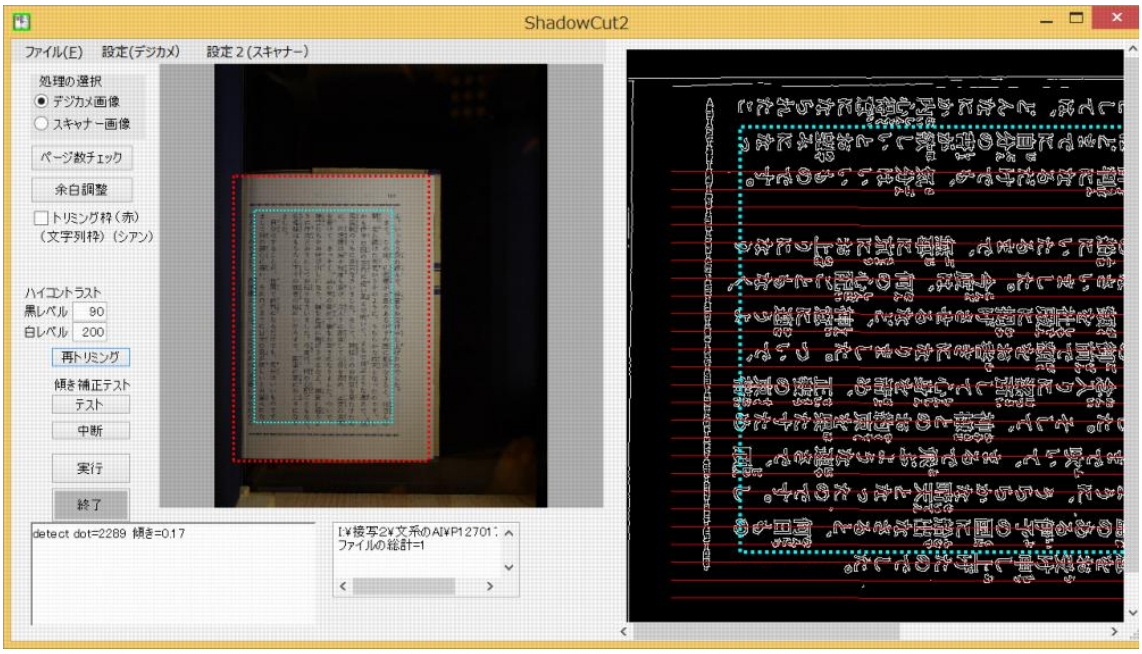
説明書
使用手順
照明ムラの補正処理

当方が日常操作するときの操作手順を述べます。対象の本としては活字本を主に考えています。画像ファイルを操作するときは、画像ビュアーXnView を用いています。処理したい画像の中から一つのファイルをドラッグアンドドローして読み込ませます。上部の設定（デジカメ）のメニューをクリックします。傾き補正をチェック、縦書きの場合、90 度回転をチェックします。閉じボタンをクリックします。マウスをドローしてトリミング範囲を決めます。トリミング枠はレッド線です。トリミング枠のチェックを外します。文字列範囲を適当に決めます。文字列枠はシアン線です。傾き補正テストのテストボタンをクリックします。右のウィンドウに傾き補正の結果が表示されます。傾き補正が適正なら次に実行ボタンをクリックします。デフォルトではハイコントラスト画像が右ウインドウに表示されます。文字が見やすいように、黒レベル、白レベルを調整します。再トリミングボタンをクリックすると処理前の画像表示に戻ります。条件が良いとなりましたら、XnView のファイルリスト表示から必要な個数のファイルを選択し、ドラッグアンドドローして読み込ませます。トリミング枠、文字列枠は改めて設定する必要はありません。実行ボタンをクリックします。挿絵等が存在する場合、ブロックサイズを大きくすると良好な結果が得られることがあります。下地、背景の明るさが取得できにくいときは、参照画像との比較、補正を選択します。例を図に示します。ShadowCut2.exe が存在するディレクトリーにある temp5.png が、参照画像ファイル（活字ページ、無地ページ）から生成される明るさ補正の基準となるファイルです。



トリミング範囲、文章範囲(この範囲を基に傾き補正)を指定

ハイコントラスト処理後の画像

参照画像読込で比較補正。明るさ画像を生成し、それを基に明るさ補正を行った場合の例。

写真処理でのシェーディング補正と類似する処理です。

設定２（スキャナー）を用いる場合

スキャナーの場合、挿絵に有利なように、ブロックサイズの ydot は画像の縦サイズに等しく、xdot は５に固定しています。トリミング、傾き補正は無効です。明るさの均一化が不十分な場合、初めにデジカメ画像処理した後、スキャナー画像処理で２分割処理のみを行う方が、良好な結果が得られると思います。

