ISC XC SDK V2
ライブラリマニュアル

【ご注意】

1．本マニュアルの内容の一部または全部を無断転載することは禁止されています
2．本マニュアルの内容に関しては将来予告なしに変更することがあります
3．本マニュアルの内容について万全を期して作成しております、万一ご不審な点や誤り、記載漏れなどお気づきのことがございましたら、ご連絡ください
4．運用した結果の影響に関しては、3．項にかかわらず責任を負いかねますのでご了承ください

目次

１．はじめに

本文は、ISC SDK バージョン２（以下　ISC SDK V2）について説明します。

ISC SDK XC V2 は、ISC-100XC/PP2(FPGA バージョン 20)以降のカメラに対応する SDK です。バージョン 20 以降のカメラを使用する場合は、本 SDK をお使いください。

2．動作環境

【サポートするカメラ】

本体	ISC-100XC/PP2
FPGA バージョン	20（HEX）以降

【動作確認済み環境】

プラットフォーム	OS
x86　PC　※1	Widnows10　64bit
	Ubuntu 18.04 LTS/16.04 LTS　※2
Jetson Xavier	Jetpack 4.3　※2
Jetson TX2	Jetpack 4.2/3.2　※2

※1　Intel® Core™ i7-7700 CPU @3.6GHz　16GB RAM

※2　動作確認情報については、当社で用意したテスト環境における動作結果であり、お客様のご利用環境での Linux の動作を保証するものではありません。

【Build 環境】

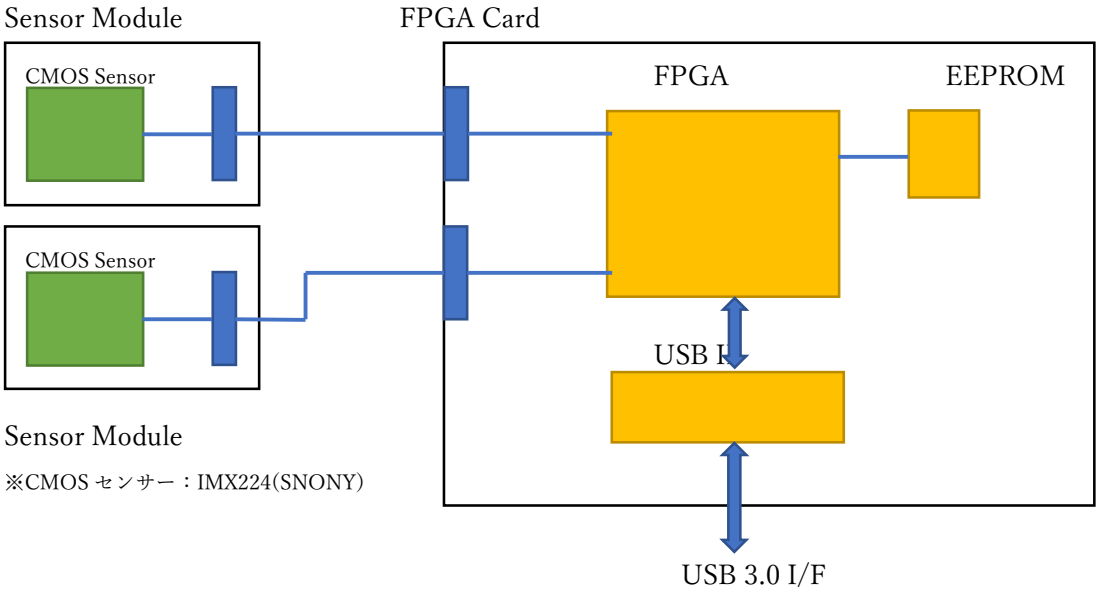
OS	コンパイラ
Widnows10　64bit	Visual Studio 2017 プラットフォームツールセット　v141 SDK バージョン　10.0.17134.0
Ubuntu	GCC 7.3

【使用言語】

OS	言語
Widnows10	C/C++
Ubuntu	C/C++

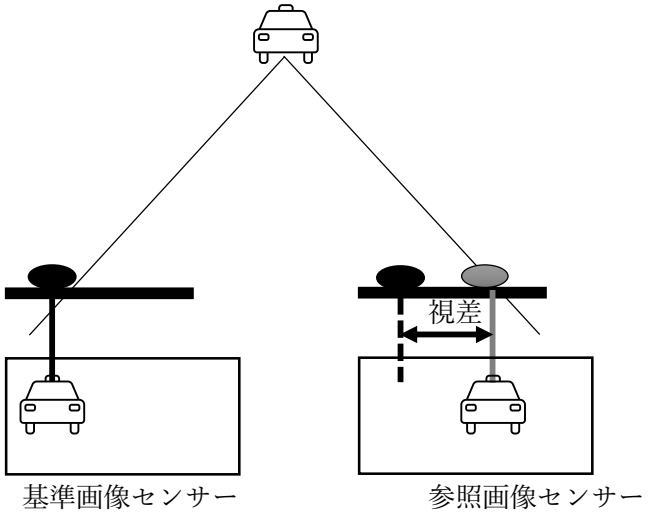
３．ステレオカメラの構成

下図にカメラの簡略なブロック図を示します。
カメラはセンサーを搭載した Sensor Module と、ステレオ処理及び USB インターフェースを行う FPGA Card によって構成されています。



３．１　基準画像カメラと参照画像カメラ

ステレオカメラで対象物までの距離を求めるには，下図に示すように三角測量の原理を利用して算出しており、同じ対象物を 左右のカメラで撮像した際の撮像位置の差分（視差）を求めています。



3．2　画像データフォーマット

（1）画像有効サイズ　　1280（幅）×　720（高）　画素

（2）視差有効サイズ　　1024（幅）×　720（高）　画素

（3）画像データフォーマット　（モノクロ画像）

風景

受信データ

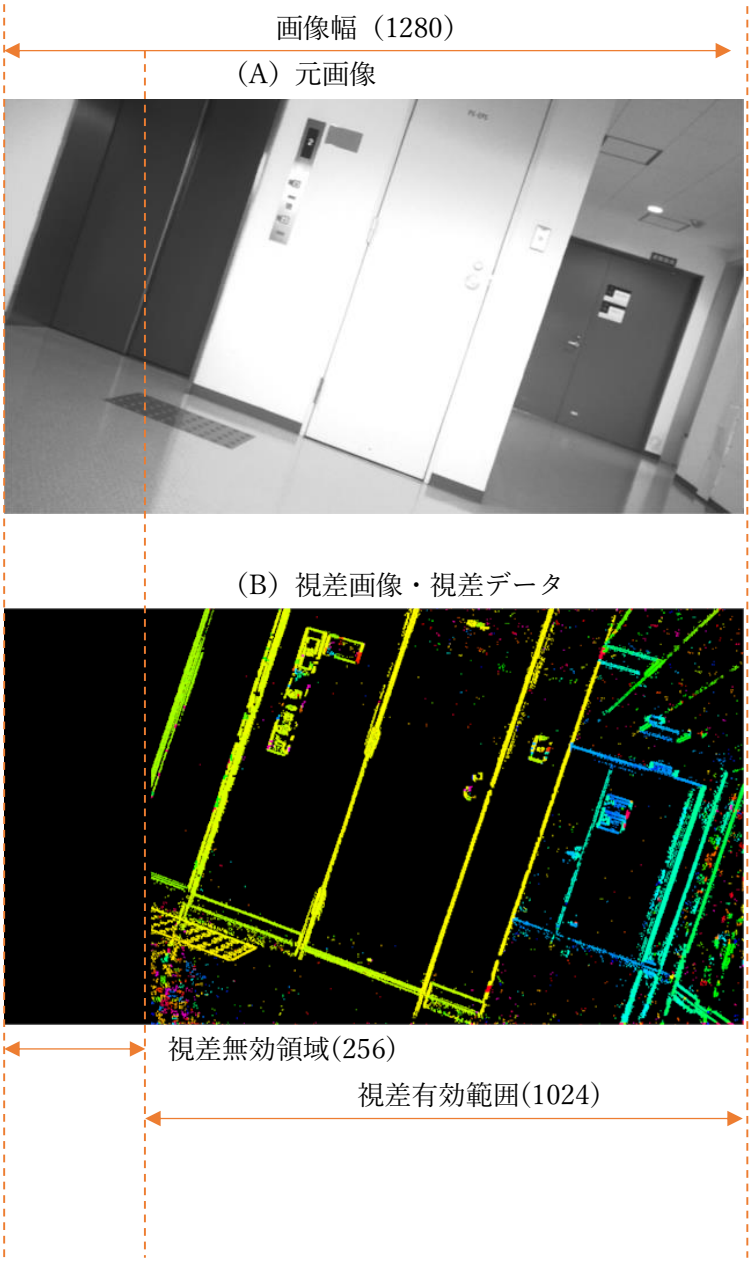
受信データは、画像の左下画素より格納されています。

画像バッファー（1280x720）

0	1				1278	1279
920319						921599

（５）視差の無効領域

下図のように視差データは無効な領域を含みます。



３．３　EEPROM（不揮発性メモリ）の機能

EEPROM には、下記のデータを保存しています。
ユーザーは、API を使用して一部のデータを取得できますが、直接アクセスすることはできません。

データ名称	備考
補正校正テーブル	
システム設定データ	一部のデータは API（GetCameraParamInfo）を使用して取得できます
Auto Calibration データ	動作パラメータ及びエラーステータスが保存されます

