

*WIL*プログラマーズガイド

プログラム作成 C#編

☆第1版☆

はじめに

本書は画像処理ライブラリ **WIL** を用いて、これから画像処理ソフトウェアの開発を行う方向けのガイドブックです。

Microsoft Visual Studio 2013 を開発環境として **Visual C#**にてプロジェクトの生成から画像処理の実行までの基本的な方法について実際の作業手順を例にして、**WIL** のコーディングについて解説します。

— ご注意 —

- (1) 本書の内容の一部または全部を転載することは固くお断りします。
- (2) 本書の内容については将来予告なしに変更する事があります。

— 商標について —

FAST Vision は株式会社ファーストの日本国内の登録商標です。

Windows は **Microsoft** 社の登録商標です。

その他、各会社名、各製品名は各社の商標または登録商標です。

目次

1. 開発の準備	1
1.1 プロジェクトの作成	1
1.2 参照設定の追加	1
1.3 ライブラリの初期化処理	2
1.4 エラーコードについて	3
2. 画像表示の方法	4
2.1 画像ビューの追加	4
2.2 画像ビューの貼り付け	5
2.1 オーバーレイについて	5
3. 画像入力の方法	7
3.1 カメラからの画像入力	7
3.1.1 カメラ設定ファイルの選択	7
3.1.2 画像入力ボードFVC07 のオープン処理	9
3.1.3 単発取込の実行	9
3.1.4 連続取込の実行	12
3.2 画像ファイルからの入力	16
3.3 FVILリファレンス	19
3.4 サンプルプログラムの紹介	19
4. 処理範囲について	20
5. ベイヤー式カラーカメラの画像変換について	21
5.1 FIEライブラリの初期化処理	21
5.2 ベイヤー色合成	21
5.3 FIEリファレンス	24
6. 2値ブローブ解析	25
6.1 2値化処理	25
6.2 2値ブローブ解析	28
6.3 ブローブ選別フィルタ処理	30
6.4 結果の描画	31
6.5 処理範囲の設定	33
6.6 FVILリファレンス	35
6.7 サンプルプログラムの紹介	36
7. 正規化相関サーチ(GS2)	37
7.1 パターンオブジェクトの生成	37
7.2 正規化相関サーチ(GS2)	40
7.3 結果の描画	42
7.4 FVILリファレンス	44
7.5 サンプルプログラムの紹介	44

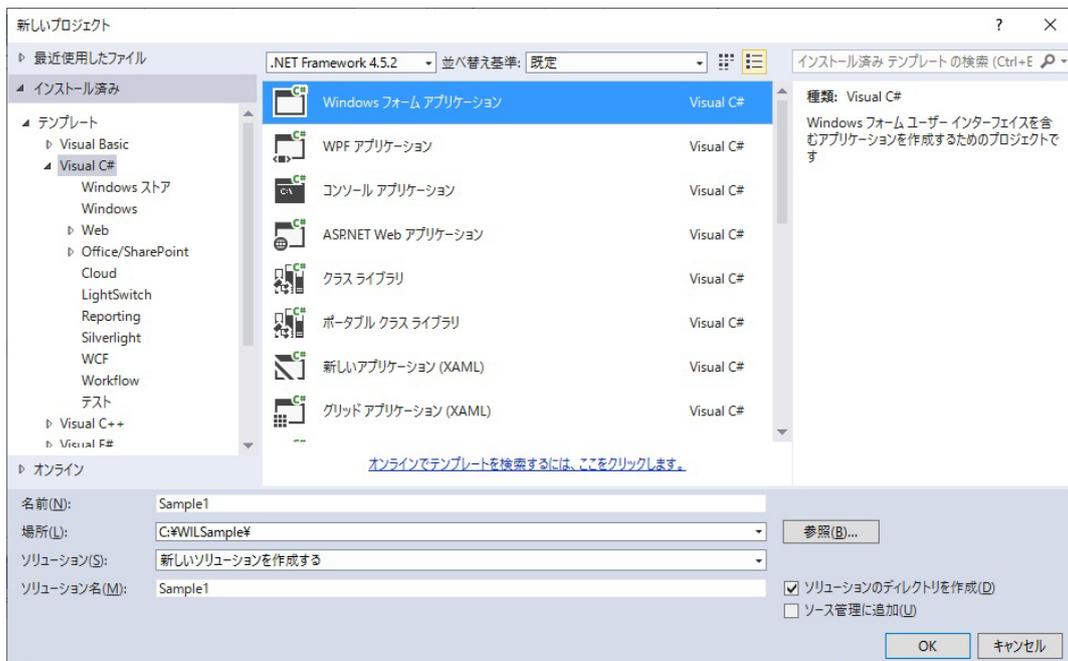
1. 開発の準備

アプリケーションを作成する前の準備として、プロジェクトの作成、ライブラリの参照設定の追加や、ライブラリの初期化の方法について解説します。

1.1 プロジェクトの作成

Visual Studio によるプロジェクトの作成手順を説明します。

- ①Microsoft Visual Studio 2013 を起動します。
- ②メニューより「ファイル(F)」->「新規作成(N)」->「プロジェクト(P)」を選択すると、[画面 1] の「新しいプロジェクト」ダイアログが開きます。
- ③[画面 1]の「新しいプロジェクト」ダイアログより、テンプレートから **Visual C#**を選択し、「Windows フォームアプリケーション」を選択します。ソリューション名、場所を任意に設定して「OK」をクリックしてください。
本書では、以降ソリューション名を「Sample1」、場所を「C:¥WILSample」とします。



【画面 1】

プロジェクトを作成すると、デザイン Form が 1 つあるプロジェクトが生成されます。Form のデフォルト名は Form1 となっています(変更可能)。

本書では以降、この Form1 に対してコードを追加していきます。

1.2 参照設定の追加

WIL ライブラリでは以下の 4 つのアセンブリを提供しています。必要なアセンブリを選択して、以下の手順にてアプリケーションの参照設定に追加を行います。

- ①メニューバーの[プロジェクト]より[参照の追加]を選択し、[画面 2]の「参照マネージャー」ダ

イアログを開きます。

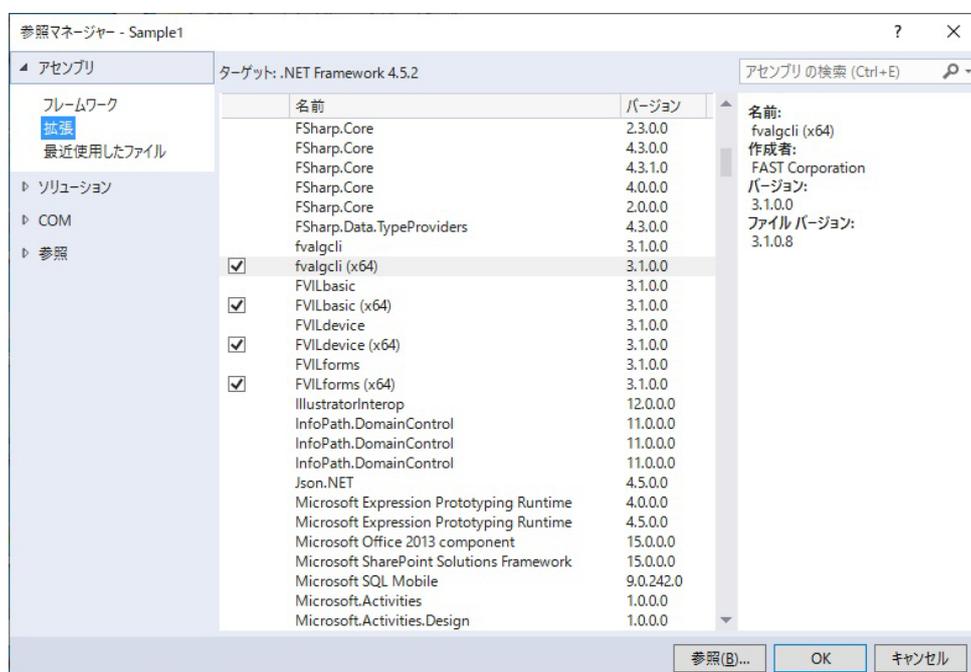
- ②[画面 2]の「アセンブリ」にある「拡張」を選択します。
- ③[画面 2]の「.NET タブ」から「fvalgcli」、「FVILbasic」、「FVILdevice」、「FVILforms」を選択し、OK を押して追加します。

- fvalgcli : 画像処理機能の中核です。
- FVILbasic : 基本機能が集約されたライブラリですので必須です。
- FVILdevice : デジタル入出力とビデオ入力のデバイスコントローラが集約されたライブラリです。
- FVILforms : GUI コンポーネントが集約されたライブラリです。

※ご注意

「参照マネージャー」ダイアログに FVIL が表示されない場合は、FVIL が GAC 登録されていないか、AssemblyFoldersEx が設定されていない可能性があります。その際は、WIL-Diagnostics を使用して再構成してください。

詳細につきましては ReleaseNote の『環境設定・診断ツール』をご参照ください。



【画面 2】

1.3 ライブラリの初期化処理

アプリケーションにて WIL を使用する場合、WIL の初期化が必要になります。

- ①Form1 ダイアログ上でダブルクリックし、Load イベントハンドラを作成します。
- ②Load イベントハンドラ「Form1_Load0」内に、ライブラリの初期化処理を追加します。
下記の赤字部分を「Form1.cs」に追加してください。

```
private void Form1_Load(object sender, EventArgs e)
{
    FVIL._SetUp.InitVisionLibrary(); //FVIL ライブラリの初期化…(1)
}
```

- (1)処理を開始する前に、この関数を呼び出してライブラリを初期化する必要があります。
アプリケーション起動時に一度だけ行ってください。
また、試用期間中であっても、この初期化処理は必ず行う必要があります。
なお、初期化を行わなかった場合は、プロテクトされたクラスの実行でエラーになります。

1.4 エラーコードについて

WIL では、FVIL クラスで発生した例外を通知する為の例外クラスとして `FVIL.CFviException` を用意しています。

`try-catch` にて取得したエラーコード番号に関しては、WIL のヘルプファイル `FVIL Reference` の `CFviException` にカテゴリ分けされた一覧を記載しています。各カテゴリのクラス部分をクリックすると、そのクラスのエラーコードと、その内容が記載されているページが表示されますので、そちらをご参照ください。

・ FVIL Reference

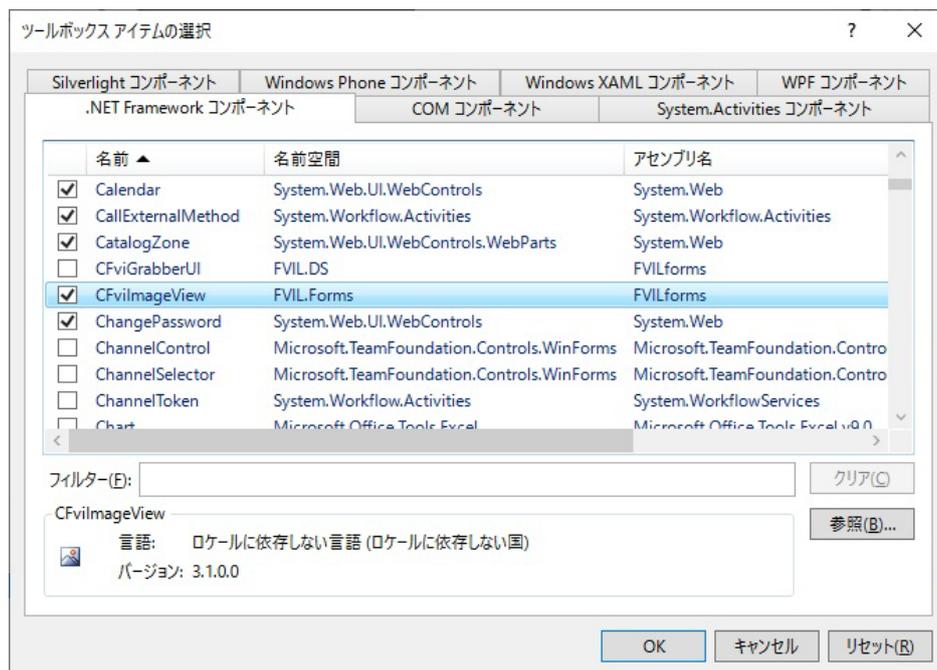
目次より→FVIL リファレンス→FVIL→CFviException

2. 画像表示の方法

画像を表示する画像ビューの設定について説明します。

2.1 画像ビューの追加

- ①メニューバーの「表示」より「ツールボックス」を選択し、ツールボックスを表示します。
- ②ツールボックス上で右クリック→「アイテムの選択」をクリックし、[画面 3]の「ツールボックスのアイテム選択」ダイアログより、「CFviImageView」にチェックし、[OK]ボタンを押します。



[画面 3]

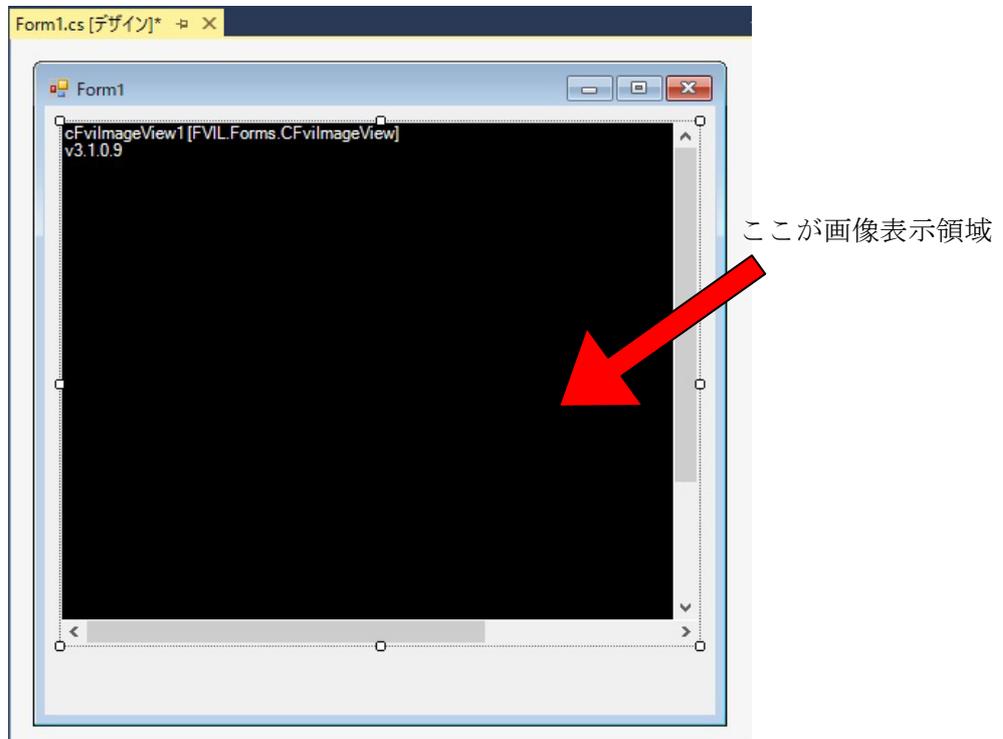
- ③[画面 4]の赤丸部の様に、デザイン時のツールボックス内に「CfviImageView」が追加されます。