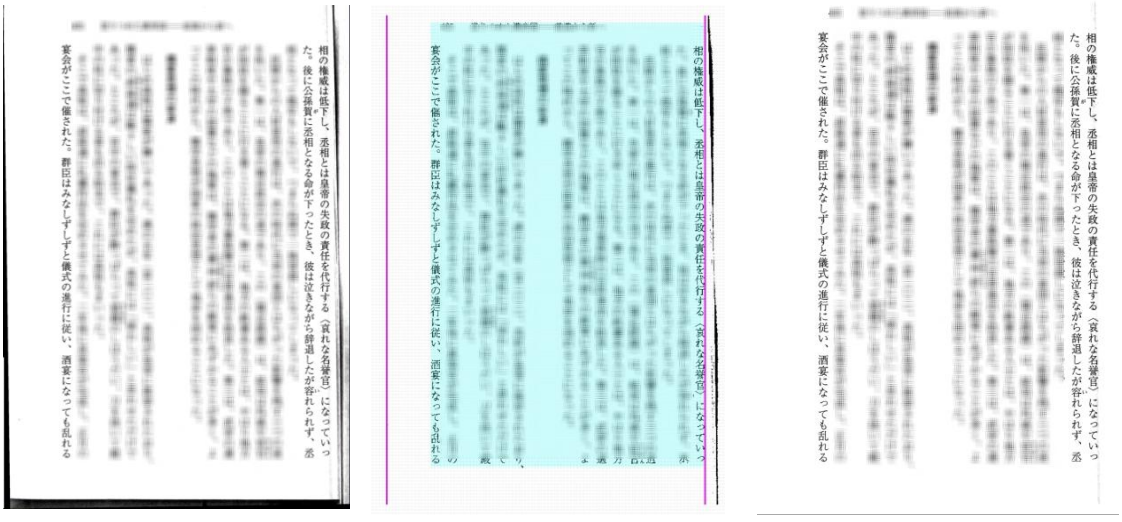
ページ数チック

これは後で付け加えた機能です。以前はページ数のチェックは XnView の閲覧画面で確認を行っていました。本によってはページ数の表示が小さくて見にくいことがたまにありました。拡大すると表示位置がずれて読めなくなる等、何かいい方法はないかと検討した結果、本機能を追加しました。手順は以下の通りです。XnView のファイルリスト表示から必要な個数のファイルを選択し、ドラッグアンドドローして読み込ませます。ページ数チェックボタンをクリックすると、新しいウィンドウが開きます。開始ページ欄に例えば 251 又は 0、増加頁欄に－2 又は 2 を記入します。開始（最終）ページはふつう印刷されていないので、数える必要があります。Start ボタンをクリックします。最終ページおよび推定ページが表示されます。次ページボタンを何回かクリックし、ページ数が表示されたのち、読みやすい位置になるように画像を左右上下に移動させます。デジカメの場合、奇数頁、偶数頁いずれかが連続するので表示位置記憶のメモ１，メモ２を共にクリックします。スキャナーの場合、左右で表示位置が異なるので、メモ１，メモ２を区別して記憶させます。画像上のページ数と推定頁が一致していることを確認しておきます。一致しない場合、開始ページまたは増加頁が誤っています。修正後、再度 Start ボタンをクリックします。次ページボタンクリックし、推定頁と画像が一致していることを２，３頁、確認します。頁+10 ボタンをクリックし、推定値と一致しているか確認します。頁+10 ボタンを繰り返しクリックします。一致しないページが現れた場合、前ページボタンをクリックし、一致するページまで戻ります。頁欠落または頁重複ボタンをクリックして推定ページと表示ページが一致するようにします。不都合が起きた位置のファイル名をメモするか、ファイル名保存ボタンをクリックします。ファイルは画像ファイルのディレクトリーに保存されます。頁+10 ボタンを用いるときの注意点として、運悪く 10 ページの範囲内に欠落と重複が共存していた場合、不都合ファイルを見逃すことになります。撮影時の欠落は認識できませんが、重複が発生したことは認識できるのでメモして置けば、欠落と重複の共存を見過ごしは避けることが出来ると思います。

余白調整

この処理も後で追加した機能なので、ハイコントラスト処理を行った後、変換ファイルフォルダーにある処理済みファイルを再度、ドラッグアンドドローして読み込む必要があります。この処理は、余白を削除するものではなく、余白にあるゴミを除去します。ただ、