４．露光調整機能

４．１　概要

カメラのシャッター制御を行います。
シャッター制御はマニュアルモード及びシングルシャッターモードとダブルシャッターモードを持ちます。
入力された補正画像の指定されたエリアの輝度ヒストグラムを作成し Exposure （シャッターを開いている時間 = 蓄積時間)を制御し、CMOS センサーに入射する光量を最適化すると共に、CMOS センサーのアンプの Gain(利得)を制御することにより、最適な輝度出力が得られるようにします。

４．２　露光調整測定エリアの指定

シャッターの値を制御するときには、イメージ全体の輝度分を使うわけではなく、指定されたエリアの輝度分布を使用します。
エリアの指定は以下の 2 種類が準備されています。

① 上下左右の 4 か所の座標を指定して長方形のエリアを指定します
② 上下の座標と、上辺の左右、下辺の左右の 6 か所の座標を指定して四角形又は三角形のエリアを指定します

詳細は、API(SetMeasArea 及び SetMeasAreaEx)を参照してください。

４．３　シングルシャッターモード（SampleViewer 設定名：Normal On）

1 フレームの入力画像を解析し Exposure と Gain を決定する為、早い応答速度で最適な露光調整値を算出します。
４．4 章のダブルシャッターモードと比較し、1/3～同等（数 frame～数 10frame）の応答速度となります。

４．４　ダブルシャッターモード

ダイナミックレンジの広い入力画像に対して、視差情報の欠損を最小限に抑えるよう
シャッターを制御する機能です。
一般的な HDR では長いシャッターと短いシャッターの画像を合成し 1 フレームを生成する為、合成後の画像では輝度情報が圧縮されてしまいます。
輝度情報から視差情報を生成するステレオカメラにとって、輝度情報が圧縮される事は視差情報が減少する事に繋がり、HDR の効果を最大限得ることができません。
この問題を解消する機能がダブルシャッターとなります。

ダブルシャッターモードでは 1 フレーム毎に長いシャッターと短いシャッターの画像が交互に入力されるようにシャッターの値を制御します。
視差情報の算出をそれぞれのフレームで行う事で視差情報の欠損を抑えています。

４．４．１　画像合成有り（SampleViewer 設定名：Double On（Image Fusion））

カメラから交互に出力される長いシャッターと短いシャッターの画像を SDK で合成し表示するモードです。
ダブルシャッターによるシャッター制御を視覚的に理解する為の表示モードです。
またこの画像を使い後段アプリを設計する事も可能です。

４．４．２　画像合成無し（SampleViewer 設定名：Double On）

カメラから交互に出力される長いシャッターと短いシャッターの画像をそのまま表示するモードです。従って、ダイナミックレンジの広いシーンではフレーム毎に明暗の点滅が起きます。
SDK で画像合成を行わない為、マイコンなど限られたリソース環境での開発に適しています。

４．５　マニュアルモード（SampleViewer 設定名：Off）

シャッター制御をユーザーの指定した値とします。
API（SetExposureValue/SetGainValue）を使用して値を指定します。

５．自動調整機能（Auto Calibration）

経年劣化や熱膨張等でカメラが変形した事による視差密度低下を FPGA で検出・改善する機能です。
自動調整には以下２モードがあります。

- 自動調整：カメラ内部で視差密度低下を検出し改善します。
- 強制調整：アプリケーションからのトリガにより視差密度の改善を行います。

6．関数一覧

関数名	概要	備考
OpenISC	ライブラリのオープン	
CloseISC	ライブラリのクローズ	
StartGrab	画像取り込み開始	
StopGrab	画像取り込み停止	
GetImage	画像取得	
GetDepthInfo	視差データ取得	
SetRGBEnabled	カラーモード　On/Off	
GetYUVImage	YUV データ取得	
ConvertYUVToRGB	RGB へ変換	
ApplyAutoWhiteBalance	自動ホワイトバランス処理	
CorrectRGBImage	カラー画像の補正処理	
GetCameraParamInfo	カメラ固有パラメータ取得	
GetImageSize	画像サイズの取得	
SetAutoCalibration	自動調整モード指定（自動・強制）	
GetAutoCalibration	自動調整モード取得（自動・強制）	
SetShutterControlMode	露光調整モード指定	
GetShutterControlMode	露光調整モード取得	
SetExposureValue	露光設定値指定	
GetExposureValue	露光設定値取得	
SetGainValue	ゲイン設定値指定	
GetGainValue	ゲイン設定値取得	
SetNoiseFilter	ノイズフィルター値指定	
GetNoiseFilter	ノイズフィルター値取得	
SetMeasArea	露光調整　測定エリア指定	4 点指定
GetMeasArea	露光調整　測定エリア取得	4 点指定
SetShutterControlModeEx	露光調整モード指定	ダブルシャッター指定可能
GetShutterControlModeEx	露光調整モード取得	ダブルシャッター指定可能
SetMeasAreaEx	露光調整　測定エリア指定	4／6 点指定
GetMeasAreaEx	露光調整　測定エリア取得	4／6 点指定

７．関数説明

実装している関数について説明します。

注１
API は原則として Windows 版と Linux 版で共通ですが、いくつかの違いがあります。
以降は Windows 版について説明します。
Windows 版と Linux 版の違いについては、１４．Linux 版を参照してください。

OpenISC

C++/C	int OpenISC()

Parameter		
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	ライブラリの使用を開始します

	int ret = OpenISC();
	if(ret !=0){
	　// Error 処理
	}

CloseISC

C++/C	int CloseISC ()

Parameter		
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	ライブラリの使用を終了します

	int ret = CloseISC ();
	if(ret !=0){
	　// Error 処理
	}

StartGrab

C++/C	int StartGrab(int nMode);

Parameter	nMode	取り込みモードを指定します 2：視差モード 3：補正後画像モード 4：補正前画像モード
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	画像の取り込みを開始します

	int nMode=2;
	int ret = StartGrab(nMode);
	if(ret !=0){
	　// Error 処理
	}

StopGrab

C++/C	int StopGrab();

Parameter		
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	画像取り込みを停止します

	int ret = StopGrab ();
	if(ret !=0){
	　// Error 処理
	}

GetImage

C++/C	int GetImage(unsigned char* pBuffer1, unsigned char* pBuffer2,int nSkip)

Parameter	pBuffer1 pBuffer2	※StartGrab で指定する取り込みモード及び SetShutterControlMode(Ex)で指定するモードによって内容が変わります
	nSkip	未使用　0 を指定する
Return Value	成功時　0 失敗時　!=0	
Remarks	GetImage とカラー画像取得 API（GetYUVImage）は、同時に呼び出されないないように同一 Thread で呼び出すか、排他制御を行ってください	
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	1 Frame の画像を取得する

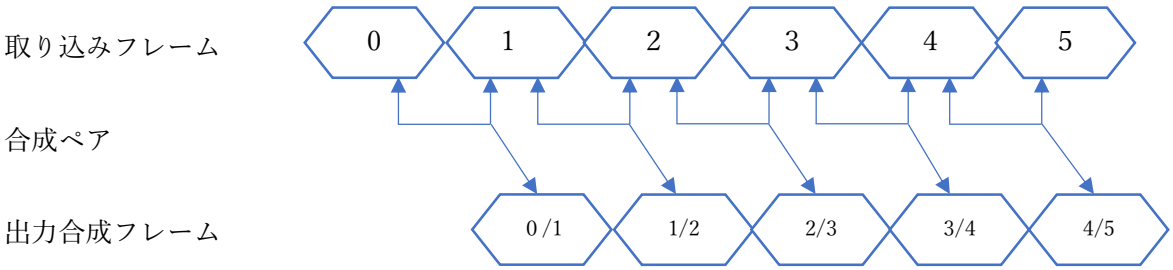
	unsigned char* pBuffer1 = new unsigned char[width * height];
	unsigned char* pBuffer2 = new unsigned char[width * height];
	Int ret = GetImage(pBuffer1, pBuffer2, 0);
	If(ret= 0){
	　// Error
	}
	// データ使用
	・・・
	delete[] pBuffer1;
	delete[]pBuffer2;

【データの内容】

		pBuffer1	pBuffer2
露光調整モード	取り込みモード		
マニュアル・ シングルシャッター	視差画像モード	視差画像	基準センサー補正後画像
	補正後画像モード	比較センサー補正後画像	基準センサー補正後画像
	補正前画像モード	比較センサー補正前画像	基準センサー補正前画像
ダブルシャッター （画像合成あり）	視差画像モード	合成視差画像	合成基準センサー補正後画像
	補正後画像モード	比較センサー補正後画像	基準センサー補正後画像
	補正前画像モード	比較センサー補正前画像	基準センサー補正前画像
ダブルシャッター （画像合成なし）	視差画像モード	視差画像	基準センサー補正後画像
	補正後画像モード	比較センサー補正後画像	基準センサー補正後画像
	補正前画像モード	比較センサー補正前画像	基準センサー補正前画像