[画面 4]

2.2 画像ビューの貼り付け

- ① ツールボックスより前項にて追加した画像ビュー「CFviImageView」を選択し、Form1.cs[デザイン]の Form1 にドロップして、画像ビューを[画面 5]の様に貼り付けます。
この操作を行なうと、ソースコード上では「cFviImageView1」という名称のコントロールが生成されます。



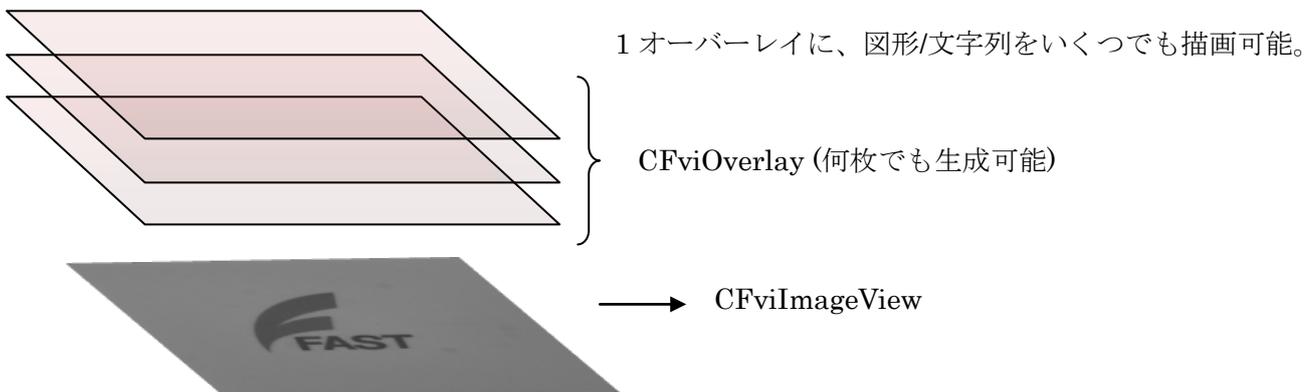
[画面 5]

2.1 オーバーレイについて

画像上への線や文字などの描画は、FVIL.GDI.CfviOverlay(オーバーレイ)へ書き込むことで実現します。

オーバーレイとは、画像の上に重なった透明なシートのようなものとお考え下さい。

オーバーレイは何枚でも生成できますので、例えば1枚目のオーバーレイのみ表示したり、1枚目と2枚目を重ねて表示することが可能です。



図形の表示は、下記のように描画データ型変数(文字・円・点等)を作成し、これをオーバーレイに追加します。

```
// 十字マーク表示
FVIL.GDI.CFviGdiPoint gdiPos = new FVIL.GDI.CFviGdiPoint(); //描画用点データ生成
gdiPos.X = 256;           //X座標設定
gdiPos.Y = 240;           //Y座標設定
gdiPos.Style = FVIL.GDI.FigureStyle.Cross;           //十字スタイルに設定
gdiPos.Size = new Size(5,5);           //サイズ設定
gdiPos.Pen.Color = Color.Pink;           //色設定
m_overlay.Figures.Add( gdiPos );           //オーバーレイへの追加
```

又、オーバーレイに表示している図形ごと、ファイルに保存する場合は下記のように行います。

```
FVIL.Data.CFviImage image = new FVIL.Data.CFviImage(); //画像メモリの生成
cFviImageView1.Display.SaveImage( image );           //表示画面を一旦画像メモリに保存
file.Save( " C:/image.bmp ", image );           //画像をファイルとして保存
```

3. 画像入力の方法

画像入力の方法として、カメラからの画像入力と、BMP ファイル等の画像を読み込む方法があります。これらの方法について解説します。

3.1 カメラからの画像入力

本項では、FVC07CLB を使用した、カメラからの画像入力を行う方法について解説します。

手順は

- ・カメラ設定ファイルの選択(カメラの選択)
- ・画像入力ボードのオープン
- ・画像入力の実行と表示

となります。

画像入力は、単発の画像入力と連続の画像入力の 2 種類の方法について解説します。

なお、プロジェクトの作成、参照設定、画像ビューの追加、画像ビューの貼り付け、ライブラリの初期化処理に関しては、前項を参照して下さい。

3.1.1 カメラ設定ファイルの選択

カメラ映像を取り込む場合、カメラ設定ファイル(INI ファイル形式)を使用して、ビデオ入力デバイスに初期パラメータを設定する必要があります。

本項では、その為のカメラ設定ファイルの設定を行います。

なお、カメラ設定ファイルは、使用するカメラ用のファイルを指定しなければ正しく動作しませんのでご注意ください。

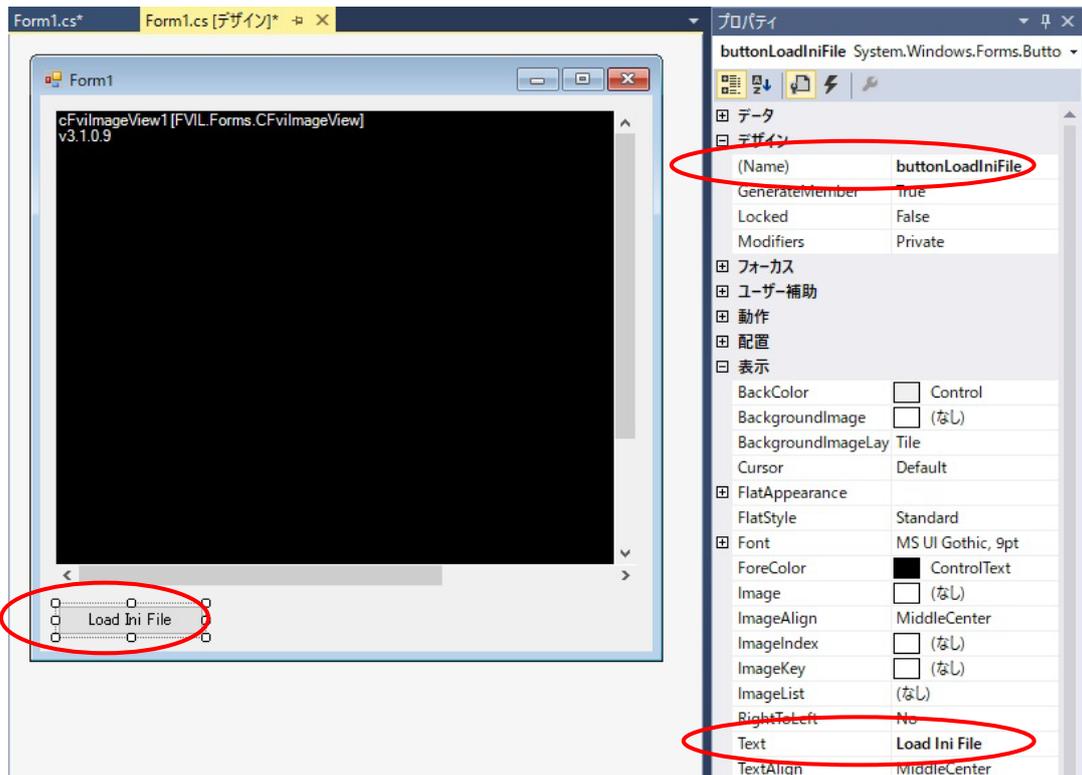
- ① 選択したカメラ設定ファイルのパスを格納する文字列を生成する為、Form1 ダイアログ上で右クリック→「コードの表示」よりコードを表示し、下記の赤字部分を「Form1.cs」に追加してください。

```
public partial class Form1 : Form
{
    //FVIL変数宣言
    public String m_strIniFilePath; //iniファイルのパス用…(1)
```

- (1) 選択したカメラ設定ファイルのパスを格納する文字列型を生成します。

- ② 「Form1.cs[デザイン]」にて、ツールボックスより、Form1 ダイアログへボタンコントロールを追加します。

本書では、[画面 6]の様に追加したボタンコントロールのプロパティの Text を「Load Ini File」、Name を「buttonLoadIniFile」とします。



【画面 6】

- ③ 「Load Ini File」 ボタンをダブルクリックし、イベントハンドラ(buttonLoadIniFile_Click)を作成します。
- ④ イベントハンドラ(buttonLoadIniFile_Click)内に、カメラ設定ファイル読み込みの処理を追加します。下記の赤字部分を「Form1.cs」に追加してください。

```
private void buttonLoadImage_Click( object sender, EventArgs e )
{
    try{
        //ビデオ設定ファイルの読み込み
        OpenFileDialog dlg = new OpenFileDialog(); //…(2)
        dlg.Filter = "iniファイル(*.ini)|*.ini"; //…(3)
        dlg.DefaultExt = "*.INI"; //…(3)

        if (dlg.ShowDialog(this) == DialogResult.OK) //…(4)
        {
            m_strIniFilePath = dlg.FileName; //カメラ設定ファイルのパスを設定…(5)
        }
        //～以降にビデオ入力デバイスFVC07のオープン処理コードを記述します～ …(6)
    }
    catch(FVIL.CFviException ex) //エラーコードの取得…(7)
    {
        MessageBox.Show(ex.Message, "Error"); //…(8)
    }
}
```

- (2) カメラ設定ファイルを選択する為の、OpenFileDialog クラスを生成します。
- (3) OpenFileDialog のファイル種類を決定するフィルタ文字列を「ini」に設定します。
- (4) OpenFileDialog にて、カメラ設定ファイルを選択し、ファイル名称を取得します。

- (5) 選択したカメラ設定ファイルのパスを「m_strIniFilePath」に設定します。
- (6) この行以降に 3.1.2 項のビデオ入力デバイス FVC07 のオープン処理コードを記述します。
- (7) 「try～catch」にて、WIL のエラーコードを取得します。
「catch」の引数に「FVIL.CFviException」を指定する事で、WIL のエラーコードの取得が可能です。
- (8) エラーメッセージをメッセージボックスで表示します。

3.1.2 画像入力ボード FVC07 のオープン処理

画像入力ボードのオープン処理を実行するコードを追加します。

- ①ビデオ入力デバイスを使用する為に、FVC07 クラスの生成と撮像画像を格納する画像メモリを生成します。その為、下記の赤字部分を参考にして「Form1.cs」に追加してください。

```
public partial class Form1 : Form
{
    //FVIL変数宣言
    public String m_strIniFilePath; //iniファイルのパス用
    //FVC07クラス宣言
    public FVIL.Video.CFviVideoFVC07 m_video = new FVIL.Video.CFviVideoFVC07(); //…(1)
    //画像メモリの生成
    public FVIL.Data.CFviImage m_image = new FVIL.Data.CFviImage(); //…(2)
}
```

- (1) カメラ撮像の為にビデオ入力クラスを生成します。
なお、本項では FVC07CLB を使用した例を記載する為、「FVIL.Video.CFviVideoFVC07」を使用します。
- (2) 撮像画像を格納する画像メモリを生成します。
- ②イベントハンドラ(buttonLoadIniFile_Click)内の、3.1.1 項の(6)の行以降に、ビデオ入力デバイス FVC07 のオープン処理を追加します。下記の赤字部分を「Form1.cs」に追加してください。

```
//～以降にビデオ入力デバイスFVC07のオープン処理コードを記述します～

m_video.Open(-1); //ビデオ入力デバイスオープン…(3)
m_video.LoadIniFile(m_strIniFilePath); //カメラ設定ファイルを設定…(4)
m_video.ChangeImageSize(m_image); //画像メモリのサイズを変更…(5)
}
```

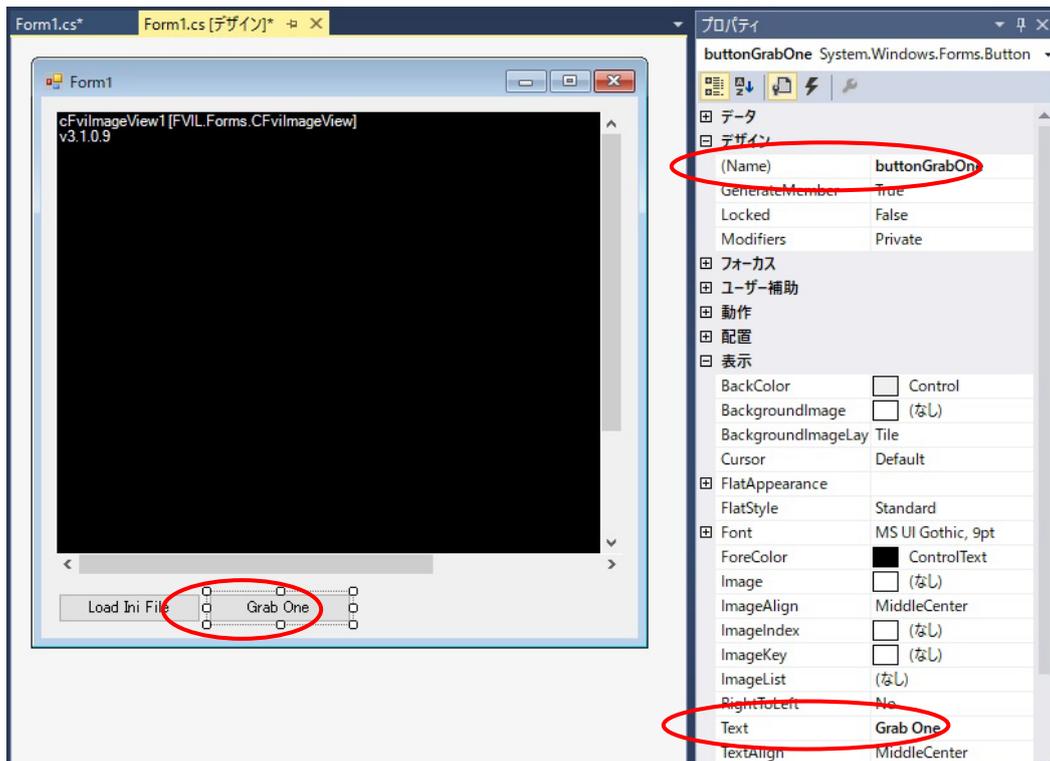
- (3) ビデオ入力デバイスをオープンします。
- (4) ビデオ入力デバイスに、選択したカメラ設定ファイルのパスを渡して設定します。
- (5) 設定した画像メモリの画像サイズを、ロードしたカメラ設定ファイルのサイズに変更します。