一つのゴミしか除去できません。ゴミのない画像を処理すると文字列を除去してしまいます。余白幅が広い場合、ゴミは ChainLP、Ralpha などを用いてトリミングするのが確実と思います。余白調整は、トリミングが難しいとき補助的に利用すると良いと思います。例を図に示します。処理工程は次のような手順で行っています。画像端を白になるようにしておき、横方向から内部に向かってラインスキャンを行い、ライン上に３ドット以上の黒を検出し、かつ横方向に２ドット以上連続していると、ゴミの領域に入ったと判断します。ライン上に３ドット以上の白を検出し、かつ横方向に３ドット以上連続していると、ゴミ領域を通過して空白の領域に入ったと判断します。次に黒い領域（文字列領域）に入る位置までを白で塗りつぶします。認識のドット数等を変更すると多少動作が変化します。この余白調整は誤動作をすることがあるので、後で必ず目視する必要があります。目視用画像として、白色化する位置を示す赤ラインおよびブルーの塗り潰し領域を表示するファイルを生成します。ゴミが適切に削除されたかどうかを確認するために、ブルー領域を表示しています。ゴミが２つある場合の対処としては、左右の白色幅を大きくする、白と認識する最小の幅を大きくするなど。または余白調整処理を２回行う。この処理は周辺を白く塗りつぶすだけなので、画像の劣化は原理的に無いと思います。ゴミを除去する目的は、ChainLP 等で自動余白削除を行う場合、ゴミがあると誤動作をする可能性があるためです。ScanTailor では文章領域の認識機能が高いので、事前のゴミ取りの必要はありません。Scan Tailor は非常に高性能なソフトですが、たまに「版面を選択」のプロセスで領域の認識が不適切なことがあります。挿絵があるときに誤動作が起こりやすいように思いますが、領域認識の能力は非常に高いと思います。どういう手法を用いているのか興味があります。OpenCV を用いれば簡単にできるものなのか試してみたいと思っています。



右端のゴミを除去した例

Canny フィルター

画像をドラッグアンドドローして読み込ませ、設定（デジカメ）内の Cannyfilter 調整ボタンをクリックすると Canny フィルター像が表示されます。

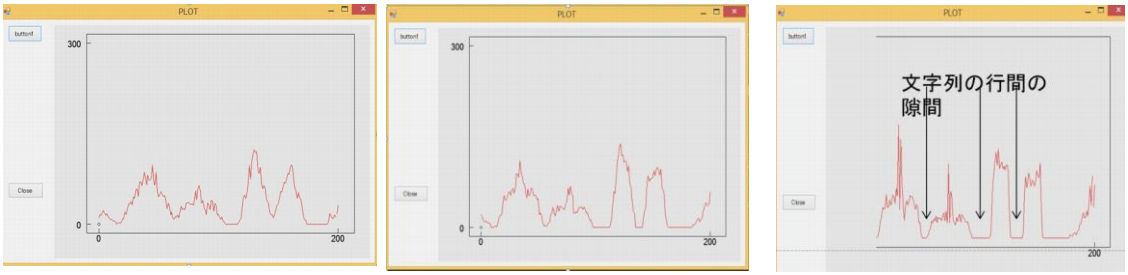
プログラムの開発に用いた方法

傾き補正について

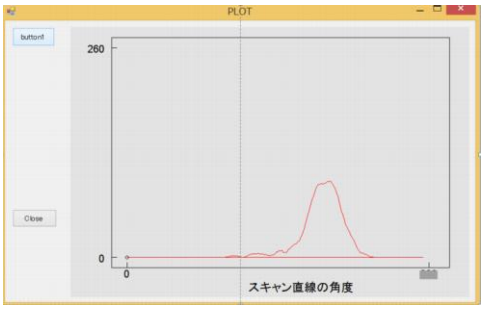
ChainLP、Scan Tailor には優秀な自動傾き補正があり、どういう原理で傾き補正ができるのか知りたかったので、web で色々調べ、プログラムを検討しました。

「文章画像のレイアウト」(http://tdl.libra.titech.ac.jp/hkshi/xc/contents/pdf/116685485/6)が基礎知識として非常に参考になりました。研究者が解説している最新の方法は、基礎知識がないのでよく理解できませんでした。ハフ変換を利用した解説記事がいくつかあったので、当方も試してみました。ShadowCut2 の中でもスキャナー画像の２分割にハフ変換を利用しています。ハフ変換の原理については web 上に多数の解説があり、原理は十分に理解することができたと思ったのですが、コーディングをしようとすると、どういった流れでプログラム書けばよいか、一日考えても全くアイデアが浮かびませんでした。自作は諦めて、「図解入門よくわかる最新画像処理アルゴリズムの基本と仕組み／長尾 智晴、P.249」にある C 言語で記述されたハフ変換ソースを C#に移植しました。傾き補正にハフ変換を利用する方法は、原理としては非常にシンプルで良いと思いましたが、実際にテストをすると精度に少し不満がありました。ハフ変換で文書画像内の直線を検出させると、バラつい