※補正画像：レンズの歪みを補正した画像です

【合成のデータについて】
合成は、常に手前のフレームと行います。



【エラー処理】
Return value の標準的な処理例を以下に示します。

```
int ret = GetImage(xxxx):

if(ret != 0){
  // ERROR

  If(ret == ERR_USB_NO_IMAGE){
    画像の準備ができていないため、新しく表示又は処理するデータがありません
    例えば、カメラの受信周期より速い周期で呼び出した場合に発生します
  }
  else if(ret == FT_IO_ERROR){
    USB の接続エラーです
    例えば、ケーブルが外れたケースなどです
    アプリケーションに合わせて適切に処理してください
  }
  else if(ret == ERR_NO_VALID_IMAGES_CALIBRATING){
    自動調整中のため表示又は処理するためのデータがありません
  }
  else {
    画像に起因するエラーが多いため、確認のために表示することを推奨します
    画像を確認し適切に処理してください
  }
}
```

GetDepthInfo

C++/C	int GetDepthInfo(float* pBuffer)

Parameter	pBuffer	視差データ
Return Value	成功時　0 失敗時　!=0	
Remarks	先行して GetImage()を呼びだす必要があります 視差データを距離(m)に変換するための方法は、Appendix A 視差の使い方　を参照してください	
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	1 Frame の視差情報を取得します

	float* pBuffer = new float[width * height];
	int ret = GetDepthInfo(pBuffer);
	if(ret != 0){
	　// Error
	}
	//・・・データ使用
	delete[] pBuffer;

【視差データフォーマット】
視差値（浮動小数点）が画像データと同じ順番で入ります。

SetRGBEnabled

C++/C	int SetRGBEnabled(int nMode)

Parameter	nMode	カラーモードの On/Off を指定します 0 ：Off 1 ：On
Return Value	成功時　0 失敗時　!=0	
Remarks	カラーモードを On にするためには、毎回 StartGrab()の呼び出し前に設定が必要です ダブルシャッター機能が On の場合には、カラーモードは On にできません	
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	カラーモードを On に設定します

	int mode=1;
	int ret = SetRGBEnabled (mode);
	if(ret != 0){
	　// Error
	}
	//取り込み開始
	ret = StartGrab(2);

GetYUVImage

C++/C	int GetYUVImage(unsigned char* pBuffer, int nSkip)

Parameter	pBuffer	YUV422 フォーマットのカラーデータです バッファーのサイズは（幅×高）×２必要です
	nSkip	未使用です 0 を指定してください
Return Value	成功時　0 失敗時　!=0	
Remarks	GetYUVImage と画像取得 API（GetImage）は、同時に呼び出されないないように同一 Thread で呼び出すか、排他制御を行ってください	
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	YUV データを取得します

	unsigned char* yuvBuffer = new unsigned char[width* height * 2];
	int ret = GetYUVImage (yuvBuffer, 0);
	if(ret != 0){
	　// Error
	}
	//YUV データ処理
	・・・・・・
	delete[] yuvBuffer;

「フォーマット」

Y0	U	Y1	V	Y2	- - - - -	Y1279	V

受信データは、画像の左下画素より格納されています。

風景

受信データ

画像バッファー

0	1				2558	2559
1840638						1843199

ConvertYUVToRGB

C++/C	int ConvertYUVToRGB(unsigned char *yuv, unsigned char *prgbimage, int dwSize)

Parameter	yuv	YUV422 フォーマットのカラーデータです
	prgbimage	変換した RGB（24bit）データです 格納順序は、BGR となっています
	dwSize	prgbimage バッファーのサイズ(バイト)です
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	RGB データを取得します

	unsigned char* rgbBuffer = new unsigned char[width* height * 3];
	int ret = ConvertYUVToRGB (yuvBuffer, rgbBuffer, width* height * 3);
	if(ret != 0){
	　// Error
	}
	//RGB データ処理
	・・・・・・
	delete[]rgbBuffer;

「変換処理」

R=y+(1.402*(v-128))

G=y-(0.71414*(v-128))-(0.34414*(u-128))

B=y+(1.722*(u-128))

ApplyAutoWhiteBalance

C++/C	int ApplyAutoWhiteBalance(unsigned char *prgbimage, unsigned char *prgbimageF)

Parameter	prgbimage	入力 RGB（24bit）データです
	prgbimageF	調整後 RGB（24bit）データです
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	ホワイトバラン調整を行います

	unsigned char* awBuffer = new unsigned char[width* height * 3];
	int ret = ApplyAutoWhiteBalance (rgbBuffer, awBuffer);
	if(ret != 0){
	　// Error
	}
	//RGB データ処理
	・・・・・・
	delete[]awBuffer;

「処理」
(1)原画像の平均値を求めます。
（赤：緑：青）＝（Rave:Gave:Bave）とします

(2)以下のように RGB 成分のゲインを計算します。
KRave = (Rave + Gave + Bave) / (3 * Rave);
KGave = (Rave + Gave + Bave) / (3 * Gave);
KBave = (Rave + Gave + Bave) / (3 * Bave);

(3)RGB 3 チャンネルの値を次のように調整します。
Rvalu = R * KRave;
Gvalu = G * KGave;
Bvalu = B * KBave;

CorrectRGBImage

C++/C	int CorrectRGBImage(unsigned char *prgbimageF, unsigned char *AdjustBuffer)

Parameter	prgbimageF	入力 RGB（24bit）データです
	AdjustBuffer	補正後のデータです
Return Value	成功時　0 失敗時　!=0	
Remarks	補正後の画像は、モノクロ画像とは完全には一致しません	
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	補正を行います

	unsigned char* adjustBuffer = new unsigned char[width* height * 3];
	int ret = CorrectRGBImage (awBuffer, AdjustBuffer);
	if(ret != 0){
	　// Error
	}
	//RGB データ処理
	・・・・・・
	delete[]adjustBuffer

GetCameraParamInfo

C++/C	int GetCameraParamInfo(CameraParamInfo* pParam)

Parameter	pParam	カメラ固有のパラメータ構造体です
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	パラメータを取得します

	CameraParamInfo parameter = {};
	int ret = GetCameraParamInfo(&parameter);
	if(ret != 0){
	　// Error
	}
	// ・・・データ使用

【CameraParamInfo 構造体】

型	変数名	内容
float	fD_INF	無限遠値
unsigned int	nD_INF	※1
float	fBF	BF 値
float	fBaseLength	基線長
float	dZ	ΔZ 値　※1
float	fViewAngle	視野角
unsigned int	nImageWidth	画像幅
unsigned int	nImageHeight	画像高さ
unsigned int	nProductNumber	製品番号
unsigned int	nProductNumber2	製品番号（上位桁）
char	nSerialNumber	製品シリアル番号文字列 最大　8 文字（半角）
unsigned int	nFPGA_Version_Major	製品 FPGA メジャーバージョン
unsigned int	nFPGA_Version_Minor	製品 FPGA マイナーバージョン
unsigned int	nDistanceHistValue	※1
unsigned int	nParallaxThreshold	※1

※1　未使用です。　将来のために予約されています

GetImageSize

C++/C	GetImageSize(unsigned int* pnWidth, unsigned int* pnHeight)

Parameter	pnWidth	画像幅
	pnHeight	画像高さ
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	画像の幅、高さを取得します

	unsigned int width=0, height=0;
	int ret = GetImageSize(&width, &height);
	if(ret != 0){
	　// Error
	}
	// ・・・データ使用

SetAutoCalibration

C++/C	int SetAutoCalibration(int nMode)

Parameter	nMode	自動調整モードを指定します 0 ：停止 1 ：自動調整 On 2 ：強制調整開始
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	調整モードを設定します

	int mode = 1;
	int ret = SetAutoCalibration(mode);
	if(ret !=0({
	　// Error
	}

GetAutoCalibration

C++/C	int GetAutoCalibration(int* nMode)

Parameter	nMode	現在の自動調整モードを表します bit / 内容 0 / 強制調整動作状態 0 ：待機中 1 ：動作中 1 / 自動調整 On/Off 0 ：Off 1 ：O n 2~15 / 予約
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	調整モードの状態を取得します

	int mode = 0;
	int ret = GetAutoCalibration (&mode);
	if(ret !=0({
	　// Error
	}
	if((mode & 0x00000001) != 0){
	　// 調整中
	}

SetShutterControlMode

C++/C	int SetShutterControlMode(bool nMode)

Parameter	nMode	シングルシャッターモードの使用を指定します true：使用する false：使用しない
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	シングルシャッターモードを使用します

	bool mode = true;
	int ret = SetShutterControlMode (mode);
	if(ret !=0({
	　// Error
	}

GetShutterControlMode

C++/C	int GetShutterControlMode(bool* nMode)

Parameter	nMode	シングルシャッターモードの使用を取得します true：使用する false：使用しない
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	シングルシャッターモードの状態を取得します

	bool mode = false;
	int ret = GetShutterControlMode (&mode);
	if(ret !=0({
	　// Error
	}

SetExposureValue

C++/C	int SetExposureValue(unsigned int nValue)

Parameter	nValue	露光制御の値を指定します 有効設定範囲：2~748 数値が大きいほど暗くなります
Return Value	成功時　0 失敗時　!=0	
Remarks	マニュアルモードを設定時のみ有効です	
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	露光値を設定します

	unsigned int value = 120;
	int ret = SetExposureValue (value);
	if(ret !=0({
	　// Error
	}

