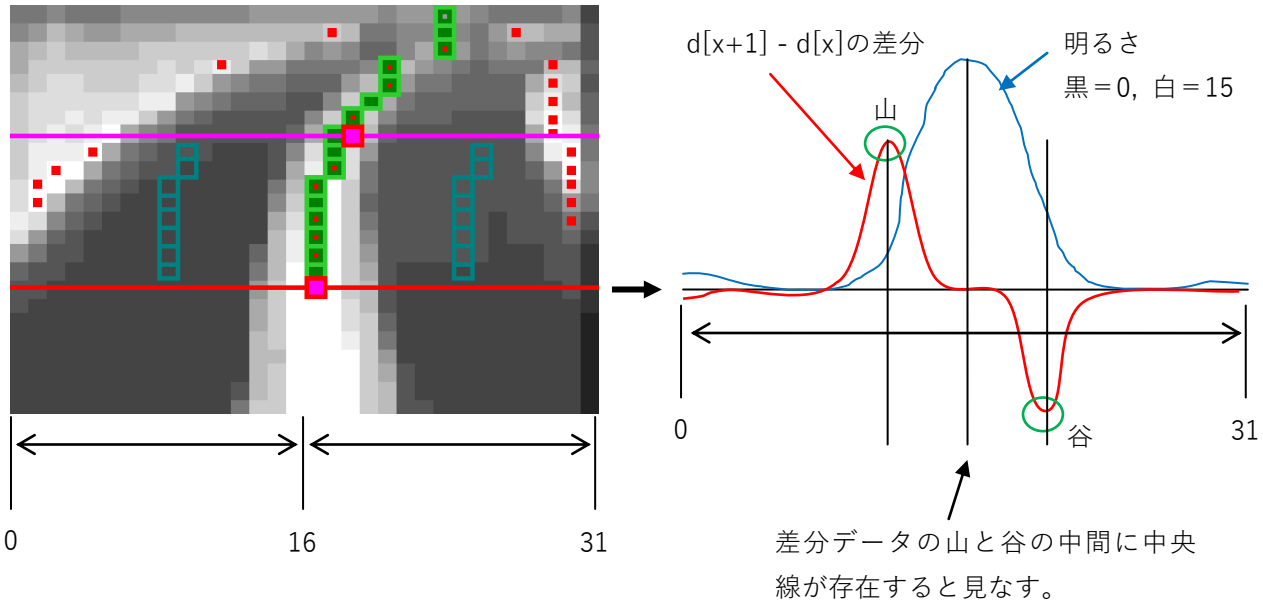
カメラから取得した情報は，32×24 画素のモノクロ濃淡データに変換し，このモノクロ濃淡データから中央線の位置を検出している。この段階で 1 と 0 の 2 値化してしまうと，せっかく見えている白線を見失ってしまう。中央線の位置は 0~31 の値で示され，16 であれば中央からずれていないことになる。



上図は前方に明るい窓がある場面。赤い点は白線として検出した場所を示しており，緑の四角は中央線として検出している場所を示す。かなり真っ白な状況でもある程度白線として認識している。