3.1.3 単発取込の実行

ボタンをクリックすると画像を 1 枚取り込んで表示するコードを追加します。

- ①3.1.1 項の「Load Ini File」ボタンの時と同じ要領で、Form1 ダイアログへボタンコントロールを追加します。

以下の[画面 7]の様に、追加したボタンコントロールプロパティの Text を「Grab One」、Name を「buttonGrabOne」とします。



【画面 7】

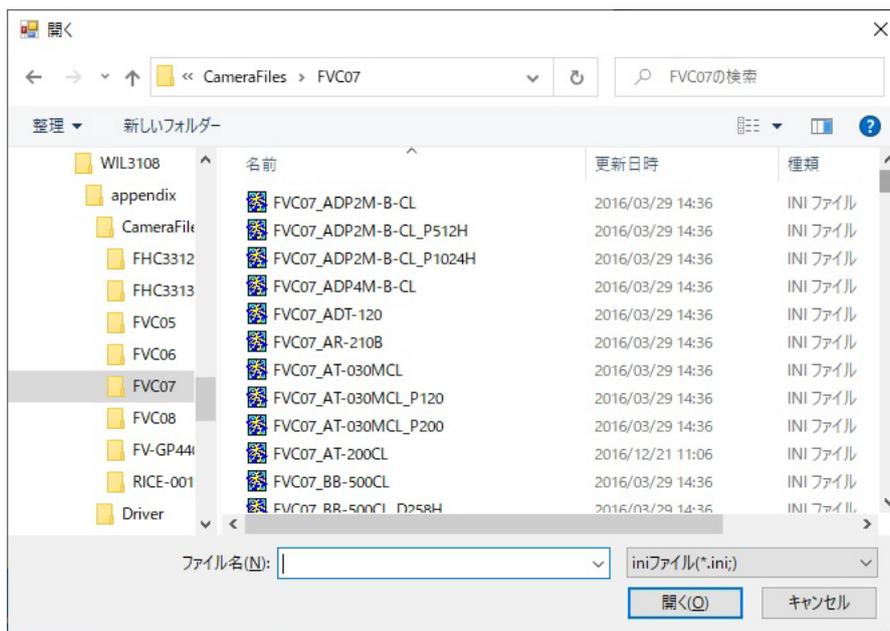
- ② 「Grab One」 ボタンをダブルクリックし、イベントハンドラ(buttonGrabOne_Click)を作成します。
- ③ イベントハンドラ(buttonGrabOne_Click)内に、カメラ映像を取り込み、その画像を表示する処理を追加します。下記の赤字部分を「Form1.cs」に追加してください。

```
private void buttonGrabOne_Click ( object sender, EventArgs e )
{
    try
    {
        m_video.GrabImageSync(m_image);           //カメラ取込み…(1)
        cFviImageView1.Image = m_image;         //画像ビューに表示する画像メモリの設定…(2)
        cFviImageView1.Refresh();               //画像ビューの表示更新…(3)
    }
    catch(FVIL.CFviException ex)                //エラーコードの取得…(4)
    {
        MessageBox.Show(ex.Message, "Error");  //(5)
    }
}
```

- (1) 「GrabImageSync」にて、カメラ映像を取込み「m_image」に格納します。
- (2) 「cFviImageView1」の描画対象画像オブジェクトに「m_image」を設定します。
- (3) 「cFviImageView1.Refresh()」にて、表示を更新します。
- (4) 「try～catch」にて、WILのエラーコードを取得します。
- (5) エラーメッセージをメッセージボックスで表示します。

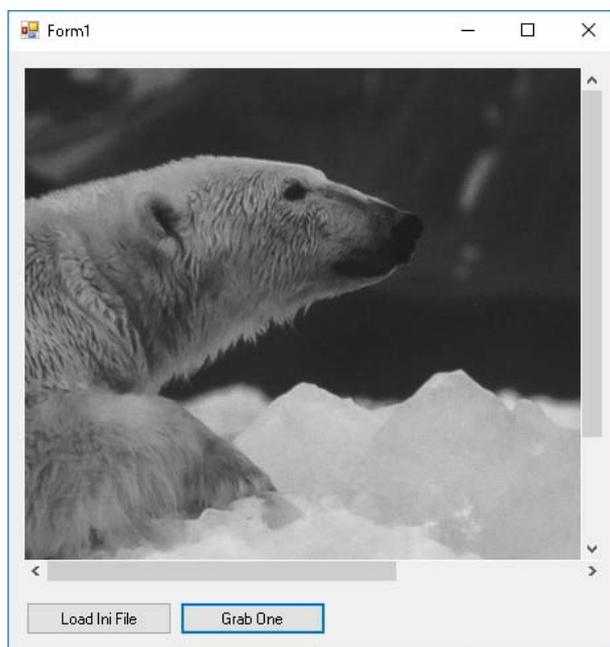
- ④プログラムをビルドし、実行します。

- ⑤アプリケーションにて「Load Ini File」をクリックし、[画面 7]のダイアログにて任意のカメラ設定ファイルをロードします。



[画面 7]

- ⑥カメラ設定ファイルのロード後、「Grab One」ボタンを押し、[画面 8]の様にカメラ映像が表示されれば成功です。



[画面 8]

3.1.4 連続取込の実行

ボタンをクリックすると画像を連続で取り込んで表示するコードを追加します。

①繰り返し取り込み状態の判定用の変数を作成します。

その為、下記の赤字部分を参考にして「Form1.cs」に追加してください。

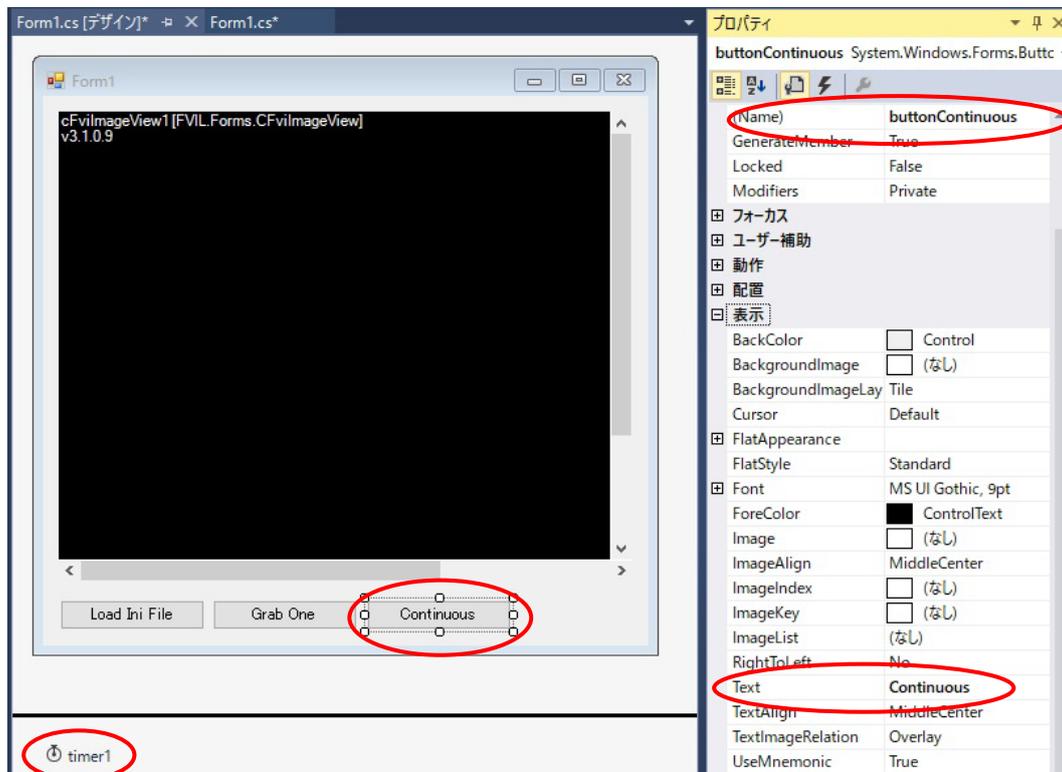
```
public partial class Form1 : Form
{
    //FVIL変数宣言
    public String m_strIniFilePath; //iniファイルのパス用
    //FVC07クラス宣言
    public FVIL.Video.CFviVideoFVC07 m_video= new FVIL.Video.CFviVideoFVC07();
    //画像メモリの生成
    public FVIL.Data.CFviImage m_image = new FVIL.Data.CFviImage();
    //取り込み状態判定用変数
    public bool m_flgContinuous = false; //(1)
}
```

(1) 繰り返し取り込みの状態を判定する変数を作成します。

②3.1.3 項の「Grab One」ボタンの時と同じ要領で、Form1 ダイアログへボタンコントロールを追加します。

本書では、[画面 9]の様に追加したボタンコントロールプロパティの Text を「Continuous」、Name を「buttonContinuous」とします。

③ツールボックス内の「Timer」をダブルクリックし、「Timer1」を追加します。



[画面 9]

④「Continuous」ボタンをダブルクリックし、イベントハンドラ(buttonContinuous_Click)を作成します。

⑤イベントハンドラ(buttonContinuous_Click)内に、Timer コントロールの制御とボタン表示の変

更を行う処理を追加します。

下記の赤字部分を「Form1.cs」に追加してください。

```
private void buttonContinuous_Click( object sender, EventArgs e )
{
    //タイマー制御とボタン表示変更
    if (m_flgContinuous == true) //(1)
    {
        buttonContinuous.Text = "Grab Continuous"; //(2)
        buttonGrabOne.Enabled = true;    //(3)
        m_flgContinuous = false;    //(4)
        timer1.Enabled = false;    //(5)
    }
    else
    {
        buttonContinuous.Text = "Stop";    //(6)
        buttonGrabOne.Enabled = false;    //(7)
        m_flgContinuous = true;    //(8)
        cFviImageView1.Image = m_image; //画像をビューに設定…(9)
        timer1.Interval = 100;    //(10)
        timer1.Start();    //(11)
    }
    Application.DoEvents();    //(12)
}
```

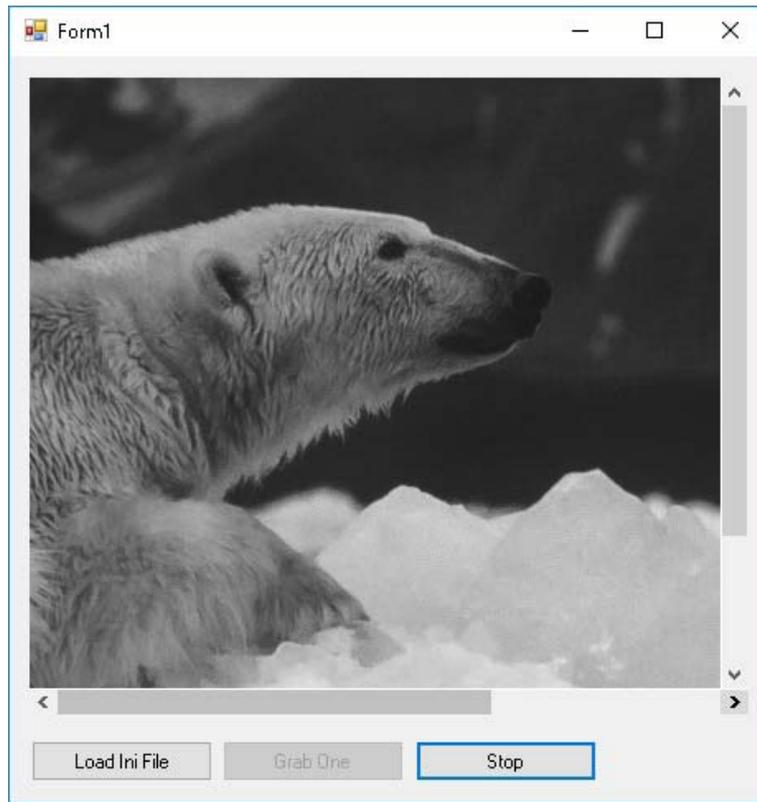
- (1) 繰り返し取り込みの状態を判定します。
- (2) 「Continuous」 ボタンのテキストに「Grab Continuous」を設定します。
- (3) 「Grab One」 ボタンコントロールを無効にします。
- (4) 取り込み状態確認用変数「m_flgContinuous」を「false」に設定します。
- (5) Timer コントロールを「false」に設定し、タイマー処理を停止します。
- (6) 「Continuous」 ボタンのテキストに「Stop」を設定します。
- (7) 「Grab One」 ボタンコントロールを有効にします。
- (8) 取り込み状態確認用変数「m_flgContinuous」を「true」に設定します。
- (9) 「cFviImageView1」の描画対象画像オブジェクトに「m_image」を設定します。
- (10) Timer コントロールのインターバルを「100ms」に設定します。
- (11) 「Start」にて、タイマー処理を開始します。
- (12) 「Application.DoEvents()」にて、タイマーイベントを処理します。