※設定値は、画像取り込みで反映されます。画像取り込みを停止している場合に設定した値は、次回の画像取り込み時に読み込むことが出来ます。
画像取り込み中に設定した値は、常時反映されます。

GetExposureValue

C++/C	int GetExposureValue(unsigned int* pnValue)

Parameter	pnValue	現在の設定値を表します
Return Value	成功時　0 失敗時　!=0	
Remarks	マニュアルモードを設定時のみ有効です	
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	現在の設定値を取得します

	unsigned int value = 0;
	int ret = GetExposureValue (&value);
	if(ret !=0({
	　// Error
	}

※画像取り込み中のみ読み込み可能です。それ以外は失敗します。

SetGainValue

C++/C	int SetGainValue(unsigned int nValue)

Parameter	nValue	ゲインの値を指定します 有効設定範囲：0~719
Return Value	成功時　0 失敗時　!=0	
Remarks	マニュアルモードを設定時のみ有効です	
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	ゲイン値を設定します

	unsigned int value = 20;
	int ret = SetGainValue (value);
	if(ret !=0({
	　// Error
	}

GetGainValue

C++/C	int GetGainValue(unsigned int* pnValue)

Parameter	pnValue	現在の設定値を表します
Return Value	成功時　0 失敗時　!=0	
Remarks	マニュアルモードを設定時のみ有効です	
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	現在の設定値を取得します

	unsigned int value = 0;
	int ret = GetGainValue (&value);
	if(ret !=0({
	　// Error
	}

※画像取り込み中のみ読み込み可能です。それ以外は失敗します。

SetNoiseFilter

C++/C	int SetNoiseFilter(unsigned int nDCDX)

Parameter	nDCDX	ノイズフィルターの値を指定します 有効設定範囲： 0 ～255 値が大きいほどノイズ除去が強くなります
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	フィルター値を設定します

	unsigned int value = 4;
	int ret = SetNoiseFilter (value);
	if(ret !=0({
	　// Error
	}

GetNoiseFilter

C++/C	int GetNoiseFilter(unsigned int* nDCDX)

Parameter	nDCDX	現在の設定値を表します
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	現在の設定値を取得します

	unsigned int value = 0;
	int ret = GetNoiseFilter (&value);
	if(ret !=0({
	　// Error
	}

SetMeasArea

C++/C	int SetMeasArea(int nTop, int nLeft, int nRight, int nBottom)

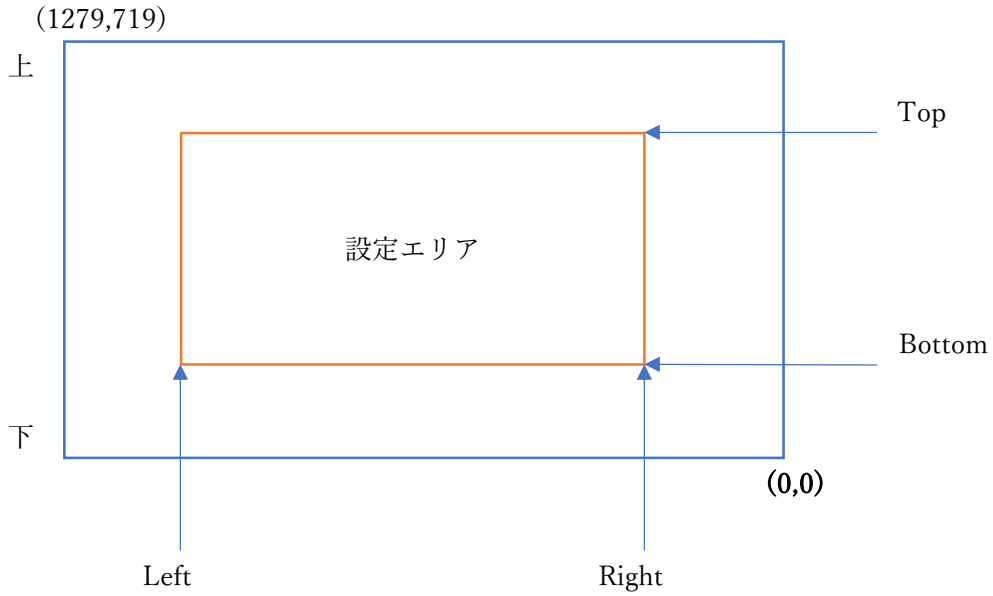
Parameter		露光調整の計算に使用する領域を指定します
	nTop	上座標
	nLeft	左座標
	nRight	右座標
	nBottom	下座標
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	4 点を指定します

	int nTop = 700;
	int nLeft = 1200;
	int nRight = 2;
	int nBottom = 80;
	
	int ret = SetMeasArea (nTop, nLeft, nRight, nBottom);
	if(ret !=0({
	　// Error
	}

【座標系】

・4 か所の座標を指定



GetMeasArea

C++/C	int GetMeasArea(int* nTop, int* nLeft, int* nRight, int* nBottom)

Parameter		露光調整の計算に使用する領域を取得します
	nTop	上座標
	nLeft	左座標
	nRight	右座標
	nBottom	下座標
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	現在の設定値を取得します

	int nTop = nLeft = nRight = nBottom = 0;
	
	int ret = GetMeasArea (&nTop, &nLeft, &nRight, &nBottom);
	if(ret !=0({
	　// Error
	}

SetShutterControlModeEx

C++/C	int SetShutterControlMode(int nMode)

Parameter	nMode	露光調整モードを指定します 0：マニュアルモード 1：シングルシャッターモード 2：ダブルシャッター動作モード 3：ダブルシャッター動作モード （画像合成なし動作）
Return Value	成功時　0 失敗時　!=0	
Remarks	ダブルシャッター動作モードにおける画像合成は、ホストコンピューターで処理を行います 合成処理は GetImage()　呼び出し時に行うため、ホストコンピューターの性能により遅延を生じます カラーモードが On の場合には、ダブルシャッター動作モードに設定できません	
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	シングルシャッターモードを指定します

	int mode = 1;
	int ret = SetShutterControlMode (mode);
	if(ret !=0({
	　// Error
	}

GetShutterControlModeEx

C++/C	int GetShutterControlMode(int* nMode)

Parameter	nMode	現在の露光調整モードを表します 0：マニュアルモード 1：シングルシャッターモード 2：ダブルシャッター動作モード 3：ダブルシャッター動作モード （画像合成なし動作）
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	露光調整モードを取得します

	int mode = 0;
	int ret = GetShutterControlMode (&mode);
	if(ret !=0({
	// Error
	}

SetMeasAreaEx

C++/C	int SetMeasArea(int mode, int nTop, int nLeft, int nRight, int nBottom, int nTop_Left, int nTop_Right, int nBottom_Left, int nBottom_Right)

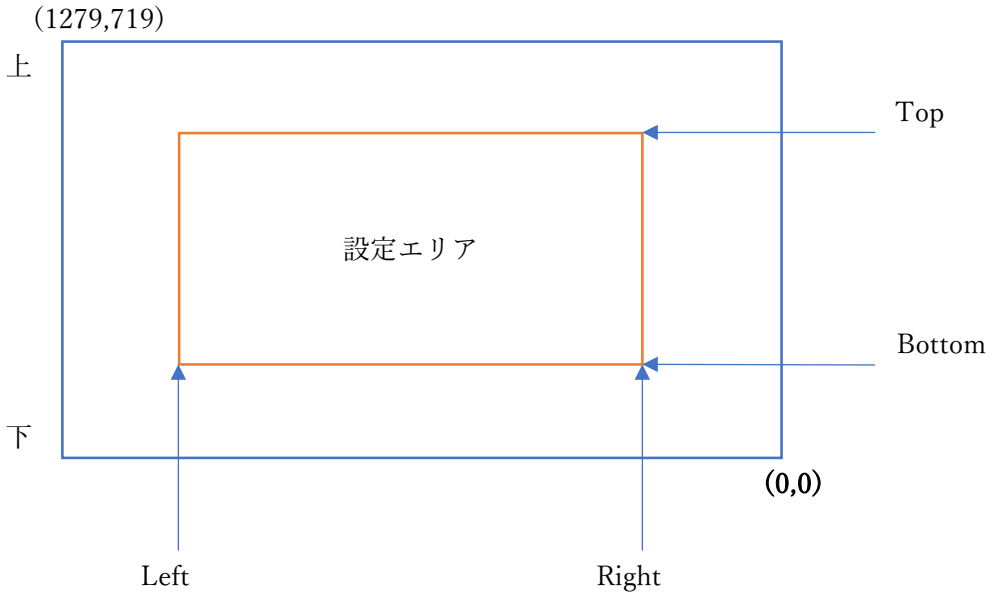
Parameter	mode	露光調整測定エリアの設定方法を指定します 0 ：4 か所の座標を指定 ※以下 mode(0)と書きます 1 ：6 か所の座標を指定 ※以下 mode(1)と書きます
	nTop	mode(0)：上座標 mode(1)：上座標
	nLeft	mode(0)：左座標 mode(1)：(未使用)
	nRight	mode(0)：右座標 mode(1)：(未使用)
	nBottom	mode(0)：下座標 mode(1)：下座標
	nTop_Left	mode(0)：(未使用) mode(1)：上辺の左座標
	nTop_Right	mode(0)：(未使用) mode(1)：上辺の右座標
	nBottom_Left	mode(0)：(未使用) mode(1)：下辺の左座標
	nBottom_Right	mode(0)：(未使用) mode(1)：下辺の右座標
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	mode(0)で 4 点を指定します
	int mode = 0;