た傾きの直線が幾つか検出されました。画像にあるのは直線ではなく、文字列の集合体なので検出された直線の傾きがバラつくのは仕方ないと思われます。ハフ変換の分割角度を小さくしても、傾きにバラつきがあるため、精度が上がらないと思われました。また、何らかの画像前処理をするとハフ変換をもっと有効に利用できるかもしれません。別の方法として、-5～5 度まで傾きを変化させた直線で上から下に Canny 画像をラインスキャンしたときの白ドットの数を計測し、その分布を検討しました。図にはスキャン直線の傾きを変化させたときの白ドットの検出個数を示しています。縦軸は白ドットの検出個数、横軸はスキャン直線の移動量です。図 1,2,3 を検討した結果、文字列の角度とスキャン直線が平行になったとき、文字列間の空白の隙間が大きくなっていることがわかります。文字列間の空白の隙間を最大にする角度を求めることにより傾き角度を知ることが可能でした。ただ、文字列以外にイラスト等の挿絵が含まれている場合、文字列間の空白を計測できないことがわかりました。文字列の行間の空白測定は、挿絵等の外乱因子に対して影響を受けやすいのが欠点と思われます。別の方法として、白ドットの検出個数の分布曲線の形を比較すると、スキャン線が文字列と平行になったときブロードな分布曲線がシャープなものに変化することが認められました。在職中は無機化合物の合成実験をよく行っていましたが、生成物の結晶子サイズを評価するとき、粉末 X 線回折のピーク波形がブロードかシャープかによりその評価を行っていました。そのときは半価幅（ピーク高さの半分の位置での波形幅）という方法を用いました。検出個数の分布曲線の形がいびつな形をしていたので、今回は別な方法で分布曲線のブロード、シャープの度合いを評価しました。簡単な方法として、分布曲線の微分の絶対値の積算値を用いることにしました。ただ、分布曲線には細かいピークが含まれているため、そのままの積算値はシャープの度合いの評価値としては不適切でした。分布曲線を平滑化した後の積算値は、文字列とスキャン線の傾きの差に相関性が認められました。図に示すように、文字列とスキャン線が平行になるとき、積算値は極大を示しました。この方法は挿絵等の外乱に対してあまり影響を受けないことが観察されました。ShadowCut2 ではこのスキャン法を用いています。



縦軸はスキャンライン上に検出されたドット数、横軸は上から下へのスキャンラインの移動量。左図は傾きの差が大きい。右図は文字列とスキャンラインの傾きが等しい。



縦軸は検出ドット曲線の微分の絶対値を積算した値。横軸はスキャンラインの傾き。文字列の傾きとスキャンラインの傾きが等しいとき極大。

スキャナー画像の自動分割について

書籍の Canny フィルター画像にハフ変換処理を行うと、配列:theta_rho[theta,rho]に投票数が格納されます。この配列データを元に逆算することにより、任意の一組の x,y,theta の値に対応する投票数を一意的に求めることができます。投票数=メソッド (x,y,theta)とするメソッドを記述します。(x,y)座標における 2 分割線の角度範囲を 85 から 95 度とし、その角度範囲での最大投票数を返すメソッド(x,y)を用意します。画像の作業サイズは、1000x1000 としているので、y=500 と固定して x=480 から 520 まで変化させたときの投票数を調べ、最大投票数を返す x 値を分割位置としました。投票数が 50 以下の場合、直線が見つかりませんとメッセージを表示し、x=500 としています。

Canny フィルターについて

ソルベーフィルター、ラプラスフィルター等を色々検討したところ、Canny フィルターがもっとも高性能と思われました。ただ、Canny フィルターを活用するとなると、OpenCV を使う方法しか見つけることができませんでした。 OpenCV は使った経験がありませんし、c#ではラッパーを使わないといけないとかハードルがかなり高いので、今回は他の方法を検討しました。そこでさらに調べると http://d8yd.blog105.fc2.com/?no=84 のじゅんじ氏という方が Canny フィルターのアルゴリズム、ソースを公開されていました。ソースは http://sakef.jp/blog/2011/01/canny01/ に公開されていたのですが、現在はリンク切れになっています。公開ソースは Java、Ruby などの言語と思われますが、当方には未知の言語で書かれていました。 ただ、ソースの構文自体は C#に非常に類似していたので C#に書き換えることが可能でした。オリジナルソース（抜粋）以下のようなものでした。

```
package
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.Sprite;
```

```
import flash.display.StageAlign;
import flash.display.StageQuality;
import flash.display.StageScaleMode;
import flash.events.Event;
import flash.filters.ColorMatrixFilter;
import flash.filters.ConvolutionFilter;
import flash.geom.Point;
import flash.geom.Rectangle;
import flash.media.Camera;
import flash.media.Video;

public class Main extends Sprite
{
    private static const COLOR_MATRIX:ColorMatrixFilter = new ColorMatrixFilter([0.3,0.59,0.11,0,0,0.3,0.59,0.11,0,0,0.3, 0.59, 0.11,0,0,0,0,0,1,0]);
    private static const COVLUTION:ConvolutionFilter = new ConvolutionFilter(5,5,[2,4,5,4,2,4,9,12,9,4,5,12,15,12,5,4,9,12,9,4,2,4,5,4,2],159);
    private static const SOBEL_X:ConvolutionFilter = new ConvolutionFilter(3,3,[-1,0,1,-2,0,2,-1,0,1]);
    private static const SOBEL_Y:ConvolutionFilter = new ConvolutionFilter(3,3,[1,2,1,0,0,0,-1,-2,-1]);
    private static const W:int = 320;
    private static const H:int = 240;
    private static const TH:int = 60;
    private static const TL:int = 30;
    private static const RECT:Rectangle = new Rectangle(0,0,W,H);
    private static const POINT:Point = new Point();
    private var source:BitmapData;
    private var canvas:BitmapData;
    private var dx:BitmapData;
    private var dy:BitmapData;
    private var video:Video;
    private var buffPower:Vector.<int>;
    private var buffTheta:Vector.<int>;
    private var buffEdge:Vector.<int>;
    private var buffTh:Vector.<int>;
```

```
private var buffTl:Vector.<int>;

public function Main()
{
    stage.scaleMode = StageScaleMode.NO_SCALE;
    stage.align = StageAlign.TOP_LEFT;
    stage.quality = StageQuality.LOW;
    stage.frameRate = 10;

    var camera:Camera = Camera.getCamera();
    if(camera)
    {
        camera.setMode(W, H, 20);
        video = new Video(W, H) as Video;
        video.attachCamera(camera);

        source = new BitmapData(W, H);
        canvas = source.clone();
        dx   = source.clone();
        dy   = source.clone();

        var bmp1:Bitmap = addChild(new Bitmap(source)) as Bitmap;
        var bmp2:Bitmap = addChild(new Bitmap(canvas)) as Bitmap;
        bmp1.x = 40;
        bmp2.x = W + 50;
        bmp1.y = bmp2.y = 50;

        // 一時データを保存する Vector
        buffPower = new Vector.<int>(W*H, true);
        buffTheta = new Vector.<int>(W*H, true);
        buffEdge = new Vector.<int>(W*H, true);
        buffTh=new Vector.<int>(W*H,true);
        buffTl=new Vector.<int>(W*H,true);

        // イベント追加
        addEventListener(Event.ENTER_FRAME, onFrame);
```

```
        }
    }

    private function onFrame(e:Event):void
    {
        var i:int, j:int, n:int, theta:int, cx:int, cy:int, s0:int, s1:int, s2:int, edge:int,
tw:int, th:int;
        source.lock();
        canvas.lock();
        source.draw(video);

        // グレスケ化
        canvas.applyFilter(source, source.rect, POINT, COLOR_MATRIX);

        // ノイズ除去
        canvas.applyFilter(canvas, canvas.rect, POINT, COVLUTION);

        // sobel
        dx.applyFilter(canvas, RECT, POINT, SOBEL_X);
        dy.applyFilter(canvas, RECT, POINT, SOBEL_Y);
```