- ⑥ 「Timer1」 ボタンをダブルクリックし、イベントハンドラ(Timer1_Tick)を作成します。
- ⑦ イベントハンドラ(Timer1_Click)内に、カメラ映像を繰り返し取り込み、その画像を表示する処理を追加します。
- 下記の赤字部分を「Form1.cs」に追加してください。

```
private void timer1_Tick( object sender, EventArgs e )
{
    timer1.Stop(); // (13)
    try
    {
        m_video.GrabImageSync(m_image); // カメラ取込み…(14)
        cFviImageView1.Refresh(); // 画像ビューの表示更新…(15)
        timer1.Enabled = true; // (16)
    }
    catch (FVIL.CFviException ex) // エラーコードの取得…(17)
    {
        buttonContinuous.Text = "Grab Continuous"; // (18)
        buttonGrabOne.Enabled = true; // (19)
        m_flgContinuous = false; // (20)
        timer1.Enabled = false; // (21)
        MessageBox.Show(ex.Message, "Error"); // (22)
    }
}
```

- (13) 「Stop」にて、タイマー処理を停止します。
- (14) 「GrabImageSync」にて、カメラ映像を取込み「m_image」に格納します。
- (15) 「cFviImageView1.Refresh()」にて、表示を更新します。
- (16) Timer コントロールに「true」を設定し、タイマー処理を開始します。
- (17) 「try～catch」にて、WIL のエラーコードを取得します。
- (18) 「Continuous」 ボタンのテキストに「Grab Continuous」を設定します。
- (19) 「Grab One」 ボタンコントロールを有効にします。
- (20) 取り込み状態確認用変数「m_flgContinuous」を「false」に設定します。
- (21) Timer コントロールを「false」に設定し、タイマーの処理を停止します。
- (22) エラーメッセージをメッセージボックスで表示します。

- ⑧ プログラムをビルドし、実行します。
- ⑨ アプリケーションにて「Load Ini File」をクリックし、任意のカメラ設定ファイルをロードします。
- カメラ設定ファイルのロード後、「Continuous」 ボタンを押し、[画面 10]の様にカメラ映像が繰り返し表示されれば成功です。



[画面 10]

3.2 画像ファイルからの入力

カメラから画像入力を行う以外に、画像ファイルを読み込んで処理を行う事もできます。

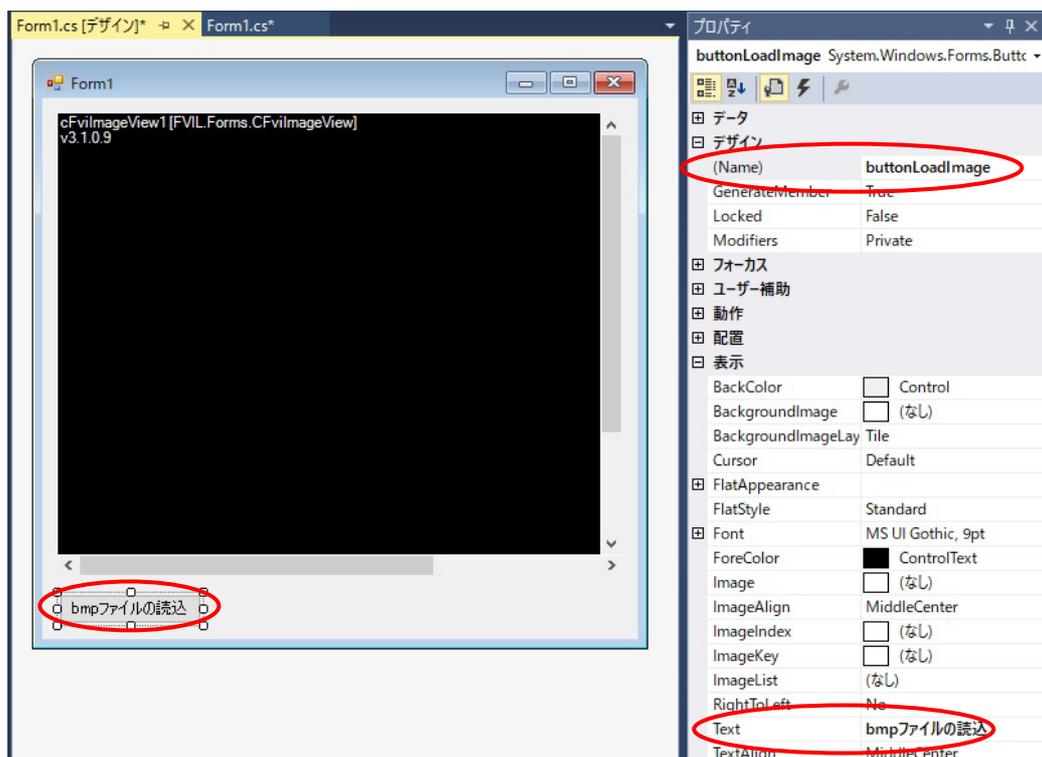
- ①読み込んだ画像を格納する画像メモリを生成する為、Form1 ダイアログ上で右クリック→「コードの表示」よりコードを表示し、下記の赤字部分を「Form1.cs」に追加してください。

```
public partial class Form1 : Form
{
    //FVIL変数宣言
    public FVIL.Data.CFviImage m_image = new FVIL.Data.CFviImage(); //画像メモリの生成…(1)
}
```

- (1)読み込む画像ファイルを格納する画像メモリを生成します。

- ②「Form1.cs[デザイン]」にて、ツールボックスより、Form1 ダイアログへボタンコントロールを追加します。

本書では、[画面 11]の様に追加したボタンコントロールのプロパティの Text を「bmp ファイルの読込」の読込、Name を「buttonLoadImage」とします。



[画面 11]

- ③「bmp ファイルの読込」ボタンをダブルクリックし、イベントハンドラ(buttonLoadImage_Click)を作成します。
- ④イベントハンドラ(buttonLoadImage_Click)内に、画像ファイルの読み込みの処理を追加します。下記の赤字部分を「Form1.cs」に追加してください。

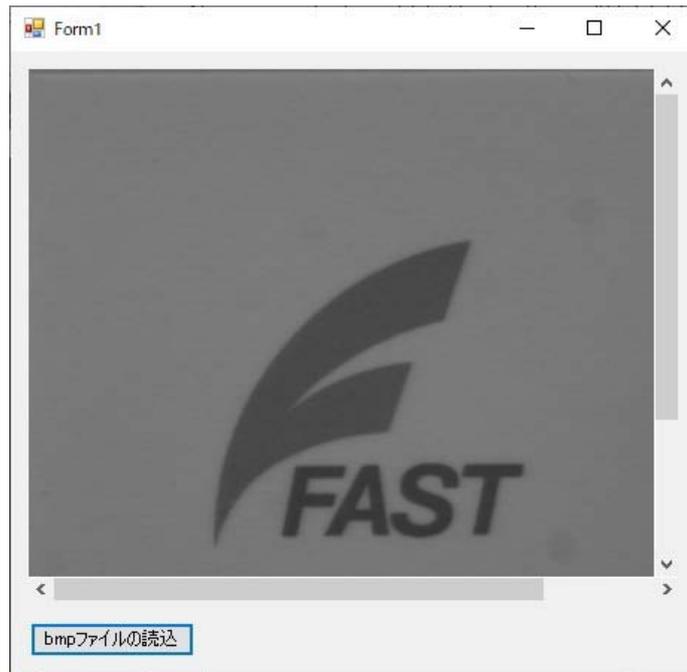
```

private void buttonLoadImage_Click( object sender, EventArgs e )
{
    try{
        //画像ファイルの読み込み
        OpenFileDialog dlg = new OpenFileDialog(); //…(2)
        if (dlg.ShowDialog(this) == DialogResult.OK) //…(3)
        {
            m_image.Load(dlg.FileName);           //画像をロード…(4)
            cFviImageView1.Image = m_image;      //画像ビューに表示する画像メモリの設定…(5)
            cFviImageView1.Refresh();           //画像ビューの表示更新…(6)
        }
    }
    catch(FVIL.CFviException ex)                //エラーコードの取得…(7)
    {
        MessageBox.Show(ex.Message, "Error");
    }
}

```

- (2) 画像ファイルを選択する為の、OpenFileDialog クラスを生成します。
- (3) OpenFileDialog にて、画像ファイルを選択し、ファイル名称を取得します。
- (4) 「m_image.Load()」にて、選択した画像ファイルを画像メモリ「m_image」に格納します。
- (5) 「cFviImageView1」の描画対象画像オブジェクトに「m_image」を設定します。
- (6) 「cFviImageView1.Refresh()」にて、表示を更新します。
- (7) 「try～catch」にて、WIL のエラーコードを取得します。
「catch」の引数に「FVIL.CFviException」を指定する事で、WIL のエラーコードの取得が可能です。

- ⑤プログラムをビルドし、実行します。
- ⑥アプリケーションにて「bmp ファイルの読込」をクリックし、任意の Gray 画像をロードします。
- appendix フォルダ内の¥Samples¥WIL-Builder¥sample01 にある「fast0.BMP」ファイルを使用した場合、[画面 12]の様になります。
本書では、以降「fast0.BMP」を使用して説明を行います。



[画面 12]

WIL では BMP ファイルの他に JPG、PNG、RAW、TIFF ファイルも使用可能です。

また、下記のように、直接使用したい画像ファイルのファイルパスを `CFviImage` コンストラクタに渡し、直接呼び出すことも可能です。

```
FVIL.Data.CFviImage image = new FVIL.Data.CFviImage("C:\¥fast0.BMP");
```

※補足として `CFviImageView` コントロールには表示関連の補助機能も用意されていますので一部紹介いたします。

これら表示関連操作後には、`Refresh()`を呼び出すことで設定が反映されます。

機能	コード例
表示倍率の変更(倍率指定)	<code>cFviImageView1.Display.Magnification = 0.5;</code>
表示倍率をビューに合わせる	<code>cFviImageView1.FitImageSize();</code>
処理範囲設定	<code>cFviImageView1.SetOverlayActive(FVIL.GDI.CFviDrawProcarea.OverlayID, true);</code>
A スコープ表示	<code>cFviImageView1.SetOverlayActive(FVIL.GDI.CFviDrawAscope.OverlayID, true);</code>
グリッド線表示	<code>cFviImageView1.SetOverlayActive(FVIL.GDI.CFviDrawGrid.OverlayID, true);</code>

3.3 FVIL リファレンス

本項にて使用した WIL 関数の詳細説明に関しましては、WIL のヘルプファイル FVIL Reference の以下に記載がありますのでご参照下さい。

- CFviVideoFVC07
FVIL リファレンス→FVIL.Video→CFviVideoFVC07
- CFviVideoFVC07.GrabImageSync
FVIL リファレンス→FVIL.Video→CFviVideoFVC07→Methods→GrabImageSync→
→GrabImageSync(CFviImage)
- CFviImageView
FVIL リファレンス→FVIL.Forms→CFviImageView
- CFviImage
FVIL リファレンス→FVIL.Data→CFviImage
- CFviException
FVIL リファレンス→FVIL→CFviException
※エラーコードと簡易的な内容の記述もありますので、エラーが発生した際はご参照下さい。

3.4 サンプルプログラムの紹介

下記弊社 web にて、ビデオ取込みのサンプルプログラムを公開していますので、参考にして下さい。

https://www.fast-corp.co.jp/software_dl/jp/supportj_sampledl3.php?sid=172&scid=70

4. 処理範囲について

画像処理を行う場合、一般的には画像全体に対して処理を行います。

しかし、様々な理由で処理を行う領域を限定することにより、より効果的に画像処理が行えたり、処理時間の短縮が可能となる場合もあります。

WIL では処理範囲の設定を行う事が可能です。

処理範囲の取得、設定は、画像オブジェクト `CFviImage` の `Window` プロパティにて行います。

なお、処理範囲は各画像オブジェクト毎に保持していますのでご注意ください。

詳細に関しては、WIL のヘルプファイル `FVIL Reference` の下記を参照して下さい。

- **FVIL Reference**

目次より→`FVIL` リファレンス→`FVIL.Data`→`CFviImage`→`Properties`→`Window`

なお、6 項の 2 値ブローブ解析にて実際の処理範囲の設定方法について解説しています。

5. バイヤー式カラーカメラの画像変換について

WIL プログラマーズガイド入門編にて解説していますが、単板式カラーカメラ(バイヤー式カラーカメラ)から入力した画像はバイヤーRAW 画像と言われ、撮像した画像をそのまま表示すると市松模様のようなモノクロ画像になります。その為、バイヤー色合成処理を行い、カラー画像に変換する必要があります。

本項では、バイヤー色合成の処理方法について説明します。

バイヤー色合成は今まで用いてきた FVIL クラスではなく、FIE ライブラリを用いて処理を行います。WIL には FVIL クラス以外に更に粒度の細かいライブラリ群として FIE ライブラリをご用意しています。FIE ライブラリの使用手順についても説明します。

バイヤー色合成の処理は以下の手順にて行います。

- ・カメラからの画像入力
- ・FIE ライブラリの初期化
- ・バイヤー色合成

画像ファイルの読み込みについては、「3.1 カメラからの画像入力」を参照の上、事前にコードを追加してあるものとします。

その他、プロジェクトの作成、参照設定、画像ビューの追加、画像ビューの貼り付け、ライブラリの初期化処理に関しても、1～3 項を参照して下さい。

5.1 FIE ライブラリの初期化処理

アプリケーションにて FIE ライブラリを使用する場合、FIE ライブラリの初期化が必要になります。

- ① 「1.3 ライブラリの初期化処理」で作成した Load イベントハンドラ「Form1_Load0」内に、FIE ライブラリの初期化処理を追加します。下記の赤字部分を「Form1.cs」に追加してください。

```
private void Form1_Load(object sender, EventArgs e)
{
    FVIL._SetUp.InitVisionLibrary();           //FVIL ライブラリの初期化
    fvalgcli.api.fnFIE_setup0;                 //FIE ライブラリの初期化…(1)
}
```