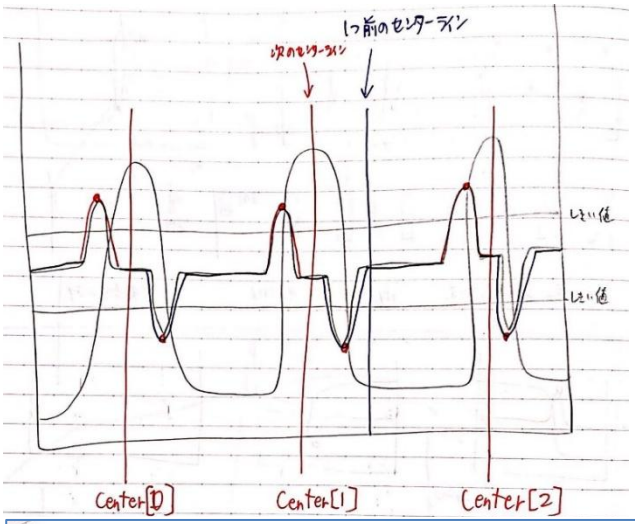


上図は，坂の頂上で線を見失いかけている状況であるが，僅かな濃淡の差から中央線をしっかりと検出できている。



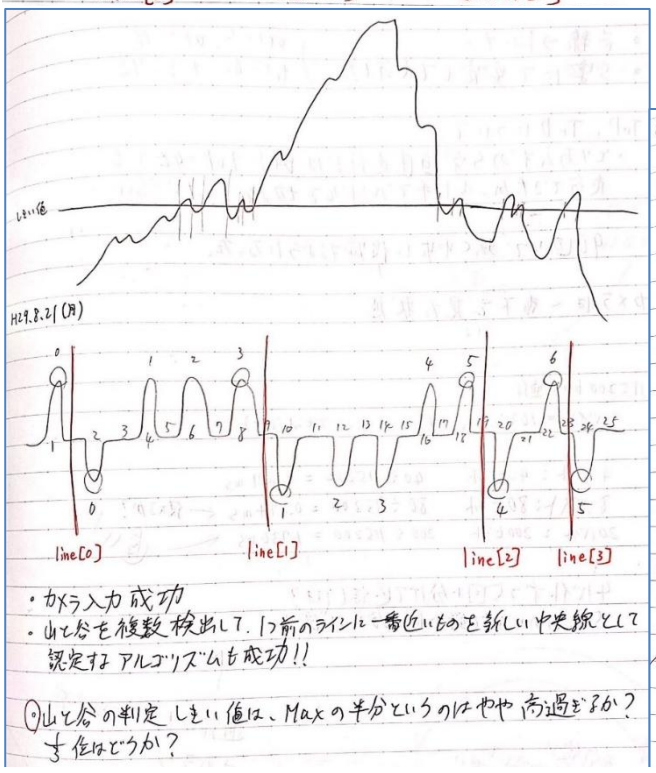
上図において，左の赤い線の部分の明るさ d[x]をグラフに表すと，右の青線のグラフのようになる。そして隣り合う画素同士の差分データ（ d[x+1]－d[x] ）のグラフは，赤線のようになる。左から右に向かって黒から白に変化する部分では赤線のグラフは山となり，白から黒に変化する部分では谷となる。よって白線は差分データの山と谷の中間にあると判断する。その位置は 0~31 で示され，16 で中央，0 に近付くほど左にずれており，31 に近付くほど右にずれていると判断する。ハンドルを切る量は，中心からのずれ量に応じて適切なゲインをかけた値とする。それによって，クランクやレーンチェンジ以外の通常のライントレース処理は非常に簡単な記述で済むようになった。

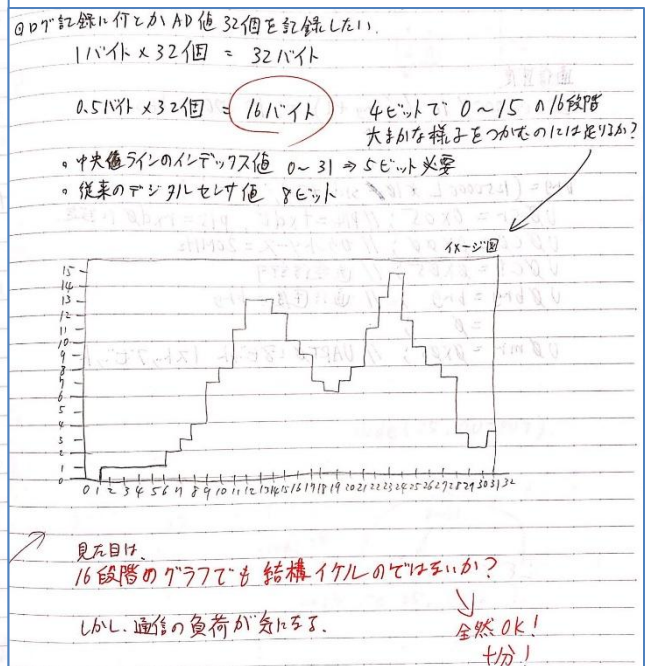
差分データを見ることによって，全体的に暗くなる坂の下や坂の頂上，前方に明るい窓があり全体的に明るくなる場合などでも確実に白線を検出することができる。