先頭からこの行までのソースは当方には解読不能であり、特に重要と思われなかったので、移植には含まれていません。メインプログラムに入る前の準備のための記述と思われます。

ここより以下の行を C#へ移植しました。

```
        // 勾配と強さを計算
        for(j=0 ; j<H ; j++)
        {
            for(i=0 ; i<W ; i++)
            {
                cx = int((dx.getPixel(i,j))&0xff);
                cy = int((dy.getPixel(i,j))&0xff);
                theta = (Math.atan2(cy, cx)*180/Math.PI)>>0;
                if(theta < 0) theta += 180;
                n = j*W + i;
```

```
            buffPower[n] = Math.sqrt(cx*cx + cy*cy)>>0;
            buffTheta[n] = theta;
        }
    }

    // エッジの線細化
    tw = W-1;
    th = H-1;
    for(j=1 ; j<th ; j++)
    {
        for(i=1 ; i<tw ; i++)
        {
            n = j*W + i;
            theta = buffTheta[n];
            s1 = buffPower[n];
            s0 = s2 = 0;

            if((theta >= 0 && theta <= 22) || (theta >= 157 && theta <= 180))
            {
                s0 = buffPower[n-1];
                s2 = buffPower[n+1];
            }
            else if((theta >= 22 && theta <= 67) )
            {
                s0 = buffPower[n-1+W];
                s2 = buffPower[n+1-W];
            }
            else if((theta >= 67 && theta <= 112))
            {
                s0 = buffPower[n+W];
                s2 = buffPower[n-W];
            }
            else if((theta >= 112 && theta <= 157))
            {
                s0 = buffPower[n+W+1];
                s2 = buffPower[n-W-1];
```

```
            }

            if(s1 > s0 && s1 > s2) buffEdge[n] = s1;
            else buffEdge[n] = 0;
        }
    }

    // 閾値処理
    for(j=1 ; j<th ; j++)
    {
        for(i=1 ; i<tw ; i++)
        {
            n = j*W + i;
            edge = buffEdge[n]>>0;
            buffTh[n] = (TH <= edge)?(255):(0);
            buffTl[n] = (TL <= edge)?(255):(0);
        }
    }
```

オリジナルソースではフィルタ類はライブラリーを使用していたので、これらの代替モジュールは新たに作成しました。後半の if 文の部分を改変していますが、他はオリジナルソースを概ねそのまま移植しました。if 文の変更点は閾値像の連結を 1 ピクセル拡張したことです。このプログラムの Canny フィルターとして完成度どの程度なのか素人なので判断できません。原著者は「アルゴリズムに沿って作ってるので結構重い。FPS が 10 くらい。一応 Canny っぽいのが出来た・・・気がする。間違いに気付いた方がいたら教えてください 」と述べられています。処理画像を見る限り、Canny フィルターとして概ね機能していると思っています。原著者には感謝です。ソーベルフィルタについては、「こるなごⅢ氏(http://csharpimage.blog60.fc2.com/blog-entry-5.html)」のプログラムソースを利用させていただきました。Canny フィルターの C#ソースを見つけるのはなかなか難しいと思われたのでソースを添付しています。Canny フィルターを活用する実行型のフリーアプリは見つけることができませんでした。ShadowCut2 は Canny フィルター像の保存が可能なので、他のフィルター像との比較検討に使えると思います。Canny フィルターを活用するだけなら、プログラムを移植するより OpenCV の使用法を習得することに時間を掛ける方が、時間的にも、画像処理のスキルアップの観点からも、はるかに有利と思います。ただ、当方は趣味で暇つぶしにプログラムを書いており、画像処理の細かい処理方法がどうなっているのかを知りたいため、あえてフィルター関連のモジュール作成を色々と試行しました。