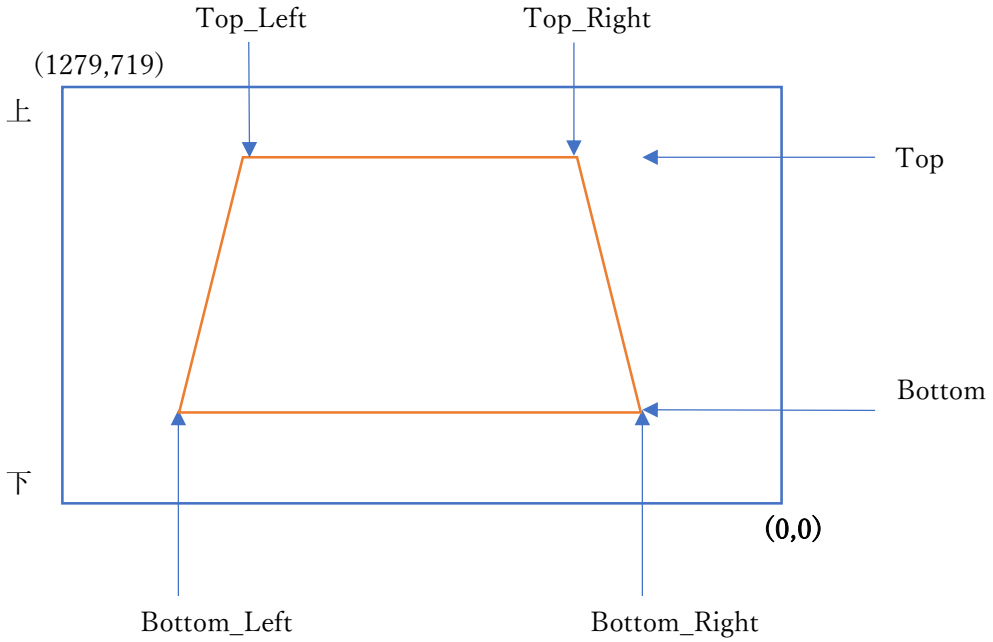
	int nTop = 700;
	int nLeft =1200;
	int nRight = 2;
	int nBottom = 80;
	int nTop_Left = nTop_Right = nBottom_Left = int nBottom_Right = 0;
	
	int ret = SetMeasArea (mode, nTop, nLeft, nRight, nBottom,
	nTop_Left, nTop_Right, nBottom_Left, nBottom_Right);
	if(ret !=0({
	　// Error
	}

【座標系】

・4 か所の座標を指定



・6 か所の座標を指定



※Top_Left=Top_Right または、Bottom_Left=Bottom_Right とすることで、三角形にすることが可能です。

GetMeasAreaEx

C++/C	int GetMeasArea(int* mode, int* nTop, int* nLeft, int* nRight, int* nBottom, int* nTop_Left, int* nTop_Right, int* nBottom_Left, int* nBottom_Right)

Parameter	mode	現在の設定方法を表します 0 ：4 か所の座標を指定 ※以下 moed(0)と書きます 1 ：6 か所の座標を指定 ※以下 moed(1)と書きます
	nTop	moed(0)：上座標 moed(1)：上座標
	nLeft	moed(0)：左座標 moed(1)：(未使用)
	nRight	moed(0)：右座標 moed(1)：(未使用)
	nBottom	moed(0)：下座標 moed(1)：下座標
	nTop_Left	moed(0)：(未使用) moed(1)：上辺左座標
	nTop_Right	moed(0)：(未使用) moed(1)：上辺右座標
	nBottom_Left	moed(0)：(未使用) moed(1)：下辺左座標
	nBottom_Right	moed(0)：(未使用) moed(1)：下辺右座標
Return Value	成功時　0 失敗時　!=0	
Remarks		
See also	失敗時のコードは、エラーコード一覧を参照してください	

Example	現在の設定値を取得します
	int mode = 0;

	int nTop = nLeft = nRight = nBottom = 0;
	int nTop_Left = nTop_Right = nBottom_Left = int nBottom_Right = 0;
	
	int ret = GetMeasArea (&mode, &nTop, &nLeft, &nRight, &nBottom,
	&nTop_Left, &nTop_Right, &nBottom_Left, &nBottom_Right);
	if(ret !=0({
	　// Error
	}

8．エラーコード一覧

コード	内容	備考
1	FT_INVALID_HANDLE	FTDI エラー
2	FT_DEVICE_NOT_FOUND	FTDI エラー
3	FT_DEVICE_NOT_OPENED	FTDI エラー
4	FT_IO_ERROR	FTDI エラー
5	FT_INSUFFICIENT_RESOURCES	FTDI エラー
6	FT_INVALID_PARAMETER	FTDI エラー
7	FT_INVALID_BAUD_RATE	FTDI エラー
8	FT_DEVICE_NOT_OPENED_FOR_ERASE	FTDI エラー
9	FT_DEVICE_NOT_OPENED_FOR_WRITE	FTDI エラー
10	FT_FAILED_TO_WRITE_DEVICE	FTDI エラー
11	FT_EEPROM_READ_FAILED	FTDI エラー
12	FT_EEPROM_WRITE_FAILED	FTDI エラー
13	FT_EEPROM_ERASE_FAILED	FTDI エラー
14	FT_EEPROM_NOT_PRESENT	FTDI エラー
15	FT_EEPROM_NOT_PROGRAMMED	FTDI エラー
16	FT_INVALID_ARGS	FTDI エラー
17	FT_NOT_SUPPORTED	FTDI エラー
18	FT_NO_MORE_ITEMS	FTDI エラー
19	FT_TIMEOUT	FTDI エラー
20	FT_OPERATION_ABORTED	FTDI エラー
21	FT_RESERVED_PIPE	FTDI エラー
22	FT_INVALID_CONTROL_REQUEST_DIRECTION	FTDI エラー
23	FT_INVALID_CONTROL_REQUEST_TYPE	FTDI エラー
24	FT_IO_PENDING	FTDI エラー
25	FT_IO_INCOMPLETE	FTDI エラー
26	FT_HANDLE_EOF	FTDI エラー
27	FT_BUSY	FTDI エラー
28	FT_NO_SYSTEM_RESOURCES	FTDI エラー
29	FT_DEVICE_LIST_NOT_READY	FTDI エラー
30	FT_DEVICE_NOT_CONNECTED	FTDI エラー
31	FT_INCORRECT_DEVICE_PATH	FTDI エラー
32	FT_OTHER_ERROR	FTDI エラー

-1	読み込みサイズエラー	
-2	書き込みサイズエラー	
-3	データ受信タイムアウトエラー	
-4	ISC object 未作成エラー	
-5	USB Open エラー	
-6	USB コンフィグレーション エラー	
-7	カメラ コンフィグレーション エラー	
-8	レジスタ設定エラー	
-9	受信スレッドエラー	
-10	リセットエラー	
-11	画像取り込みモードエラー	
-12	画像取り込みモード設定エラー	
-13	補正テーブルエラー	
-14	モード設定得エラー	
-15	キャリブレーションエラー	
-16	画像取得エラー	
-17	無効データ	
-18	モードエラー(画像取り込みモード中以外)	
-19	調整動作中のため画像は無効	
-20	要求を受け付けられない	
-100	USB error	
-101	既に Open 済み	
-102	取得イメージ無し	
-201	自動調整失敗　自動調整を連続で失敗	FPGA エラー
-202	自動調整失敗　自動調整連続失敗が繰り返し発生	FPGA エラー
-203	自動調整失敗　自動調整の調整範囲外	FPGA エラー
-204	画像不良　画像が暗すぎるなど	FPGA エラー
-205	予約	FPGA エラー
-206	自動調整不能エラー	FPGA エラー
-207	予約	FPGA エラー
-208	予約	FPGA エラー
-209	予約	FPGA エラー
-210	予約	FPGA エラー

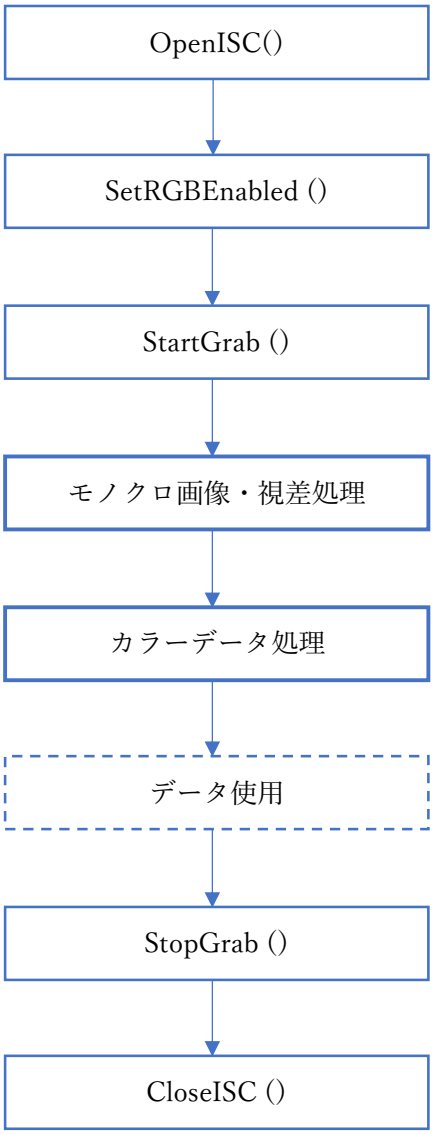
【主な FPGA エラーとその対応】

コード	エラー名	内容
-201	ERR_AUTOCALIB_GIVEUP_WARN	自動調整ワーニング－1
-202	ERR_AUTOCALIB_GIVEUP_ERROR	自動調整失敗－1
-203	ERR_AUTOCALIB_OUTRANGE	自動調整失敗－2
-204	ERR_IMAGEINPUT_IMAGEERROR	センサー画像不良
-206	ERR_AUTOCALIB_FAIL_AUTOCALIB	自動調整失敗－3