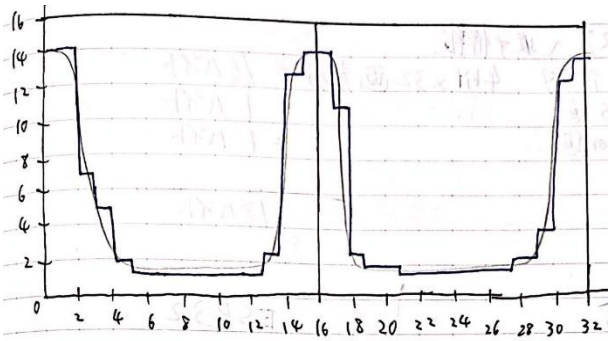


・線として認定するには，山の右側に谷があるものとする。
・山の○つ以内の右側にあるものに限定
・山の右側に次の山が来る前に谷があれば，その中間を線とする。

・山（谷）の認定は，閾値を超えたものの中で一番高い（低い）ものとする。





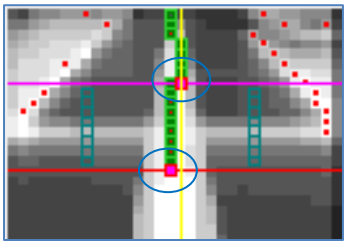


16 段階＝4bit
32 か所で 4×32＝128bit＝16 バイト

5ms 毎に 16 バイト必要
10ms で 16x2=32 バイト
10s で 32000 バイト≒32kB
20s で 64kB

【トレース処理のプログラム】（1ms 毎に呼び出している）

```
void trace( void )
{
	int center;
	switch(ms.trace_mode){
		case 1:			//手前を見る
			center = cam.getCenter();
			break;
		case 2:			//遠くを見る
			center = cam.getCenter2();
			break;
		case 3:			//より遠く１
			center = cam.getFarCenter1();
			break;
		case 4:			//より遠く２
			center = cam.getFarCenter2();
			break;
		default:		//トレース動作なし。ハンドル角固定。
			return;
	}
	ms.center	= center;
	ms.angle	= ms.handleGain * center;
	MTU2TGRD_0	= ms.servo_center + ms.angle * ms.servo_step;
}
```