その他

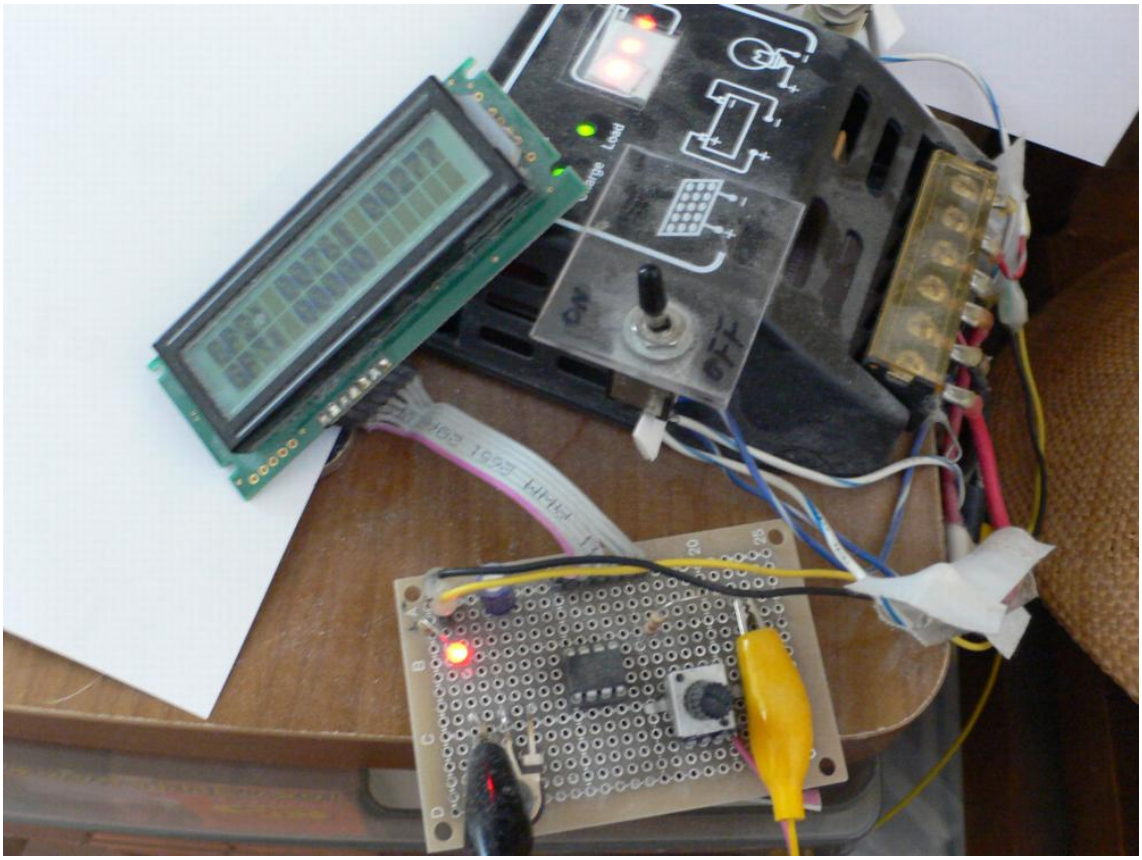
前作の ShadowCut では最大画素サイズは 10000x10000 でしたが、ShadowCut2 では 5000x5000 になっています。Delphi4 では 10000x10000 で何ら問題なかったのですが、C# では 10000x10000 にすると、GC.Collect()でメモリーを解放していてもメモリーが不足しています等のエラーが発生し、動作が不安定になりました。開発パソコンが 32bitOS であることに原因しているのか、プログラムの配列宣言の仕方に問題があるのかよくわかりません。デジタルカメラは 5000 ドットを超えるものがありますが、その場合、事前にトリミングしてサイズを小さくする必要があります。前作の ShadowCut のソースの行数は 1500 行ほどでしたが、ShadowCut2 では約 6000 行ほどになりました。前作とは使用言語が異なるので、初めから新しく書き直しました。全体、モジュールについてのメモ類を記録していなくて、ソースしか無く、記憶力も低下しているので、ShadowCut2 の更なる改良、改造は考えていません。プログラミングは改良よりも新規なものを作るのが楽しいと思っています。グレースケール 8bit 保存等の細かいスキルについては未習得ですが、画像処理の概略については慣れてきたので、今後は OpenCV などのライブラリーを利用したプログラミングを試みたいと思っています。