- (1)FIE ライブラリを使用する前に、この関数を呼び出して FIE ライブラリを初期化する必要があります。アプリケーション起動時に一度だけ行ってください。

また、WIL の試用期間中(インストール後 30 日間)であっても、この初期化処理は必ず行う必要があります。

なお、初期化を行わなかった場合は、プロテクトされた FIE ライブラリの実行でエラーになります。

5.2 バイヤー色合成

本項では、バイヤーRAW 画像をカラー画像へと変換するバイヤー色合成に関するコーディングを行

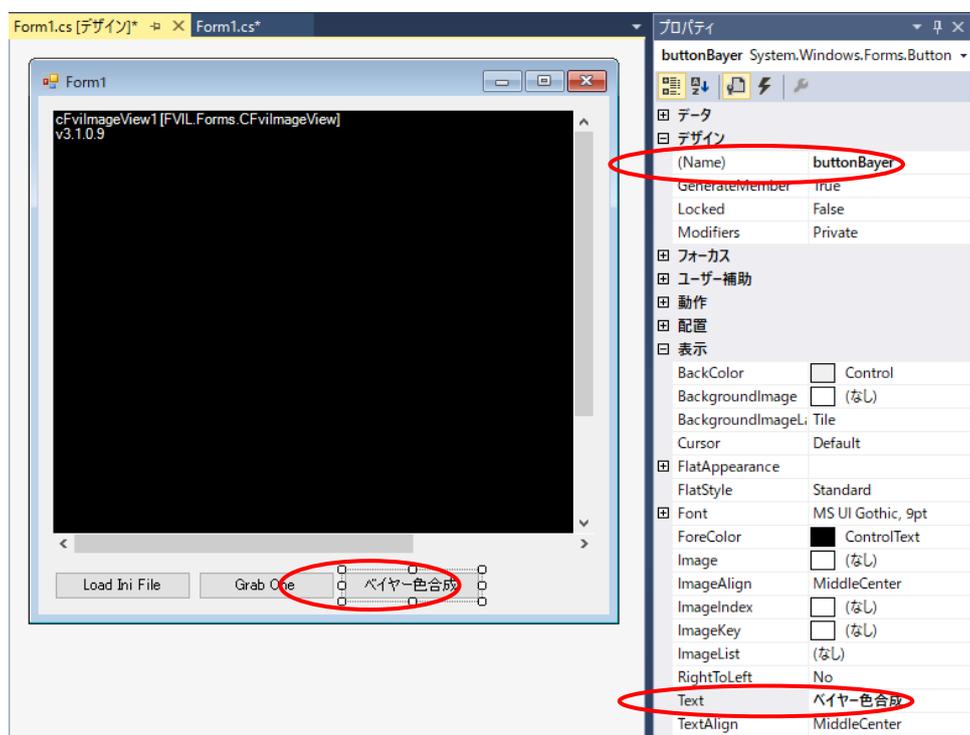
います。

- ①ベイヤー色合成では、入出力に同一の画像メモリを指定する事はできませんので、変換後のカラー画像を格納するメモリを新たに追加する必要があります。
その為、下記の赤字部分を参考に出力用の画像メモリを「Form1.cs」に追加してください。

```
public partial class Form1 : Form
{
    //FVIL変数宣言
    public String m_strIniFilePath; //iniファイルのパス用
    //FVC07クラス宣言
    public FVIL.Video.CFviVideoFVC07 m_video = new FVIL.Video.CFviVideoFVC07();
    //画像メモリの生成
    public FVIL.Data.CFviImage m_image = new FVIL.Data.CFviImage();
    //カラー画像用メモリの生成
    public FVIL.Data.CFviImage m_imagebayer = new FVIL.Data.CFviImage(); //…(2)
}
```

(2)3.1.2 項で生成した画像メモリに続けて、変換後のカラー画像を格納する為の画像メモリを生成します。

- ②3.1.3 項の「Grab One」ボタンの時と同じ要領で、Form1 ダイアログへボタンコントロールを追加します。
本書では、[画面 13]の様に追加したボタンコントロールのプロパティの Text を「ベイヤー色合成」、Name を「buttonBayer」とします。



[画面 13]

- ③「ベイヤー色合成」ボタンをダブルクリックし、イベントハンドラ(buttonBinarize_Click)を作成します。
- ④イベントハンドラ(buttonBayer_Click)内に、ベイヤー色合成処理を追加します。

下記の赤字部分を「Form1.cs」に追加してください。

```
private void buttonBayer_Click( object sender, EventArgs e )
{
    try
    {
        //出力画像の画像メモリのサイズ変更
        m_imagebayer.SetSize(m_image.HorzSize, m_image.VertSize, m_image.ImageType, 3); //…(3)
        //ベイヤー色合成のパラメータ作成
        double[] gain = new double[] { 1.0, 1.0, 1.0 }; //…(4)
        double[] offset = new double[] { 0.0, 0.0, 0.0 }; //…(5)

        fvalgcli.f_err ret = fvalgcli.f_err.F_ERR_NONE; //FIEエラーコード用…(6)

        //線形補間によるベイヤー色合成を行なう
        ret = (fvalgcli.f_err)fvalgcli.api.fnFIE_bayer_interpolation(m_image.GetFIE(),
            m_imagebayer.GetFIE(), fvalgcli.f_cfa_type.F_CFA_RGGB, gain, offset,
            fvalgcli.f_bayer_method.F_BAYER_BILINEAR); //…(7)
        if(fvalgcli.f_err.F_ERR_NONE == ret) //エラー判定…(8)
        {
            cFviImageView1.Image = m_imagebayer; //画像ビューに表示する画像メモリの設定…(9)
            cFviImageView1.Refresh(); //画像ビューの表示更新…(10)
        }
        else
        {
            MessageBox.Show("Error"); //…(11)
        }
    }
    catch(FVIL.CFviException ex) //エラーコードの取得…(12)
    {
        MessageBox.Show(ex.Message, "Error");
    }
}
```

(3) カメラから入力した画像のサイズに合わせて、カラー画像用メモリのサイズを変更します。

(4) ベイヤー色合成のパラメータのゲインを作成します。

(5) ベイヤー色合成のパラメータのオフセットを作成します。

(6) FIE エラーコード用の変数を作成します。

(7) 線形補間によるベイヤー色合成を実行します。

fnFIE_bayer_interpolation()の、入出力画像は「FHANDLE」である必要がありますので、「GetFIE」にて「FHANDLE」を渡します。

(8) FIE 関数のエラー判定を行います。

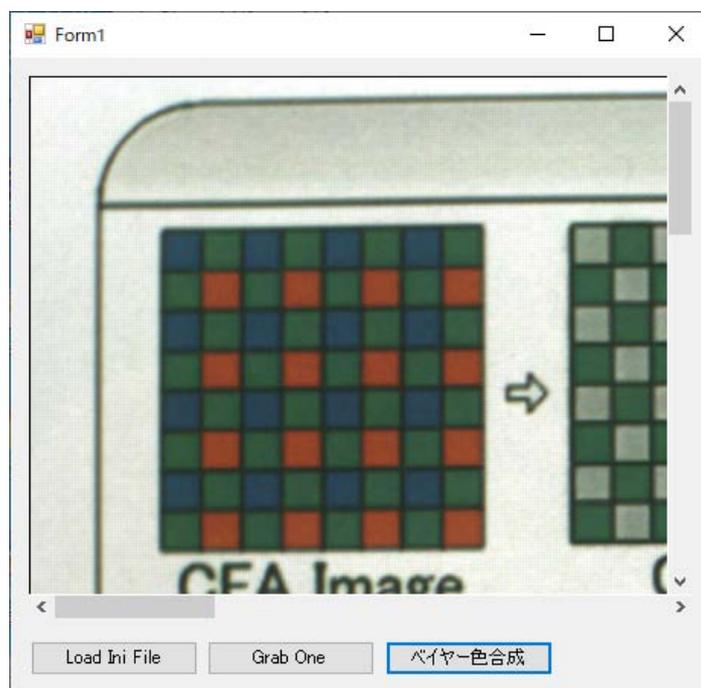
(9) 「cFviImageView1」の描画対象画像オブジェクトに「m_imagebayer」を設定します。

(10) 「cFviImageView1.Refresh()」にて、表示を更新します。

(11) FIE 関数のエラーを表示します。

(12) 「try～catch」にて、WIL のエラーコードを取得します。

- ⑤プログラムをビルドし、実行します。
- ⑥カメラ設定ファイルのロード後、「Grab One」ボタンを押し、カメラから画像を入力します。
画像ロード後、「バイヤー色合成」ボタンを押し、[画面 XX]の様にバイヤー色合成されたカラー画像が表示されれば成功です。



[画面 14]

5.3 FIE リファレンス

本項にて使用した FIE 関数の詳細説明に関しましては、WIL のヘルプファイル FIE Reference の以下に記載がありますのでご参照下さい。

- `fnFIE_bayer_interpolation`

モジュール→FIE module→画像フィルタ→バイヤー色合成→関数→ `fnFIE_bayer_interpolation`

6. 2 値ブローブ解析

本項では、画像処理手法の一つである 2 値ブローブ解析の手順について以下の手順にて解説します。

- ・ 画像ファイルの読み込み
- ・ 画像の 2 値化
- ・ 2 値ブローブ解析、
- ・ 結果の表示、
- ・ 処理範囲の設定

なお、画像ファイルの読み込みについては、「3.2 画像ファイルからの入力」を参照の上、事前にコードを追加してあるものとします。

その他、プロジェクトの作成、参照設定、画像ビューの追加、画像ビューの貼り付け、ライブラリの初期化処理に関しても、1～3 項を参照して下さい。

6.1 2 値化処理

2 値ブローブ解析にて扱う入力画像は、2 値画像である必要があります。

その為、画像メモリに格納した画像が 2 値画像で無い場合、2 値画像へ変換する必要があります。

前項で格納した画像「fast0.BMP」は、濃淡画像ですので、本項では、2 値画像へ変換する階調変換に関するコーディングを行います。

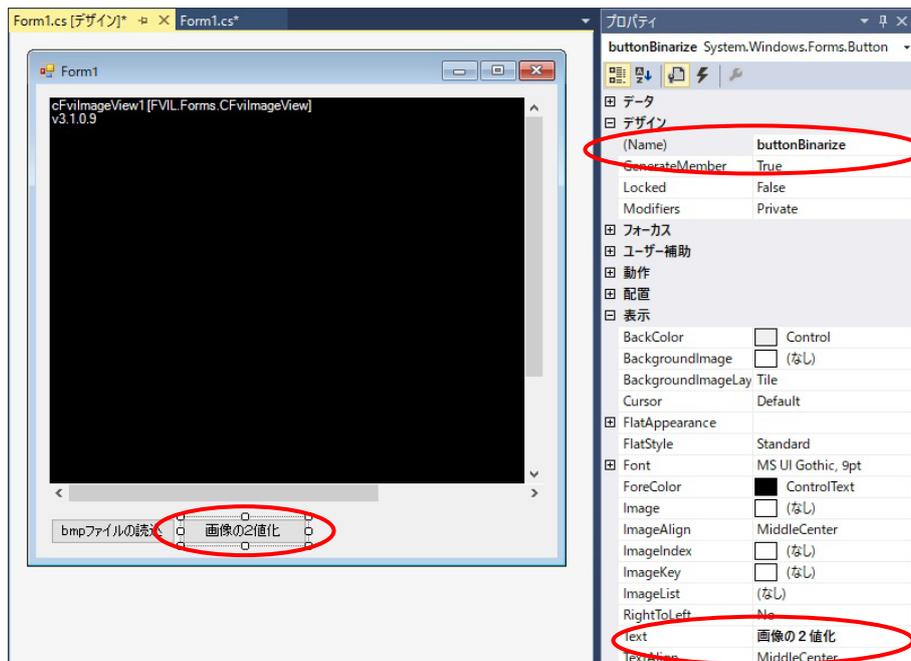
- ① 2 値化クラスでは、入出力に同一の画像メモリを指定する事はできませんので、2 値化した画像を格納するメモリを新たに追加する必要があります。その為、下記の赤字部分を参考に 2 値画像用の画像メモリを「Form1.cs」に追加してください。

```
public partial class Form1 : Form
{
    //FVIL変数宣言
    public FVIL.Data.CFviImage m_image = new FVIL.Data.CFviImage(); //画像メモリの生成
    public FVIL.Data.CFviImage m_imagebin = new FVIL.Data.CFviImage(); //2 値画像用メモリの生成…
    (1)
}
```

- (1) 前項で生成した画像メモリに続けて、変換後の 2 値画像を格納する為の画像メモリを生成します。

- ② 3.2 項の「bmp ファイルの読込」ボタンの時と同じ要領で、Form1 ダイアログへボタンコントロールを追加します。

本書では、[画面 15]の様に追加したボタンコントロールのプロパティの Text を「画像の 2 値化」、Name を「buttonBinarize」とします。



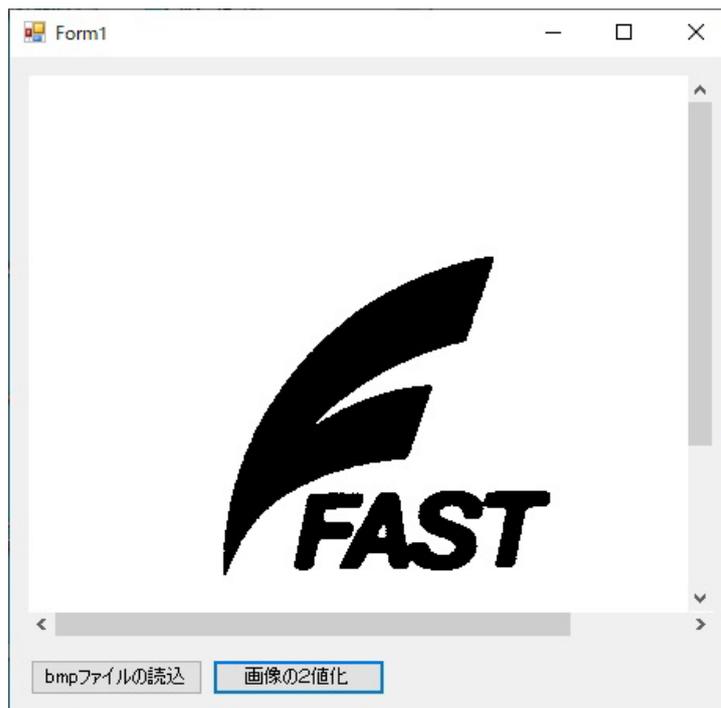
[画面 15]