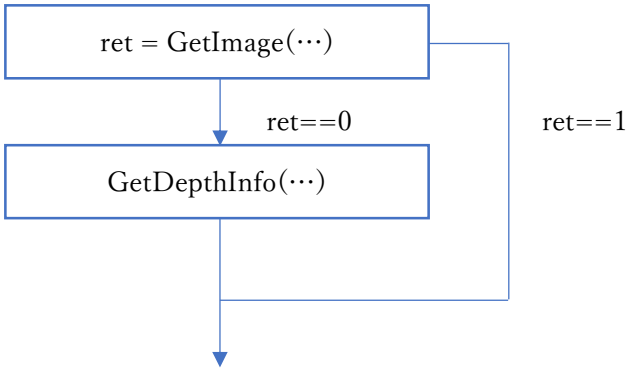
各エラーコードの詳細は、以下となります。

エラー名	種別	原因	対応
自動調整ワーニング－1	ワーニング	一定回数連続で自動調整が発動し調整に失敗した	・シーン変更 ・電源 OFF/ON
センサー画像不良	ワーニング	・レンズに異物が付着している 等	・レンズ異物除去 ・電源 OFF/ON
自動調整失敗－1	エラー	自動調整失敗ワーニングが一定回数連続で発生し調整に失敗した	カメラ再調整 （代理店へお問い合わせください。）
自動調整失敗－2	エラー	カメラ歪みが自動調整範囲を超えてしまった	カメラ再調整 （代理店へお問い合わせください）
自動調整失敗－3	エラー	自動調整に失敗した	カメラ再調整 （代理店へお問い合わせください）

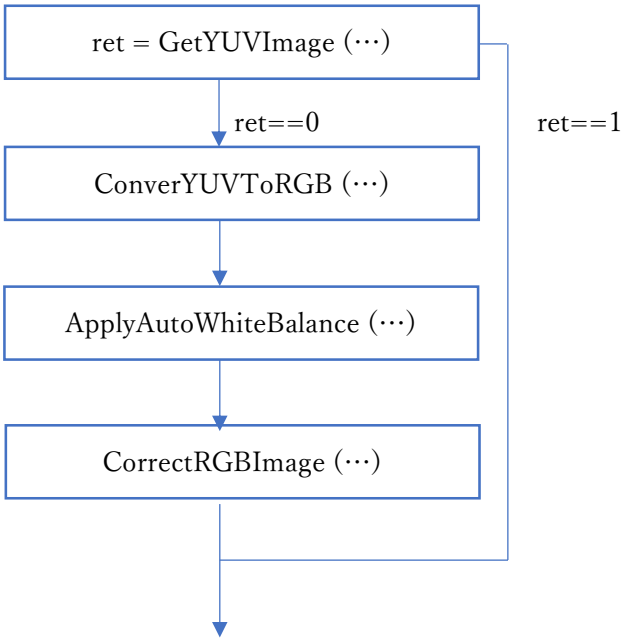
9.呼び出しフロー


OpenISC()
SetRGBEnabled ()
StartGrab ()
モノクロ画像・視差処理
カラーデータ処理
データ使用
StopGrab ()
CloseISC ()

「モノクロ画像・視差処理」



「カラーデータ処理」



１０．注意事項

１０．１　モノクロモード

API SetRGBEnabled において、mode=0 を指定した場合の動作です。

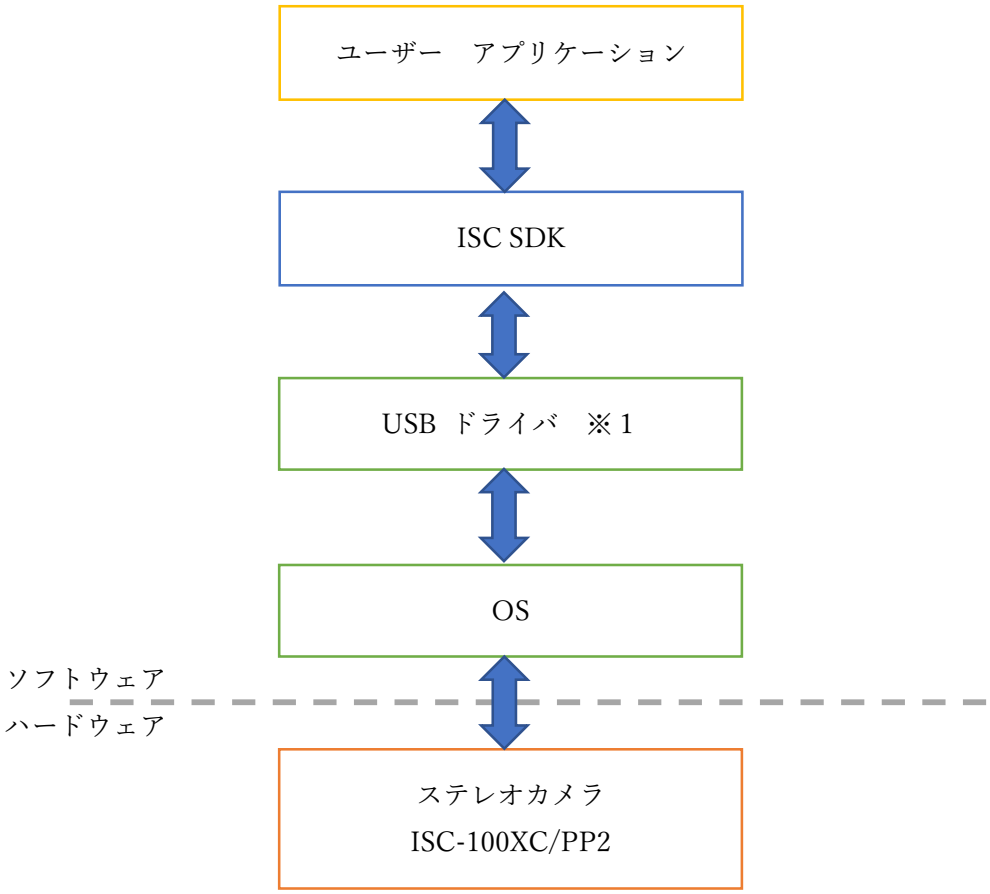
① API GetImage でデータを取得できます。
② API GetYUVImage は常に取得データ無しとなります。

１０．２　カラーモード

API SetRGBEnabled において、mode=1 を指定した場合の動作です。

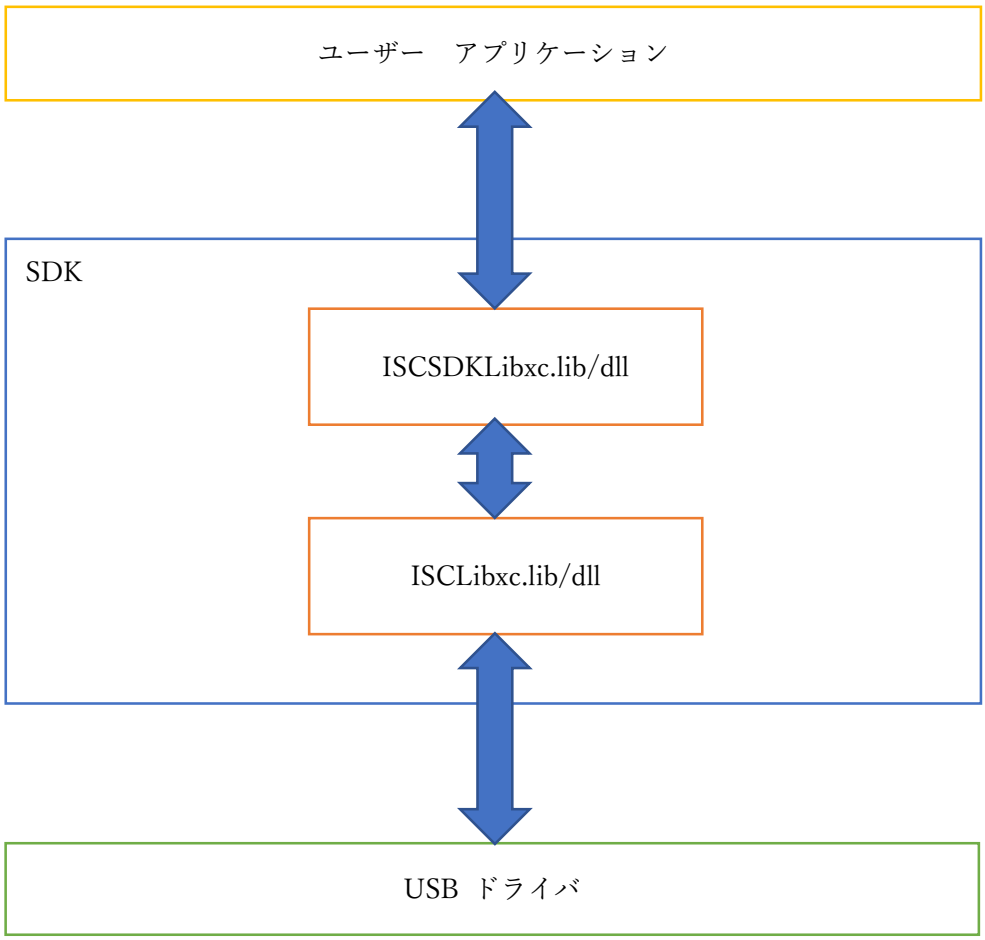
① API GetImage で、モノクロモードと同様にデータを取得できます
② API GetYUVImage で、カラーデータを取得できます