（1）中央線の検出処理

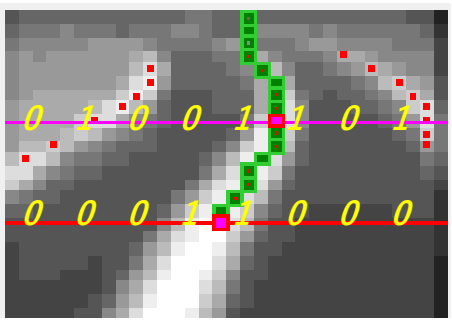
【中央線検出処理のプログラム（抜粋）】　※省略：デジタルセンサ値生成，ハーフライン検出，センター値固定処理 等

```
int			yama_n, tani_n;			//山・谷の数
int			max, min;			//最大値、最小値
int			line_n, line[16];		//ラインの数，Index
struct {						//diff[]の山と谷に関する情報
	int vertex;				//山・谷における頂点
	int index;				//頂点の Index
	int start, end;				//開始 Index、終了 Index
} yama[GASO_N], tani[GASO_N];

void Camera::getLine(void)
{
	unsigned char		data[GASO_N];		//画素データ
	int			diff[GASO_N];		//差分データ
	static int		thDigital = 0;		//白黒の閾値
	int			x;			//halfLine 検出用

	diff_max = halfLine = 0;
	for(int v=vPos; v>=0; v--){			//スキャンする横ラインの縦の位置を手前から奥へ
		//-------------------------------------------------------------------------
		//画素の 1 行スキャン
		//32x24 画素のうち上から v の位置の横 32 画素分をスキャンし，data[]にセット
		//-------------------------------------------------------------------------
		for(int h=0; h<GASO_HW; h++){
			data[h] = ImageComp_B[v * GASO_HW + h];
		}

		//-------------------------------------------------------------------------
		//各画素の差分をとり、diff[]へ格納する。
		//	diff[0]=d[1]-d[0], diff[30]=d[31]-d[30]
		//-------------------------------------------------------------------------
		for(int i=0; i<GASO_N-1; i++){
			diff[i] = (int)data[i+1] - (int)data[i];

			if(v == vPos){
				if(i>=5 && i<=GASO_N-5){
					if(diff[i] >  diff_max) diff_max =  diff[i];
					if(diff[i] < -diff_max) diff_max = -diff[i];
				}
			}
		}

		//-------------------------------------------------------------------------
		//閾値を超える山と谷を全て記録する。 ※diff は[0-30]
		//-------------------------------------------------------------------------
```

```
yama_n = tani_n = 0;
for(int i=0; i<GASO_N; i++){
	yama[i].start = yama[i].end = 0;
	tani[i].start = tani[i].end = 0;
}

if(diff[0] > th_max){                                   //左端の山開始点記録
	yama[yama_n].start = 0;
}
for(int i=GL_START+1; i<GL_END; i++){                   //i は 0-31
	//山の開始点と終了地点、数を記録する
	if(diff[i] > th_max){
		if(diff[i-1] <= th_max){
			yama[yama_n].start = i;
		}
	}
	else{
		if(diff[i-1] > th_max){
			yama[yama_n].end = i;
			yama_n++;
		}
	}

	//谷の開始点と終了地点、数を記録する
	if(diff[i] < th_min){
		if(diff[i-1] >= th_min){
			tani[tani_n].start = i;
		}
	}
	else{
		if(diff[i-1] < th_min){
			tani[tani_n].end = i;
			tani_n++;
		}
	}
}
//右端の谷終了点記録
if(tani[tani_n].start != 0 && tani[tani_n].end == 0){
	tani[tani_n].end = GL_END-1;
	tani_n++;
}

//それぞれの山の頂点を記録する。
for(int j=0; j<yama_n; j++){
	yama[j].vertex = diff[yama[j].start];
	yama[j].index  = yama[j].start;
	for(int i=yama[j].start + 1; i<=yama[j].end; i++){
		if(diff[i] > yama[j].vertex){
			yama[j].vertex = diff[i];
			yama[j].index  = i;
		}
	}
}

//それぞれの谷の頂点を記録する。
for(int j=0; j<tani_n; j++){
	tani[j].vertex = diff[tani[j].start];
	tani[j].index  = tani[j].start;
	for(int i=tani[j].start + 1; i<=tani[j].end; i++){
		if(diff[i] < tani[j].vertex){
			tani[j].vertex = diff[i];
			tani[j].index  = i;
		}
	}
}

//山の右側に谷があったら、その中間を線とみなす。
line_n = 0;
for(int i=0, j=0; i<yama_n && j<tani_n; i++){

	//山の左側の谷を読み飛ばす
	while(tani[j].index <= yama[i].index){
		j++;
		if(j >= tani_n){             //もうこれ以上谷がない
			goto line_comp_detect;
		}
```

```
		}

		//谷の左側に後続の山が続いている場合
		if(i < yama_n - 1){
			if(yama[i+1].index < tani[j].index){
				continue;
			}
		}

		//CR,LC 以外の時に・・・
		if( !((ms.mode >= 60 && ms.mode < 70) || (ms.mode >= 90 && ms.mode < 100)) ){
			//左側の山と右側の谷の間隔が開き過ぎていたら、線以外と判断し、次の山へ
			if( tani[j].index - yama[i].index > 16){	//12
				continue;
			}
		}

		//山のすぐ右側に谷がある場合、そこに線があると認定 line[]に Index 追加
		max = (int)data[yama[i].index];
		int line_left = yama[i].index;			//頂上の中央位置算出
		for(int k=yama[i].index+1; k<=tani[j].index; k++){
			if((int)data[k] > max){
				max = (int)data[k];
				line[line_n] = line_left = k;
			}
			else if((int)data[k] == max){
				line[line_n] = (k + line_left) / 2;
			}
		}
		line_n++;

	}
	line_comp_detect:

	//もし、ラインを一つも検出できなかった場合は、一つ前の値を line[0]とする。
	if(line_n == 0){
		if(v == vPos)				//一番下の場合は一つ前
			line[0] = CenterIndex[v];
		else					//それ以外は一つ下
			line[0] = CenterIndex[v+1];
	}

	//中央線を検出する
	//line[] の中で、一つ下の中央線に一番近いものを次の中央線とする
	int df, df_index;
	if(v == vPos){				//一番下の場合は一番明るいところ
		max = 0;
		df_index = CenterIndex[v];
		for(int i=0; i<line_n; i++){
			df = line[i] - CenterIndex[v];
			if(df > -CenterDiff && df < CenterDiff){
				if(data[line[i]] > max){
					max = data[line[i]];
					df_index = line[i];
				}
			}
		}
		CenterIndex[v] = df_index;	//Index を更新する
	}
	else{					//それ以外は一つ下との差分
		df = line[0] - CenterIndex[v+1];
		if(df < 0) df = -df;
		df_index = line[0];
		min = df;

		for(int i=1; i<line_n; i++){
			df = line[i] - CenterIndex[v+1];
			if(df < 0) df = -df;
			if(df < min){
				min = df;
				df_index = line[i];
			}
		}
		//一つ下との差が大きいものは却下する
		df = df_index - CenterIndex[v+1];
		if(df > -CenterDiff2 && df < CenterDiff2)
```

```
                        CenterIndex[v] = df_index;   //Index を更新する
                    else
                        CenterIndex[v] = CenterIndex[v+1];
                }
        }
}
```