- ③ 「画像の2値化」 ボタンをダブルクリックし、イベントハンドラ(buttonBinarize_Click)を作成します。
- ④ イベントハンドラ(buttonBinarize_Click)内に、画像の2値化処理を追加します。
下記の赤字部分を「Form1.cs」に追加してください。

```
private void buttonBinarize_Click( object sender, EventArgs e )
{
    // 2値計測クラスのオブジェクト生成
    FVIL.Conversion.CFviBinarize binarize = new FVIL.Conversion.CFviBinarize(); //…(2)
    try
    {
        binarize.SrcImages[0] = m_image; //入力画像を設定…(3)
        binarize.DstImages[0] = m_imagebin; //出力画像を設定…(4)
        if(!binarize.IsValid()) //画像の整合性検査…(5)
        {
            binarize.Validate(); //画像を有効化…(6)
        }
        binarize.Threshold = 100; // 2値化閾値の設定…(7)
        binarize.Execute(); // 2値化実行…(8)

        cFviImageView1.Image = m_imagebin; //画像ビューに表示する画像メモリの設定…(9)
        cFviImageView1.Refresh(); //画像ビューの表示更新…(10)
    }
    catch(FVIL.CFviException ex) //エラーコードの取得…(11)
    {
        MessageBox.Show(ex.Message, "Error");
    }
}
```

- (2) 2 値化クラスのオブジェクトを生成します。
 - (3) 2 値化処理を行う画像メモリに「m_image」を設定します。
 - (4) 2 値化処理を行った結果を出力する画像メモリに「m_imagebin」を設定します。
 - (5) 設定した入力画像と出力画像が処理可能か否か「IsValid()」にて判定します。
処理可能な条件については、2 値化クラスの説明をご参照ください。
 - (6) 「Validate()」にて、設定した入力画像に合わせて、出力画像を有効化します。
 - (7) パラメータ「Threshold」に、2 値化閾値を設定します。
本書では、「fast0.BMP」に合わせて「100」の固定値を設定しています。
※別の画像を使用する場合は、その画像に合わせた値に変更してください。
 - (8) 「Execute()」にて、2 値化処理を実行します。
 - (9) 「cFviImageView1」の描画対象画像オブジェクトに「m_imagebin」を設定します。
 - (10) 「cFviImageView1.Refresh()」にて、表示を更新します。
 - (11) 「try～catch」にて、WIL のエラーコードを取得します。
- ⑤ プログラムをビルドし、実行します。
 - ⑥ 「bmp ファイルの読込」をクリックし、「fast0.BMP」をロードします。
画像ロード後、「画像の 2 値化」ボタンを押し、[画面 16]の様に画像が 2 値化された画像が表示されれば成功です。



[画面 16]

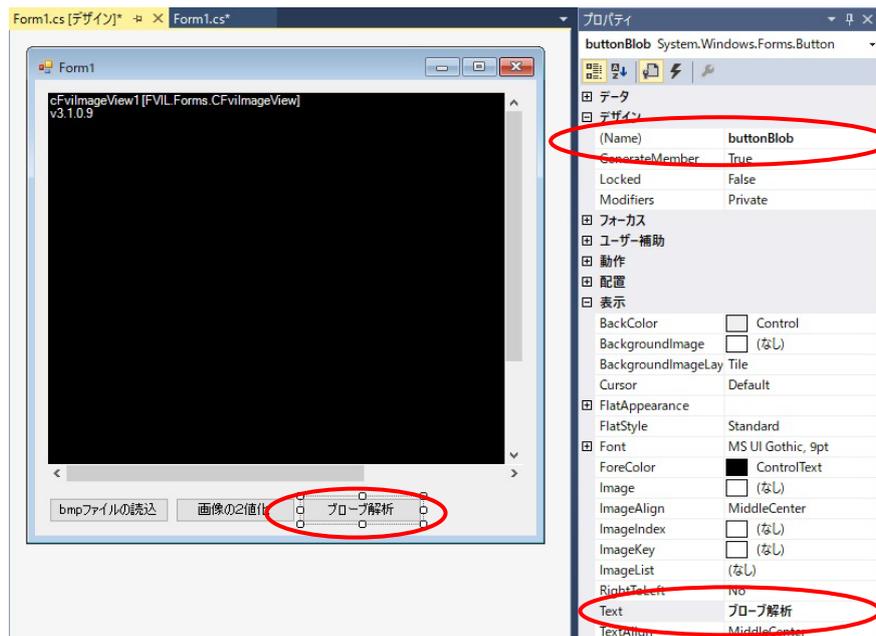
6.2 2値ブローブ解析

2 値ブローブ解析とは、2 値画像内の白塊または黒塊の図形を解析し、その特徴量を取得する手法です。

本項では、前項で2 値化した画像に対して、2 値ブローブ解析を行い、得られた解析結果に対してフィルタリング処理を行い、選別したブローブ情報をオーバーレイへ描画します。

①前項の「画像の2 値化」ボタンの時と同じ要領で、Form1 ダイアログへボタンコントロールを追加します。

本書では、[画面 17]の様に追加したボタンコントロールプロパティの Text を「ブローブ解析」、Name を「buttonBlob」とします。



[画面 17]

②「ブローブ解析」ボタンをダブルクリックし、イベントハンドラ(buttonBlob_Click)を作成します。

③イベントハンドラ(buttonBlob_Click)内に、ブローブ解析処理を追加します。

下記の赤字部分を「Form1.cs」に追加してください。

```

private void buttonBlob_Click( object sender, EventArgs e )
{
    // 2 値ブローブ解析クラスのオブジェクト生成
    FVIL.Blob.CFviBlob blob = new FVIL.Blob.CFviBlob(m_imagebin); //…(1)

    try
    {
        blob.SrcImages[0] = m_imagebin; //入力画像を設定…(2)
        blob.Param.ColorMode = FVIL.Blob.ObjectColor.BlackFG_WhiteBG; //解析対象色を設定…(3)

        blob.Execute(); // 2 値ブローブ解析実行…(4)

        //～以降にブローブ選別フィルタ処理コードを記述します～ …(5)
    }
    catch (FVIL.CFviException ex) //エラーコードの取得…(6)
    {
        MessageBox.Show(ex.Message, "Error");
    }
}

```

- (1) 2 値ブローブ解析のオブジェクトを生成します。
- (2) 2 値ブローブ解析処理を行う画像メモリに「m_imagebin」を設定します。
- (3) パラメータ「ColorMode」に、前景（解析対象の色）が黒、背景が白の「BlackFG_WhiteBG」を設定します。
- (4) 「Execute()」にて、2 値ブローブ解析処理を実行します。
解析処理の結果は「FVIL.Blob.CFviBlob.Result()」に格納されます。
- (5) この行以降に次項のブローブ選別フィルタ処理コードを記述します。
- (6) 「try～catch」にて、WIL のエラーコードを取得します。

6.3 ブローブ選別フィルタ処理

前項にて、2 値ブローブ解析を実行した際、解析処理の結果は「FVIL.Blob.CFviBlob.Result()」に格納されます。

ただし、その結果には、ノイズ等を含む画像内にある全てのブローブの解析結果が格納されていますので、その中から必要とするブローブを選別する必要があります。面積値、縦横比などのパラメータを設定し「GetList()」を行うことで任意のブローブのみ取得できますので、下記のコード記述の様に、解析結果を選別するフィルタリング処理を行って下さい。

なお、本項では、面積値が 100 以上、4000 以下のブローブを選別するコードを記述します。

①イベントハンドラ(buttonBlob_Click)内の前項の(5)の行以降に、フィルタリング処理を追加します。下記の赤字部分を「Form1.cs」に追加してください。

///～以降にブローブ選別フィルタ処理コードを記述します～ …(5)

//ブローブ選別フィルタ作成

List<FVIL.Blob.CFviBlobFilterRange> filters = new List<FVIL.Blob.CFviBlobFilterRange>(); //…(7)

FVIL.Blob.CFviBlobFilterRange filter1 = new FVIL.Blob.CFviBlobFilterRange(); //…(8)

filter1.Type = FVIL.Blob.FeatureType.AREA; //面積値フィルタ設定…(9)

filter1.Max = 4000; //面積上限値：4000…(10)

filter1.Min = 100; //面積下限値：100…(11)

filters.Add(filter1); //フィルタに追加…(12)

//選別したブローブのみ取得

FVIL.Blob.CFviBlobList bloblist = blob.Result.GetBlobList(filters); //…(13)

//～ここに解析結果を表示するコードを記述します～ …(14)

(7) 2 値ブローブ解析結果フィルタ条件構造クラスを生成します。

「FVIL.Blob.CfviBlobFilterRange」は、解析結果をフィルタリングする際のフィルタ条件を保有するクラスです。

なお、フィルタは複数の条件を組み合わせる事を考慮して、List クラスとして生成します。

(8) リストに追加する各フィルタ条件を設定するクラスを生成します。

(9) フィルタ種別「Type」に、面積の「FVIL.Blob.FeatureType.AREA」を設定します。

(10) フィルタリング対象の特徴量の上限値「Max」に、「4000」を設定します。

(11) フィルタリング対象の特徴量の下限値「Min」に、「100」を設定します。

(12) パラメータを設定したフィルタ「filter1」を (7) で生成したフィルタのリストに追加します。

(13) 2 値ブローブ解析データリストを生成し、「blob.Result.GetBlobList」の引数に「filters」を設定して、選別したブローブのみを取得します。

(14) この行以降に、次項のオーバーレイへ描画コードを記述します。

6.4 結果の描画

- ①検出したデータを画面に表示するために「オーバーレイ」インスタンスを生成します。
その為、下記の赤字部分を参考にオーバーレイクラスを「Form1.cs」に追加してください。

```
public partial class Form1 : Form
{
    //FVIL変数宣言
    public FVIL.Data.CFviImage m_image = new FVIL.Data.CFviImage(); //画像メモリの生成
    public FVIL.Data.CFviImage m_imagebin = new FVIL.Data.CFviImage(); //2値画像用メモリの生成
    public FVIL.GDI.CFviOverlay m_overlay = new FVIL.GDI.CFviOverlay(); //オーバーレイの生成…(1)
}
```

- (1) 前項で生成した2値画像用画像メモリに続けて、オーバーレイクラスを生成します。

- ②イベントハンドラ(buttonBlob_Click)内の前項の(14)の行以降に、オーバーレイ描画処理を追加します。
下記の赤字部分を「Form1.cs」に追加してください。

```
//ここに解析結果を表示するコードを記述します～ …(14)
//解析結果の表示
m_overlay.Figures.Clear(); //オーバーレイのクリア…(2)
FVIL.Blob.CFviBlobData data = new FVIL.Blob.CFviBlobData(); //ブローブデータクラスの生成…(3)

for (int i = 0; i < bloblist.Count; i++) //…(4)
{
    //選別したブローブをコピーします。
    data.CopyFrom(bloblist[i]); //…(5)
    //表示用図形を作成します。
    // --- 点
    FVIL.GDI.CFviGdiPoint center = new FVIL.GDI.CFviGdiPoint(data.Center); //…(6)
    center.Size = new Size(5, 5); //…(7)
    center.Pen.Color = Color.FromArgb(255, 0, 0); //赤…(8)

    // --- 楕円
    FVIL.GDI.CFviGdiEllipse ellipse = new FVIL.GDI.CFviGdiEllipse(data.EquivalentEllipse); //…(9)
    ellipse.Pen.Color = System.Drawing.Color.Blue; //青…(10)

    // --- ブローブ番号
    FVIL.GDI.CFviGdiString number = new FVIL.GDI.CFviGdiString(data.BlobNo.ToString()); //…(11)
    number.Color = Color.Green; //緑…(12)
    number.Position = data.Center; //…(13)

    // オーバレイに図形を追加
    m_overlay.Figures.Add(center); //…(14)
    m_overlay.Figures.Add(ellipse); //…(15)
    m_overlay.Figures.Add(number); //…(16)
}
cFviImageView1.Display.Overlays.Add(m_overlay); //オーバーレイの追加…(17)
cFviImageView1.Refresh(); //ビューの更新…(18)
```

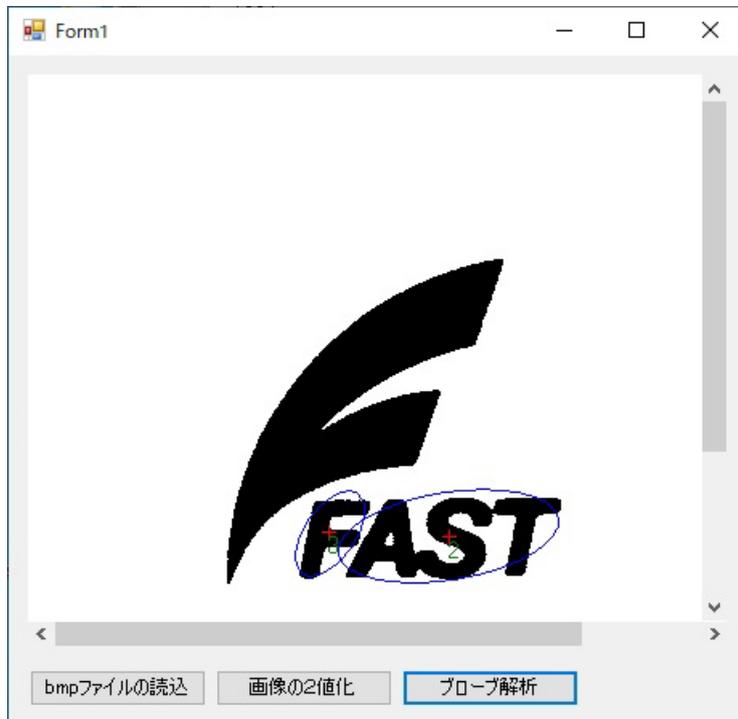
- (2) 「m_overlay.Figures.Clear()」にて、「m_overlay」内のデータをクリアします。
- (3) グローブデータクラスを生成します。
- (4) 選別したグローブ数分、繰り返します。
- (5) 選別したグローブ群を格納している「bloblist」内の指定したグローブデータを「data」にコピーします。
- (6) 重心描画用の点「center」を描画するデータ構造クラスを生成します。
- (7) 「center」のスタイルサイズの設定をします。
- (8) 「center」の描画色を赤色に設定します。
- (9) グローブの慣性等価楕円「ellipse」を描画するデータ構造クラスを生成します。
- (10) 「ellipse」の描画色を青色に設定します。
- (11) グローブ番号「number」の文字列を描画するデータ構造クラスを生成します。
- (12) 「number」の描画色を緑色に設定します。
- (13) 「number」の描画位置を設定します。
- (14) オーバレイに「center」を追加します。
- (15) オーバレイに「ellipse」を追加します。
- (16) オーバレイに「number」を追加します。
- (17) 「cFviImageView1」のオーバレイコレクションに追加します。
- (18) 「cFviImageView1.Refresh()」にて、表示を更新します。