プログラミングについて

前作 ShadowCut V1.0 は Delphi4.0 という古い言語を用いて開発を行いました。次に何かを開発するときは c#を使用したいと思っていたので、開発の前に c#を学習するために参考書として「つくって覚える C#入門／オフィス加減」、「やさしい C#／高橋 麻奈」を読みました。「つくって覚える C#入門／オフィス加減」は後半の題材が当面の課題と合わなかったので半分ほどしか読まなかったのですが、「やさしい C#／高橋 麻奈」は初めから最後まで２、３回読みました。コーディング中は辞書代わりに「Visual C# 2010 逆引き大全 555 の極意」を利用しました。c#ではクラスは基本的な概念、手法であることはよく理解したのですが、実際にプログラミングを書く段になるとクラスの書式を決めるのが面倒なため、今回はクラスを一つも作らずにコーディングをしてしまいました。プログラムにあるクラスは自動的に生成した class Form1、class Form2・・・・のみです。Delphi の癖が抜けないため、頭の部分に多数の変数を定義して、グローバル変数の代わりとして使用しました。C#のメソッドでは、引数は複数個可能ですが、戻り値は 1 個しか返せないのは不便と思っていたのですが、クラスを使えば引数、戻り値を柔軟に使用することができるようです。今後、何かプログラムを作ることがあれば、できるだけクラスを作成するようにしたいと思っています。

退職後のプログラムに関する経験は、以下のような状況です。電子工作をしたいと思い、Arduino Uno を購入し、LED 点滅プログラムを初めとし、色々試しました。デジカメ用の撮影箱を「晴釣雨木ときどき旅自炊、tges 氏、http://tges.blog117.fc2.com/blog-category-6.html」を見本にして作成しました。本をコピ

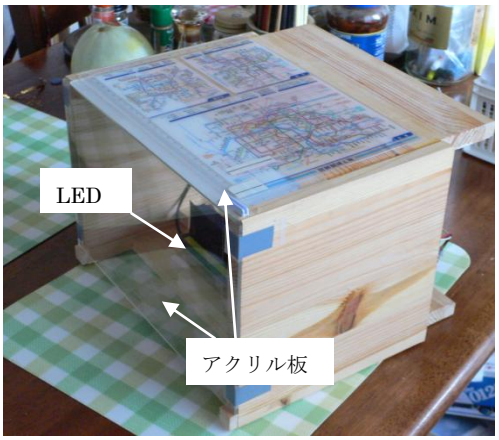
一箱にセットすると箱の周りの明るさが変化することに気付いたので、CdS を光センサーとし、サーボモータでデジカメ(オリンパス E-420)のシャッターを押すことを考えました。プログラムして、Arduino Uno に制御させると概ね可能であることが確認できました。2，3 冊はこの方式で撮影しましたが、本の固定静止のタイミングとシャッターのタイミングがしっくりしないので、手動シャッターに戻しました。Arduino Uno のプログラムは基本 C 言語なので比較的簡単に作成できました。Arduino Uno 類似のものとして PIC というチップがあることを知り、PIC12F683 という 8pin の PIC を購入しました。チップは 100 円程と非常に安価であることが一番の魅力でした。実際には書き込み機（PICkit3）が必要なので最終的には安くはなかったのですが。自宅のテラスに 20W のソーラーパネル、自動車用のバッテリー、充電コントローラを置いています。安物の充電コントローラのため LED 点灯表示しかなく、充電電流、一日の充電量が把握できないのが不満でした。そこで PIC12F683 に LCD ディスプレイを接続して充電量を表示できるようにしました。PIC の作成例を示しますが、PIC の特徴は、基盤が非常にシンプルなり、消費電流も非常に小さいことと思います。MPLAB IDE を使用して、初めて PIC のプログラミングを経験したのですが、Arduino Uno に比べると難しかったです。充電量のプログラムは 2，3 年前に作成したのですが、今、別のプログラムを作成するとなると、参考書をもう一度、最初から読まないとプログラムを書くことができないと思います。記憶力が低下しているのが大きな理由ですが、コンフィギュレーション設定、I/O ピンの設定、レジスターを HEX 値で設定するなど、細かい設定が必要なことがプログラミングを難しくしていると思います。Arduino Uno ではサンプルプログラムを何例か見れば、目的のプログラム作成ができると思います。1,2 年前にラズベリーパイ 2（1 G 容量）を購入しました。購入した一番の理由は、数学処理ソフトである Mathematica が無料で利用できることでした。その後わかったのですが、ラズベリーパイでは OpenCV のインストールが簡単にでき、Python で OpenCV を動かすことができました。今後は Python で OpenCV を勉強しようと考えています。プログラミングは楽しいのですが、何か課題がないことにはプログラムするモチベーションが起きないので、何か面白い課題はないかと探しています。