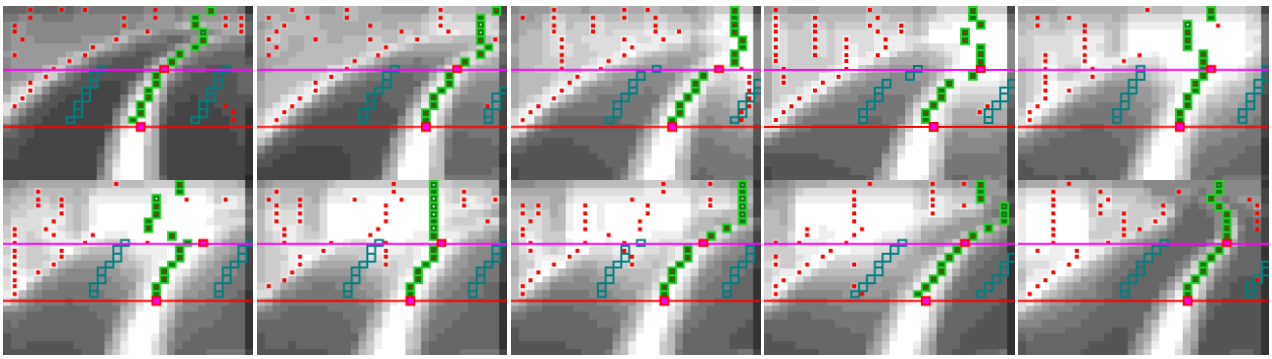


手前（赤線）と奥（紫線）の 2 カ所を 8bit のデジタル化し，クランクやレーンチェンジ等で使用している。

← 01001101

← 00011000

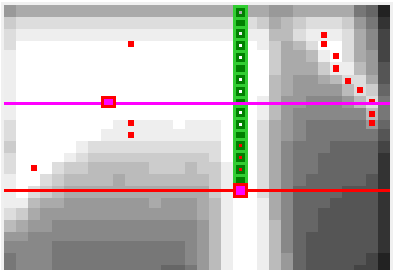
○中央線検出は，手前から奥にかけて 1 ラインずつ検出することで，外乱にも強くなる。手前（下側）の画像は距離や角度的にも条件が良く，ハッキリと白線が見えるので，読み違えることは滅多にない。1 つ奥（上）のラインで検出した白線の中から，1 つ下の中央線に一番近いものを中央線と見なす。近くに白線を検出できなかった場合は，1 つ下のラインと同じ場所を中央線の位置とする。



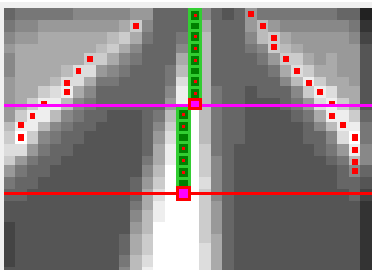
上図は，前方に明るい窓がある場所の走行動画。奥（上）の方が真っ白になっていても，手前（下）の方はハッキリと中央線を検出できているので，奥の方が外乱に乱されても，悪条件の場所をしのぐことができれば，手前から伸びた正しい中央線の位置が復活する。

6　外乱光の影響について

前方に明るい窓がある場合



外乱光がある場合



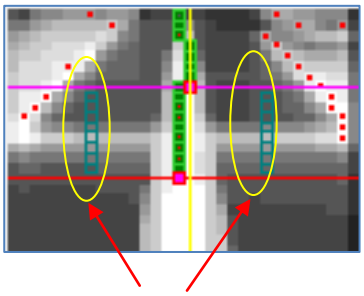
外乱光が無い場合

左の写真のように，前方に明るい窓などがあると，カメラに映るコースは想像以上に真っ白になる。上記のアルゴリズムによる通常のトレースでは，あまり問題ないが，ハーフライン読み取りにまだ課題が残っている。外乱光により左ハーフラインと誤検出することが無いよう，黒→白→黒の間隔が狭いときのみ白線と認定しないようにした。