③プログラムをビルドし、実行します。

④「bmp ファイルの読込」をクリックし、「fast0.BMP」をロードします。

画像ロード後、「画像の2値化」ボタンを押し、画像を2値化します。

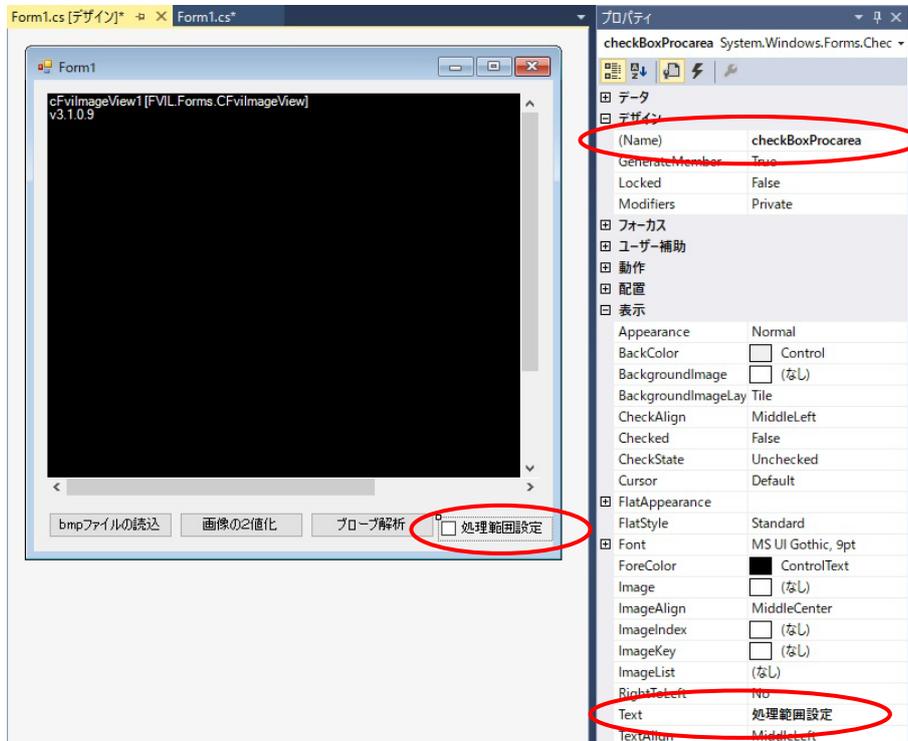
その後、「グローブ解析」ボタンを押し、[画面 18]の様に2値計測結果が表示されれば成功です。



[画面 18]

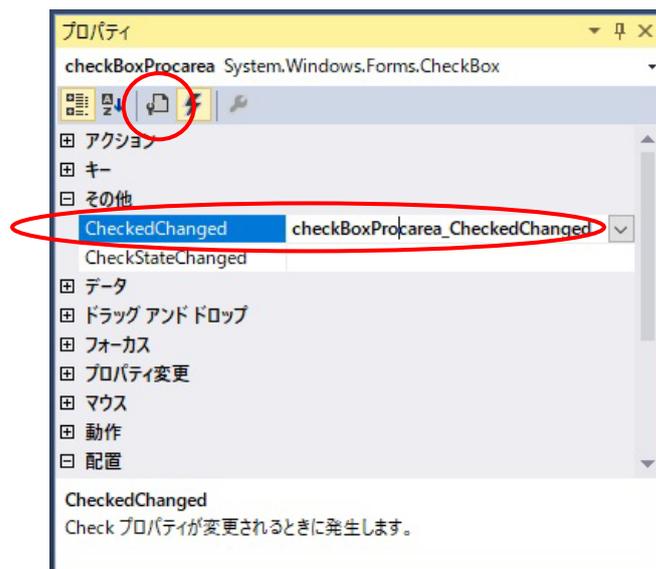
6.5 処理範囲の設定

- ① 「Form1.cs[デザイン]」にて、ツールボックスより、Form1 ダイアログへチェックボックスを追加します。
本書では、[画面 19]の様に追加したチェックボックスのプロパティの **Text** を「処理範囲設定」、**Name** を「checkBoxProcarea」とします。



[画面 19]

- ② チェックボックスのプロパティで、イベントボタン(赤丸)をクリックした後、CheckedChanged の右の空白部をダブルクリックしイベントハンドラ(checkBoxProcarea_CheckedChanged)を作成します。



[画面 20]

- ③ 作成したイベントハンドラ(checkBoxProcarea_CheckedChanged)内に、処理範囲の設定処理を

追加します。下記の赤字部分を「Form1.cs」に追加してください。

```
private void checkBoxProcarea_CheckedChanged( object sender, EventArgs e )
{
    try
    {
        if (checkBoxProcarea.Checked == true) //…(1)
        {
            //処理範囲の編集を許可する
            cFviImageView1.SetOverlayActive(FVIL.GDI.CFviDrawProcarea.OverlayID, true); //…(2)
        }
        else
        {
            //処理範囲の編集を不許可にする
            cFviImageView1.SetOverlayActive(FVIL.GDI.CFviDrawProcarea.OverlayID, false); //…(3)
        }
        cFviImageView1.Refresh(); //画像ビューの表示更新…(4)
    }
    catch(FVIL.CFviException ex)                //エラーコードの取得…(5)
    {
        MessageBox.Show(ex.Message, "Error");
    }
}
```

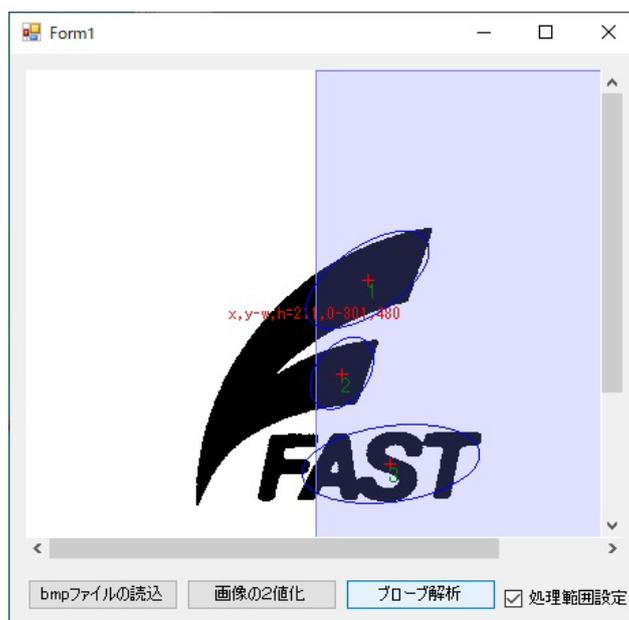
- (1)チェックボックスがチェックされているか判定します。
- (2) 処理範囲の編集を許可にする為に、SetOverlayActive にて、処理範囲のオーバーレイ ID である FVIL.GDI.CFviDrawProcarea.OverlayID を true にして有効にします。
- (3) 処理範囲の編集を不許可にする為に、SetOverlayActive にて、処理範囲のオーバーレイ ID である FVIL.GDI.CFviDrawProcarea.OverlayID を false にして無効にします。
- (4) 「cFviImageView1.Refresh()」にて、表示を更新します。
- (5) 「try～catch」にて、WIL のエラーコードを取得します。

④プログラムをビルドし、実行します。

⑤ 「bmp ファイルの読込」をクリックし、任意の Gray 画像をロードします。

画像ロード後、「画像の2値化」ボタンを押し、画像を2値化した後、「処理範囲設定」チェックボックスをチェックし、処理範囲の設定を行います。

下図のように処理範囲の設定を行った後、「プローブ検出」ボタンを押し、[画面 21]の様に処理範囲内の2値計測結果が表示されれば成功です。



[画面 21]

6.6 FVIL リファレンス

本項にて使用した WIL 関数の詳細説明に関しましては、WIL のヘルプファイル FVIL Reference の以下に記載がありますのでご参照下さい。

- CFviBinarize
FVIL リファレンス→FVIL.Conversion→CFviBinarize
- Blob
FVIL リファレンス→FVIL.Blob→Blob
- ObjectColor
FVIL リファレンス→FVIL.Blob→ObjectColor
- CFviBlobFilterRange
FVIL リファレンス→FVIL.Blob→CFviBlobFilterRange
- FeatureType
FVIL リファレンス→FVIL.Blob→FeatureType
- CfviBlobList
FVIL リファレンス→FVIL.Blob→CfviBlobList
- CFviOverlay
FVIL リファレンス→FVIL.GDI→CFviOverlay
- CFviBlobData
FVIL リファレンス→FVIL.Blob→CFviBlobData
- CFviGdiPoint
FVIL リファレンス→FVIL.GDI→CFviGdiPoint
- CFviGdiEllipse
FVIL リファレンス→FVIL.GDI→CFviGdiEllipse
- CFviGdiString
FVIL リファレンス→FVIL.GDI→CfviGdiString
- CFviDrawProcarea.OverlayID
FVIL リファレンス→FVIL.Forms→CFviImageView の中段にある、既定のオーバーレイ

6.7 サンプルプログラムの紹介

下記弊社 web にて、2 値ブローブ解析を行うサンプルプログラムを公開していますので、参考にしてください。

https://www.fast-corp.co.jp/software_dl/jp/supportj_sampledl3.php?sid=172&scid=72

7. 正規化関連サーチ(GS2)

本項では、画像処理手法の一つである正規化関連サーチ(グレイサーチ)の手順について解説します。以下の手順にて行います。

- ・ 画像ファイルの読み込み
- ・ パタンオブジェクトの生成、
- ・ 正規化関連サーチ(グレイサーチ)
- ・ 結果の表示

なお、画像ファイルの読み込みについては、「3.2 画像ファイルからの入力」を参照の上、事前にコードを追加してあるものとします。

その他、プロジェクトの作成、参照設定、画像ビューの追加、画像ビューの貼り付け、ライブラリの初期化処理に関しても、1~3項を参照して下さい。

7.1 パタンオブジェクトの生成

サーチを実行するために必要なマスターパタン(パタンオブジェクト)を生成します。

①サーチで使用するパタンオブジェクトを追加します。

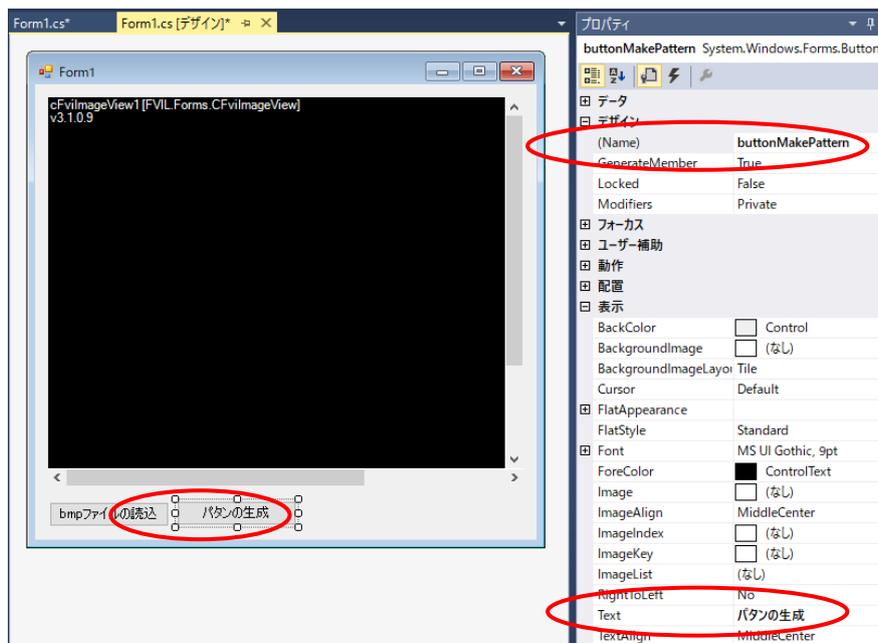
下記の赤字部分を参考にパタンオブジェクトを「Form1.cs」に追加してください。

```
public partial class Form1 : Form
{
    //FVIL変数宣言
    public FVIL.Data.CFviImage m_image = new FVIL.Data.CFviImage();           //画像メモリの生成
    //パタンオブジェクトの生成
    public FVIL.Data.CFviPattern m_pattern = new FVIL.Data.CFviPattern(); //…(1)
```

(1)前項で生成した画像メモリに続けて、パタンオブジェクトを生成します。

②前項の「bmp ファイルの読込」ボタンの時と同じ要領で、Form1 ダイアログへボタンコントロールを追加します。

本書では、[画面 22]の様に追加したボタンコントロールプロパティのTextを「パタンの生成」、Nameを「buttonMakePattern」とします。



[画面 22]