11. SDK　ライブラリアーキテクチャ



※1　Windows 用 USB ドライバ
FTDI 社　提供のドライバを使用します
初めてカメラを接続したときに PC がインターネットに接続されている場合には、自動的にドライバがインストールされます。
それ以外の場合は、FTDI 社のホームページを参照してください。

【SDK ファイル構成】


ユーザー　アプリケーション
SDK
ISCSDKLibxc.lib/dll
ISCLibxc.lib/dll
USB ドライバ

１２．SDK　フォルダー構成

```
SDK
 |
 ├──WINDOWS                  Windows 用 SDK
 |    ├──lib                 Windows 用ライブラリ
 |    |    └──x64            64bit 対応
 |    ├──sample              サンプルコード
 |    |    ├──SampleViewer   Viewre サンプルコード
 |    |    └──SDK
 |    └──viewer              Windows 用ビューワー
 |
 └──Manual                   マニュアル
```

※Linux 版は、標準では含まれていません。
　ご使用の場合は、お問い合わせください。

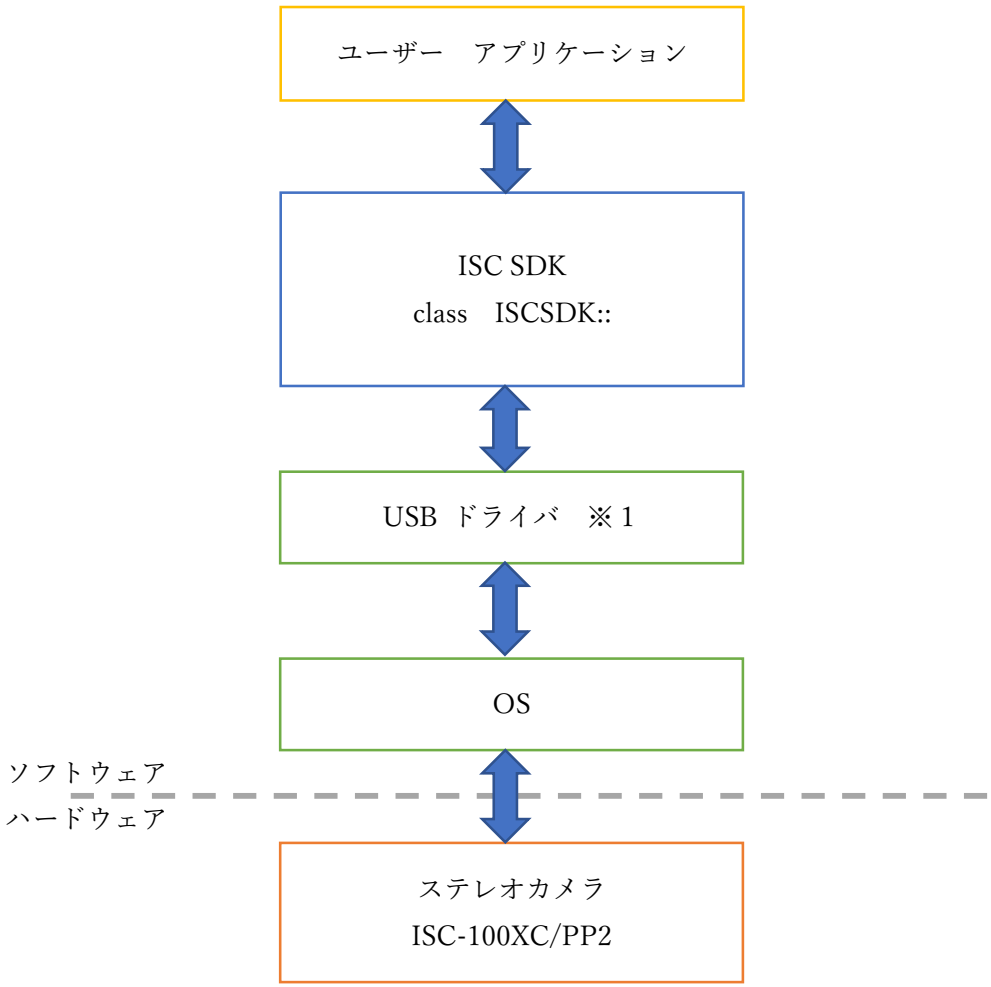
１３．サンプルコードの使い方（Windows 版）

対応環境：Windows10 64bit
開発環境：Visual Studio2017

（１） SampleViewer.sln を起動します
（２） 構成が x64 であることを確認し、Build します
（３） Lib フォルダー内より DLL を実行フォルダーへコピーします
（４） 実行します