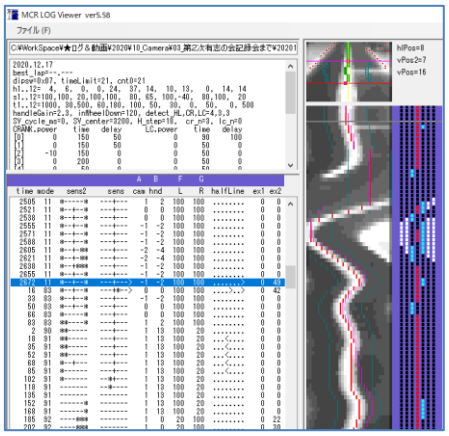
7　ハーフラインの読み取りについて

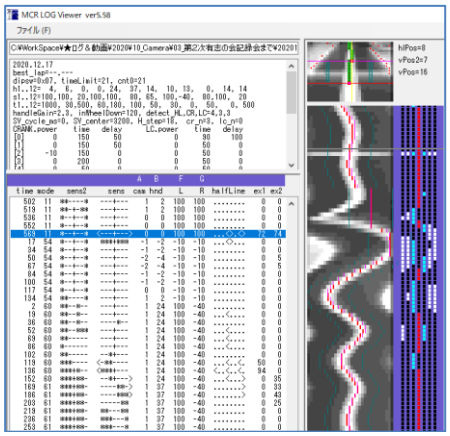
最初は，Basic マシンなどと同じように，2 次元画像内のある場所を横一直線に読み，全部白だったらクロスラインと判定するなどのアルゴリズムにしていたが，これだと線が細い場合に読み飛ばすことが頻発した。この方法はせっかくの 2 次元画像という情報の山をほとんど捨てているようなものである。

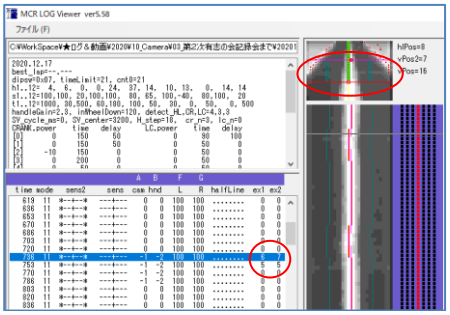
そこで下図左のように，中央線から一定距離離れた場所を縦に読んで白線の有無を判別するようにしたことで，ほぼ 100%ハーフラインを検出することができるようになった。白線の判別アルゴリズムは中央線のアルゴリズムと同じ。上下の画素の差分情報を読み，黒から白へ変化する山と白から黒へ変化する谷が一定距離以内であれば白線と判断する。これが離れすぎている場合は外乱光の可能性を疑う。



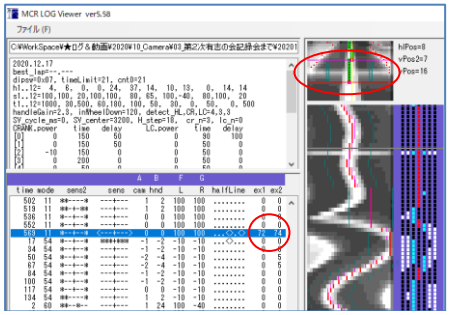
青い四角の場所を縦に読んで白線の有無を判別







コースの継ぎ目
明暗差は 5~7 辺り

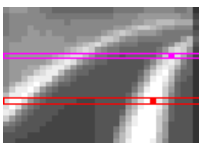


クロスライン
明暗差は 72～74 辺り

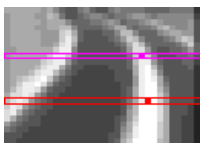
コースの継ぎ目でも白線と検出されてしまうが，黒と白の明暗差が 40 以上でないとクロスラインと認定しないようにするとよい。

8　スラロームの制御

右の画像 A と B では，双方とも中心より右側にあるが，先の状況がまるで違う。A は前方（紫）のセンター値が手前（赤）よりも外側にあり，右コーナーの最中である。この状況ではハンドルを右に切って，左右駆動輪の内輪差も右に曲がりやすいよう，右側を左側よりも回転数を落としている。