- ③ 「パタンの生成」 ボタンをダブルクリックし、イベントハンドラ(buttonMakePattern_Click)を作成します。
- ④ イベントハンドラ(buttonMakePattern_Click)内に、ブローブ解析処理を追加します。
下記の赤字部分を「Form1.cs」に追加してください。

```
private void buttonMakePattern_Click( object sender, EventArgs e )
{
    //パタンオブジェクト生成用の引数作成
    FVIL.Data.CFviRectangle rect = new FVIL.Data.CFviRectangle(168, 265, 335, 320); //…(2)
    FVIL.Data.CFviPoint point = new FVIL.Data.CFviPoint(168, 265); //…(3)

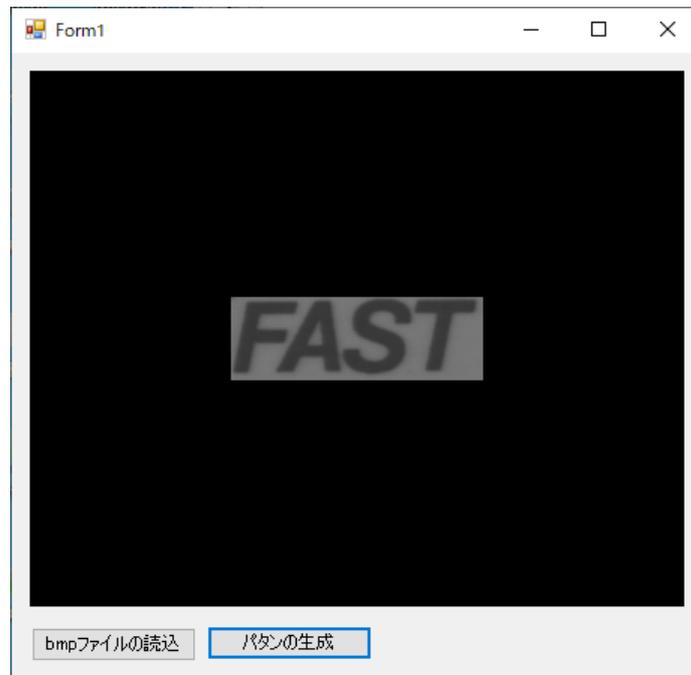
    try
    {
        m_pattern.MakePattern(m_image, rect, point); //パタンオブジェクト生成…(4)

        cFviImageView1.Image = m_pattern; //画像ビューに表示する画像の設定…(5)
        cFviImageView1.Refresh(); //画像ビューの表示更新…(6)
    }
    catch (FVIL.CFviException ex) //エラーコードの取得…(7)
    {
        MessageBox.Show(ex.Message, "Error");
    }
}
```

- (2) パタンオブジェクト生成用の引数である登録位置の矩形を作成します。
- (3) パタンオブジェクト生成用の引数である基準点を作成します。
- (4) 引数の矩形領域からパタンオブジェクトを生成します。
- (5) 「cFviImageView1」の描画対象画像オブジェクトに「m_pattern」を設定します。

- (6) 「cFviImageView1.Refresh()」にて、表示を更新します。
- (7) 「try～catch」にて、WIL のエラーコードを取得します。

- ⑤ プログラムをビルドし、実行します。
- ⑥ 「bmp ファイルの読込」をクリックし、「fast0.BMP」をロードします。
画像ロード後、「パタンの生成」ボタンを押し、[画面 23]の様に作成されたパタン画像が表示されれば成功です。



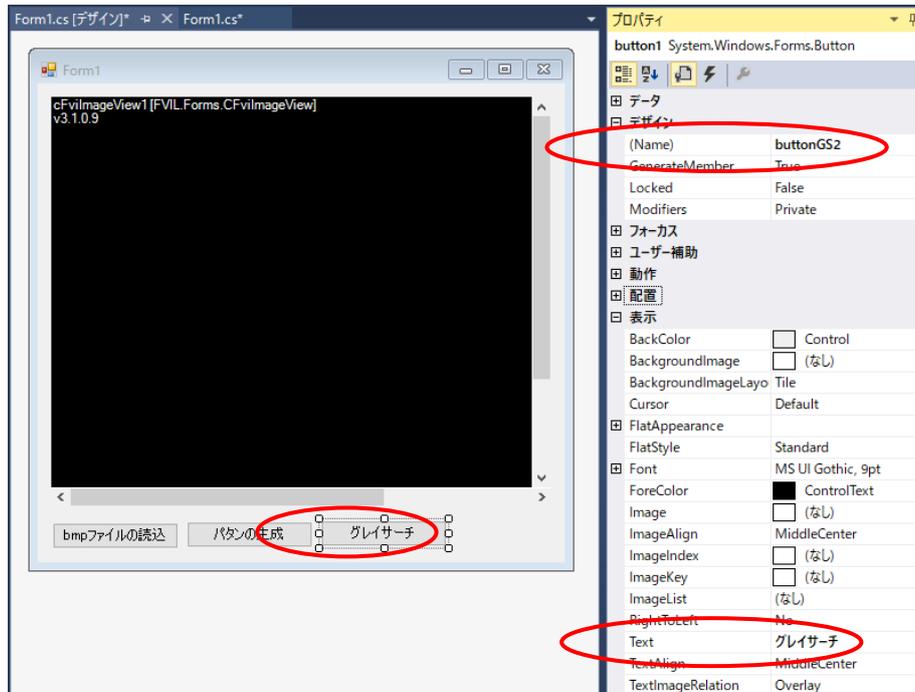
[画面 23]

7.2 正規化関連サーチ(GS2)

サーチを実行するコードを追加しましょう。

- ①前項の「パタンの生成」ボタンの時と同じ要領で、Form1 ダイアログへボタンコントロールを追加します。

本書では、[画面 24]の様に追加したボタンコントロールプロパティの Text を「グレイサーチ」、Name を「buttonGS2」とします。



[画面 24]

- ②「グレイサーチ」ボタンをダブルクリックし、イベントハンドラ(buttonGS2_Click)を作成します。
- ③イベントハンドラ(buttonGS2_Click)内に、グレイサーチ処理を追加します。下記の赤字部分を「Form1.cs」に追加してください。

```

private void buttonGS2_Click( object sender, EventArgs e )
{
    //グレイサーチクラス生成
    FVIL.GS2.CFviGS2 gs2 = new FVIL.GS2.CFviGS2(); //…(1)
    //グレイサーチパターンオブジェクト生成
    FVIL.GS2.CFviGS2Pattern gs2pattern = new FVIL.GS2.CFviGS2Pattern(); //…(2)

    try
    {
        //処理対象画像チェック
        if (FVIL.ErrorCode._SUCCESS != FVIL.GS2.CFviGS2.CheckValidity(m_image)) //…(3)
        {
            MessageBox.Show("処理対象画像が不正です", "Error");
            return;
        }

        gs2.Open(); //グレイサーチオブジェクト生成…(4)
        gs2.SrcImages[0] = m_image; //入力画像を設定…(5)
        gs2.Pattern = gs2pattern; //パターン画像を設定…(6)
        gs2pattern.Create(m_pattern, FVIL.GS2.Filter.Smooth); //パターンオブジェクト生成…(7)
        //パターン画像チェック
        if (FVIL.ErrorCode._SUCCESS != FVIL.GS2.CFviGS2.CheckValidity(m_pattern)) //…(8)
        {
            MessageBox.Show("パターン画像が不正です", "Error");
            return;
        }

        gs2.Execute(); //グレイサーチ実行…(9)

        //～以降に結果をオーバーレイへ描画するコードを記述します～…(10)
    }
    catch (FVIL.CFviException ex) //エラーコードの取得…(11)
    {
        MessageBox.Show(ex.Message, "Error");
    }
}

```

- (1) グレイサーチを実行するためのクラスを生成します。
- (2) グレイサーチパターンオブジェクトを生成するためのクラスを生成します。
- (3) 設定した入力画像が処理可能か否か「CheckValidity()」にて判定します。
処理可能な条件については、グレイサーチクラスの説明をご参照ください。
- (4) グレイサーチオブジェクトを生成します。
- (5) グレイサーチ処理を行う画像メモリに「m_image」を設定します。
- (6) グレイサーチ処理のパターンオブジェクトに「gs2pattern」を設定します。
- (7) パターンオブジェクトからグレイサーチパターンオブジェクトを生成します。
- (8) 設定したパターン画像が処理可能か否か「CheckValidity()」にて判定します。

- (9) 「Execute()」にて、グレイサーチ処理を実行します。
結果は「FVIL.GS2.CFviGS2Result()」に格納されます。
- (10) この行以降に、次項の結果をオーバーレイへ描画するコードを記述します。
- (11) 「try~catch」にて、WILのエラーコードを取得します。

7.3 結果の描画

サーチ結果を描画するコードを追加しましょう。

- ①結果を画面に表示するために「オーバーレイ」インスタンスを生成します。
その為、下記の赤字部分を参考にオーバーレイクラスを「Form1.cs」に追加してください。

```
public partial class Form1 : Form
{
    //FVIL変数宣言
    public FVIL.Data.CFviImage m_image = new FVIL.Data.CFviImage();           //画像メモリの生成
    //パタンオブジェクトの生成
    public FVIL.Data.CFviPattern m_pattern = new FVIL.Data.CFviPattern();
    //オーバーレイの生成
    public FVIL.GDI.CFviOverlay m_overlay = new FVIL.GDI.CFviOverlay();     //…(1)
}
```

- (1) 前項で生成したパタンオブジェクトに続けて、オーバーレイクラスを生成します。
- ②イベントハンドラ(buttonGS2_Click)内の前項の(10)の行以降に、オーバーレイ描画処理を追加します。
下記の赤字部分を「Form1.cs」に追加してください。

```

//～以降に結果をオーバーレイへ描画するコードを記述します～…(10)

//解析結果の表示
m_overlay.Figures.Clear(); //オーバーレイのクリア…(2)
for (int i = 0; i < gs2.Result.Count; i++) //…(3)
{
    //表示用図形を作成します。
    // ---矩形
    FVIL.GDI.CFviGdiRectangle rect = new FVIL.GDI.CFviGdiRectangle(m_pattern.RegistRect); //…
(4)
    rect.Pen.Color = System.Drawing.Color.Blue; //青…(5)

    // ---相関値(スコア)
    FVIL.GDI.CFviGdiString score = new FVIL.GDI.CFviGdiString(); //…(6)
    score.Text = gs2.Result[i].score.ToString(); //…(7)
    score.Position = rect.St; //…(8)
    score.Color = System.Drawing.Color.Yellow; //黄…(9)

    //オーバーレイに図形を追加
    m_overlay.Figures.Add(rect); //…(10)
    m_overlay.Figures.Add(score); //…(11)
}
cFviImageView1.Image = m_image; //画像ビューに表示する画像の設定…(12)
cFviImageView1.Display.Overlays.Add(m_overlay); //オーバーレイの追加…(13)
cFviImageView1.Refresh(); //ビューの更新…(14)

```

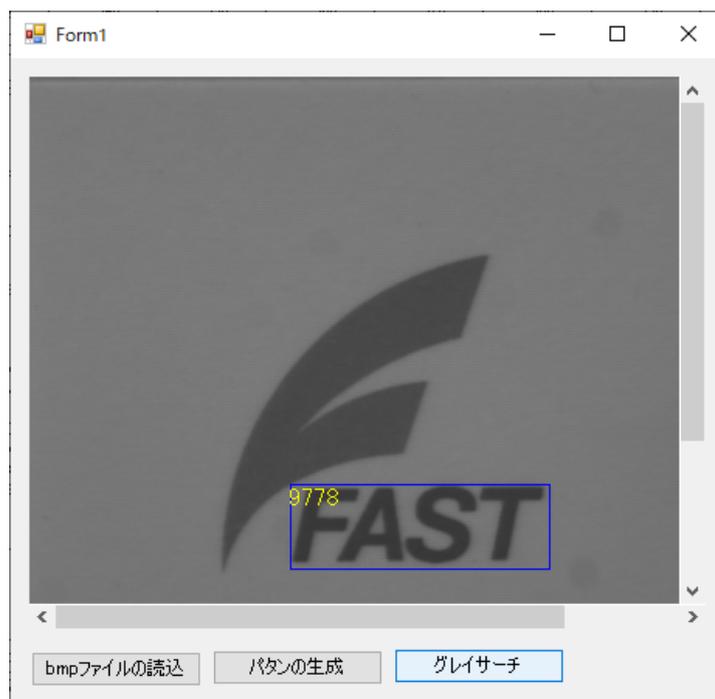
- (2) 「m_overlay.Figures.Clear()」にて、「m_overlay」内のデータをクリアします。
- (3) グレイサーチ結果の要素数分、繰り返します。
- (4) パターンオブジェクト「m_pattern」から、矩形「rect」を描画するデータ構造クラスを生成します。
- (5) 「rect」の描画色を青色に設定します。
- (6) スコア描画用の文字列「score」を描画するデータ構造クラスを生成します。
- (7) 「score」の文字列に「gs2.Result[i].score」を設定します。
- (8) 「score」の描画座標に「rect.St」を設定します。
- (9) 「score」の描画色を黄色に設定します。
- (10) オーバレイに「rect」を追加します。
- (11) オーバレイに「score」を追加します。
- (12) 「cFviImageView1」の描画対象画像オブジェクトに「m_image」を設定します。
- (13) 「cFviImageView1」のオーバーレイコレクションに追加します。
- (14) 「cFviImageView1.Refresh()」にて、表示を更新します。

③プログラムをビルドし、実行します。

④「bmp ファイルの読込」をクリックし、「fast0.BMP」をロードします。

画像ロード後、「パタンの生成」ボタンを押し、パターンを生成します。

その後、「グレイサーチ」ボタンを押し、[画面 25]の様にグレイサーチの結果が表示されれば成功です。



[画面 25]

7.4 FVIL リファレンス

本項にて使用した WIL 関数の詳細説明に関しましては、WIL のヘルプファイル FVIL Reference の以下に記載がありますのでご参照下さい。

- CFviPattern
FVIL リファレンス→FVIL. Data→CFviPattern
- CFviRectangle
FVIL リファレンス→FVIL. Data→CFviRectangle
- CFviPoint
FVIL リファレンス→FVIL. Data→CFviPoint
- CFviGS2
FVIL リファレンス→FVIL.GS2→CFviGS2
- CFviGS2Pattern
FVIL リファレンス→FVIL.GS2→CFviGS2Pattern
- CFviGdiRectangle
FVIL リファレンス→FVIL.GDI→CFviGdiRectangle

7.5 サンプルプログラムの紹介

下記弊社 web にて、正規化関連サーチ(GS2)を行うサンプルプログラムを公開していますので、参考にしてください。

https://www.fast-corp.co.jp/software_dl/jp/supportj_sampledl3.php?sid=172&scid=71

WIL プログラマーズガイド
プログラム作成 C#編

2021年3月 第1版 発行

発行所 株式会社ファースト

本 社 〒242-0001 神奈川県大和市下鶴間 2791-5

ユーザ・サポート E-mail : support@fast-corp.co.jp