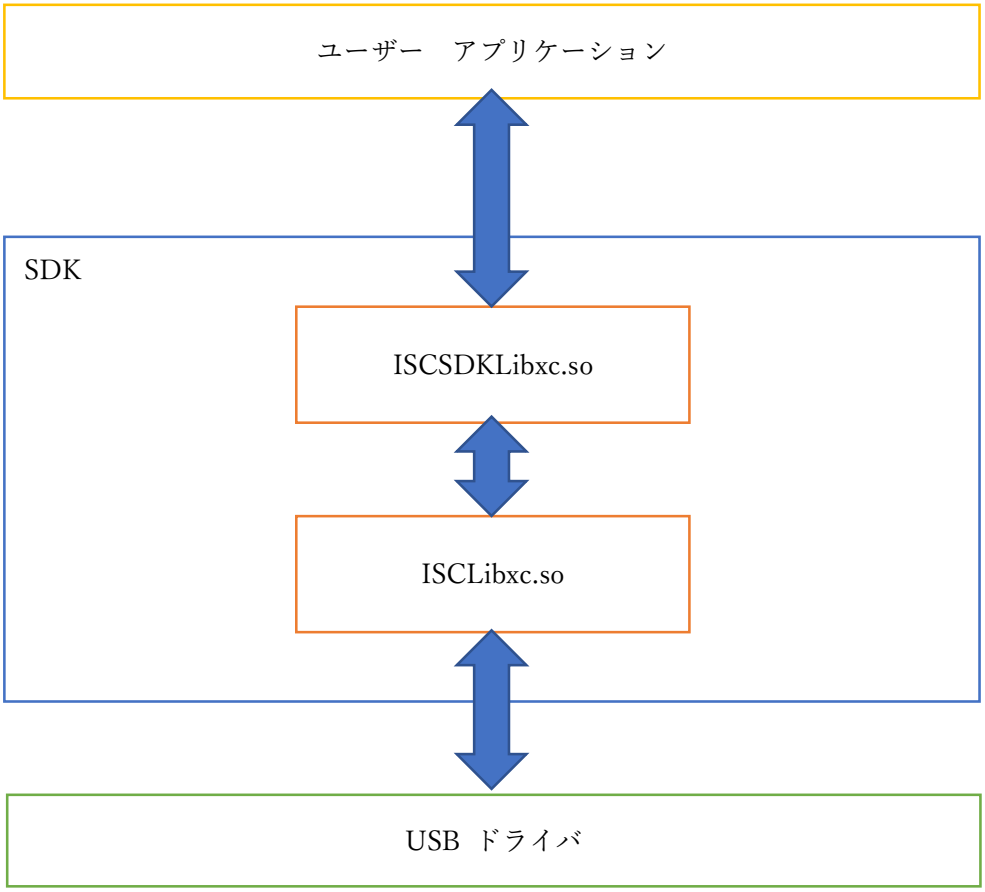
１４．Linux 版

１４．１　SDK ライブラリアーキテクチャ



※１　Linux 用 USB ドライバ
FTDI 提供のドライバを使用します。
インストール方法は、 README.md　を参照してください。

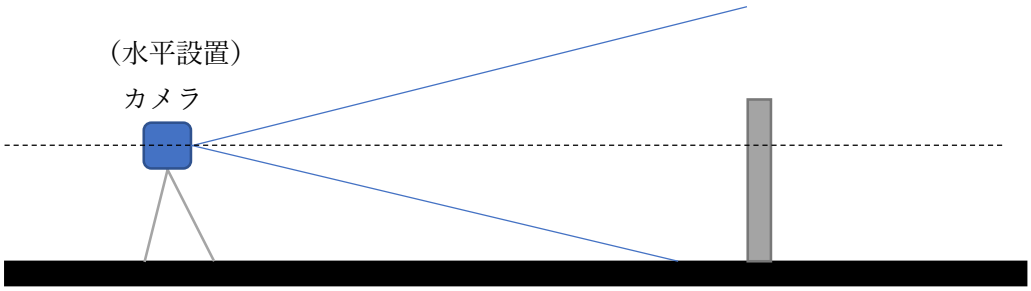
【SDK ファイル構成】



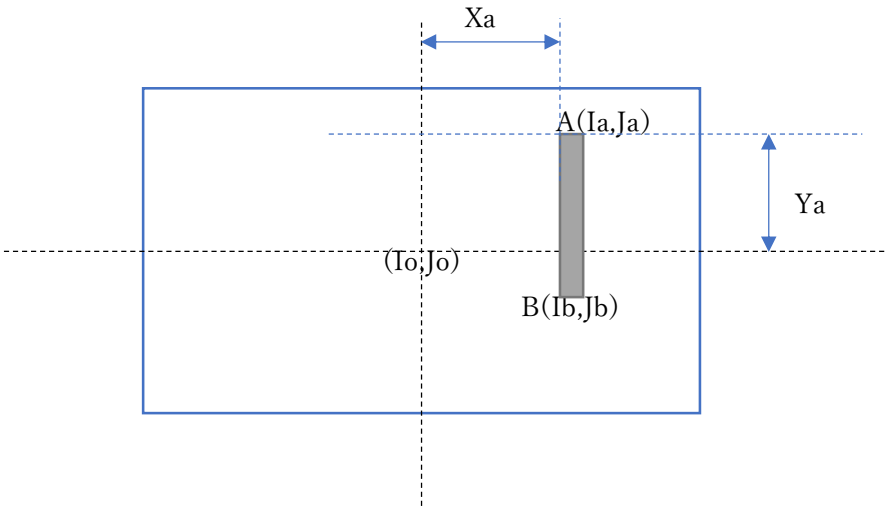
Appendix A　視差の使い方

視差データを距離に変換するための方法を説明します。

＜水平設置における距離の取得＞
カメラ及び対象物の設置が下図のようなケースにおける距離の取得方法を説明します。



カメラでは、下図のようなデータが取得されたとします。



ステレオカメラから取得できるパラメータより以下を使用します。
（GetCameraParamInfo で取得します）

① BF　　：補正校正時に決定されるカメラ固有のパラメータ
② DINF　　：補正校正時に決定されるカメラ固有のパラメータ
③ B　　：基線長

また、画面中心の座標を(Io,Jo)とします。

（画像のサイズが 1280×720 であれば、(Io,Jo)=(640,360)となります）

この時 A における視差値を Da とすると、A までの距離 Za(m)は以下の式で求めることができます。

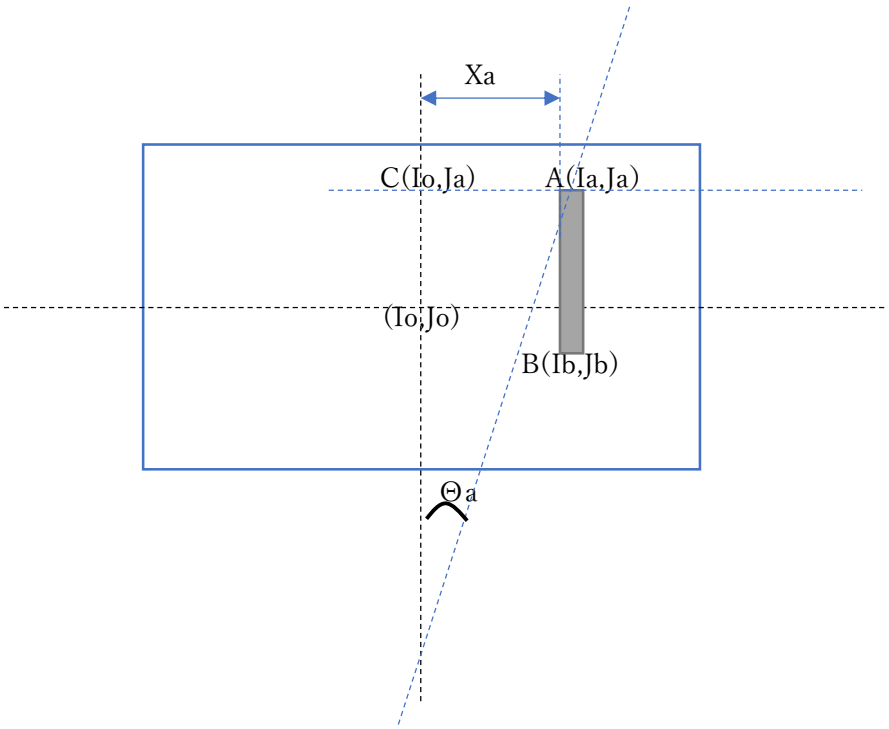
$$Za(m) = \frac{BF}{Da - DINF}$$

中心からの距離、Xa(m)及び Ya(m)は以下となります。

$$Xa(m) = \frac{B \times (Ia - Io)}{Da - DINF}$$

$$Ya(m) = \frac{B \times (Ja - Jo)}{Da - DINF}$$

中心からの角度の求め方を説明します。



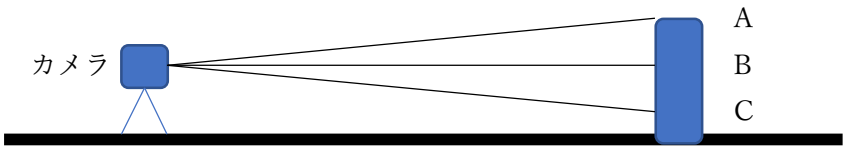
画面中心（C）から A 点までの角度は、以下で求めることができます。

$$\theta a(^\circ) = \tan^{-1}\frac{Xa}{Za}$$

※ ステレオカメラにおける距離

ステレオカメラでは、距離を平面として取得します。

例えば、下図のような構成においては、各 A,B,C までの距離 Za と Zb 及び Zc は等しくなります。(Za＝Zb=Zc)



Appendix B　ダブルシャッター合成時間

ダブルシャッター機能使用時に合成有りを指定した場合、画像取得（API：GetImage 関数）時に画像合成処理を行いますが、この時の処理時間の参考値を以下に示します。
実際の時間は、環境に依存するため参考としてください。

関数（API）	プラットフォーム		
	Windows　※1	Linux-x86　※2	JETSON TX2　※3
GetImage（合成なし）	3msec	8msec	15msec
GetImage（合成あり）	7msec	14msec	18msec

※1　Windows
OS　　Window10 /64bit
CPU　　Core i7-7700 3.6GHz

※2　Linux-x86
OS　　Ubuntu 18.04 LTS
CPU　　Core i5-7300U 2.6GHz

※3　JETSON TX2
OS　　Jetpack 4.2
　　CPU による処理です

Appendix C　カラーデータ処理

1．転送レート

ISC100XC カメラからのデータ転送レートは、最大 60FPS です。
カラーデータ有効時には、モノクロ画像データ（視差データ＋基準画像、又は左右画像）とカラー画像データをカメラより交互に送信するため転送レートは 30FPS となります。。

カラーモード	動作モード	最大転送レート	
		視差＋モノクロ 補正前後モノクロ ※1	カラーデータ ※2
OFF	視差データ＋基準画像	60FPS	
	補正後の基準画像＋比較画像	60FPS	
	補正前の基準画像＋比較画像	60FPS	
ON	視差データ＋基準画像	30FPS	30FPS
	補正後の基準画像＋比較画像	30FPS	30FPS
	補正前の基準画像＋比較画像	30FPS	30FPS

※1　GetImage()で取得
※2　GetYUVImage()で取得

２．処理速度（参考データ）

SDK の提供するカラーデータの処理時間は、以下となります。

なお、実際の時間は、環境に依存するため参考としてください。

関数（API）	プラットフォーム		
	Windows※ 1	Linux-x86※ 2	JETSON TX2※ 3
ConvertYUVToRGB	4msec	6msec	7msec
ApplyAutoWhiteBalance	4msec	5msec	7msec
CorrectRGBImage	10msec	18msec	23msec

※ 1　Windows

OS　　Window10 /64bit

CPU　　Core i7-7700 3.6GHz

※ 2　Linux-x86

OS　　Ubuntu 18.04 LTS

CPU　　Core i5-7300U 2.6GHz

※ 3　JETSON TX2

OS　　Jetpack 4.2

　　CPU による処理です

改版履歴

Rev	Date	Content
1.0.0	2018/05/12	初版
1.1.0	2018/08/15	調整機能追加
1.2.0	2018/11/15	
1.3.0	2018/12/09	電源断検知機能追加
1.4.0	2019/03/20	
1.5.0	2019/05/20	NoiseFilter 機能追加
1.6.0	2019/06/24	輝度測定エリア機能追加
1.6.1	2019/07/01	サポート外機能エラー追加
1.7.0	2020/06/01	カラー対応追加
1.7.1	2020/11/17	補正校正テーブルの順番違いを修正
2.0.0	2020/11/30	SDK V2.0 に対応
2.0.1	2021/3/12	カラーの配列を追加