A



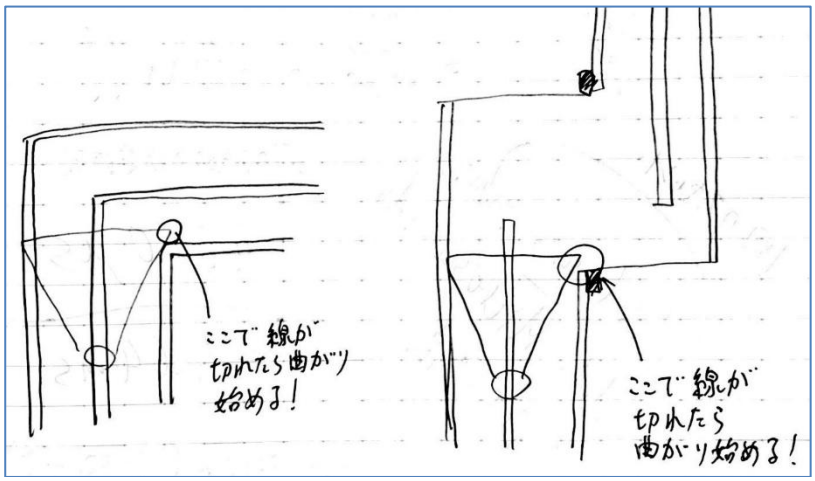
B

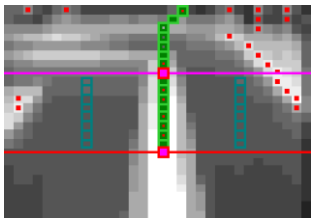
しかし B は，前方（紫）のセンター値は手前（赤）よりも内側にあり，画像からも右から左への切り返しの場面であることが分かる。この場合，例えば駆動輪を左側の回転数を右側より低くすることで車体を早めに左に曲がりやすい状態にしておくと，左右の切り返しでの挙動の乱れを抑えることができる。

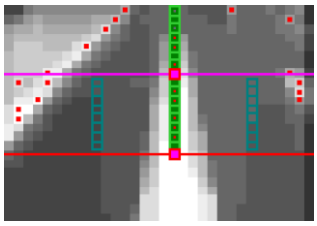
9　クランク及びレーンチェンジにおける曲げ開始の判断について

画像を解析することで，クランクやレーンチェンジにおける両サイドの外側の線もはっきりと見えていることが分かった。そして，中央線よりも早いタイミングで片方の線が途切れていることを確実に検出できていることも分かった。

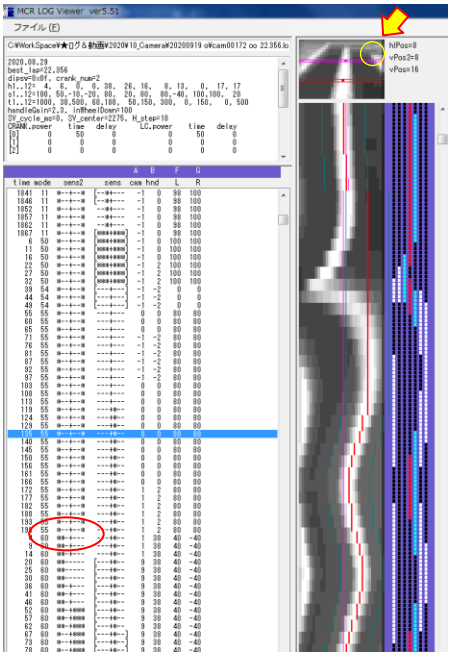
そこで，ハンドルを切り始めるタイミングを，外側の線の途切れによって開始することで，より早いタイミングでハンドルを切り始めることができる。結果的に理想的なラインでしかも速いスピードで安定してクリアできるようになった。



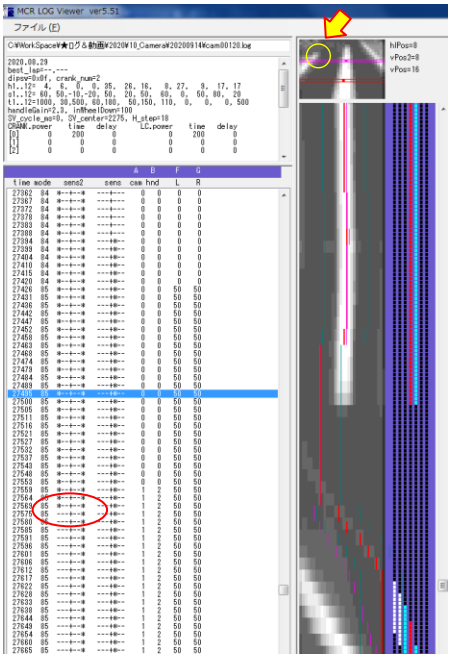




クランク



レーンチェンジ



10　スタートバーの検出について

当初，ある場所の横一直線が全部白ならスタートバーありと判定していたが，意外とスタートバーは白く見えてなかった。特に前方の窓が明るいと暗く見え，うまく検出できなかった。そこで，横一直線の明暗差がくっきりあればスタートバーなし，明暗差がなければスタートバーありと判定するとうまくいった。

11　e2 studio におけるファイル分割について

Debug フォルダ内の subdir.mk ファイルを編集する。ファイル中には「編集しないでください」と書いてあるが，太字の部分を編集すると，分割したファイルもビルド対象となる。プログラムが大きくなると，機能毎に分割した方が把握しやすくなる。

```
################################################################################
# 自動生成ファイルです。 編集しないでください!
################################################################################

# これらのツール呼び出しからの入力および出力をビルド変数へ追加します
CPP_SRCS += ¥
       ../001_setup.cpp ../010_always_process.cpp ¥
       ../100_main.cpp ../110_normal_run.cpp ¥
       ../150_cr_search.cpp ../155_cr_clear.cpp ¥
       ../160_lc_search.cpp ../165_lc_clear.cpp ¥
       ../200_hw.cpp ../201_hw_motor.cpp ../202_hw_camera.cpp ../211_log.cpp ¥
       ../301_camera.cpp ¥
       ../401_pcmenu.cpp ¥

OBJS += ¥
       ./001_setup.o ./010_always_process.o ¥
       ./100_main.o ./110_normal_run.o ¥
       ./150_cr_search.o ./155_cr_clear.o ¥
       ./160_lc_search.o ./165_lc_clear.o ¥
       ./200_hw.o  ./201_hw_motor.o ./202_hw_camera.o ./211_log.o ¥
       ./301_camera.o ¥
       ./401_pcmenu.o ¥

CPP_DEPS += ¥
       ./001_setup.d ./010_always_process.d ¥
       ./100_main.d ./110_normal_run.d ¥
       ./150_cr_search.d ./155_cr_clear.d ¥
       ./160_lc_search.d ./165_lc_clear.d ¥
       ./200_hw.d  ./201_hw_motor.d ./202_hw_camera.d ./211_log.d ¥
       ./301_camera.d ¥
       ./401_pcmenu.d ¥

# サブディレクトリーはすべて，それ自身がコントリビュートするソースをビルドするためのルールを提供しなければなりません
%.o: ../%.cpp
       @echo 'Build Target : $<'
       @echo 'Calling : Cross ARM C++ Compiler'
       arm-none-eabi-g++ -mcpu=cortex-a9 -march=armv7-a -marm -mthumb-interwork -mfloat-abi=hard -mfpu=vfpv3
-mno-unaligned-access -O2 -fmessage-length=0 -ffunction-sections -fdata-sections -fno-common -Wall -Wextra  -g
-DDEVICE_ERROR_PATTERN=1 -DARM_MATH_CA9 -DFEATURE_LWIP=1 -D__FPU_PRESENT -D__MBED__=1 -DDEVICE_I2CSLAVE=1

      :
      :
  （以下省略）
```

12　感想

1 年間の研究を通して，Camera クラスマイコンカーの安定化及び高速化は，ある程度満足できるレベルに達することができたと感じている。「ロボット開発において，内部状態の把握は最重要課題の一つである」という以前からの信念を改めて再確認した。そして，2 次元画像は，今までのラインセンサからすると「未来」の情報まで詰め込まれた貴重な情報の宝庫であり，適切に解析することで先々の動きを先手を打って制御できるという，これまでにない制御方法の可能性に溢れたものである。うまくいけば Basic クラスよりも速いマイコンカーができるのではないかと感じている。