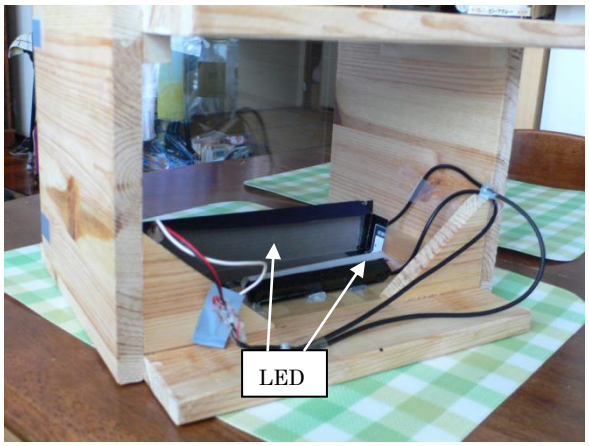


PIC12F683 基盤

撮影箱の作成

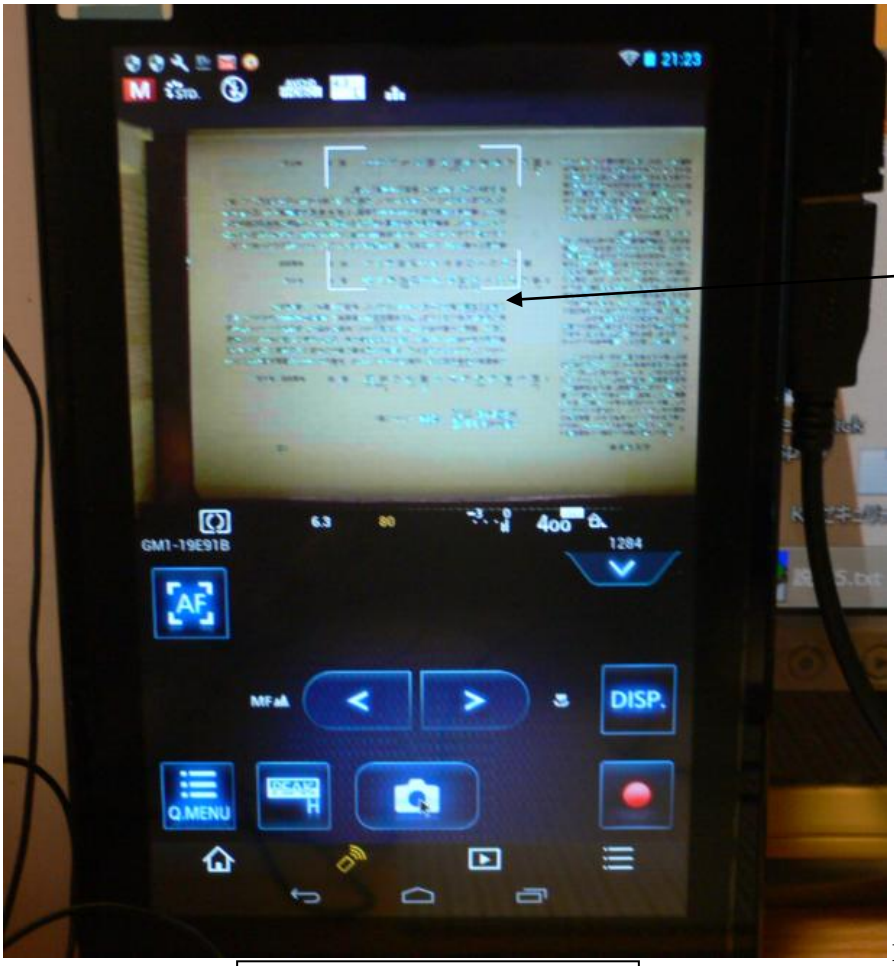
「晴釣雨木ときどき旅自炊、tges 氏、http://tges.blog117.fc2.com/blog-category-6.html」を手本にして作成しました。この web を拝見して、箱を用いることにより簡単にデジカメ撮影できることを初めて知りました。ただ、当方には工作技術があまりなく、工具もノコギリとドライバー等の簡単なものしか持っていないので、簡易的なものを作成しました。手順としては、1000x200x17mm（パイン合板）をホームセンターでカットしてもらい、250x200mm 板 2 枚、200x200mm 板 2 枚を用意しました。なお、4 カットまでは無料でした。長さ 900mm 合板を用いる場合、箱のサイズを変更する必要があります。木片に木工用ボンドを塗布し、図の様な箱になるように配置して、各辺を 90 度に保つようにテープなどで固定し、1 日乾燥しました。前面に 3mm の透明アクリル板を取り付け、強度を上げるために補強木片を付け加えました。静電気によるゴミの付着、擦り傷の発生の観点からはアクリル板でなく、ガラス板がよいと思っています。ただ何かの手違いでガラスを破損するのがいやでアクリル板にしました。次に作るとしたら、箱のサイズを少し大きくして、ガラス板を用い、照明の LED の数を増やし、照明ムラを少なくしたいと思っています。無反射ガラスは解像度を低下させる懸念があります。特に本の綴じ部の近くでガラス面との間に隙間が出来る場合もあるので、透明ガラス板が良いと思っています。



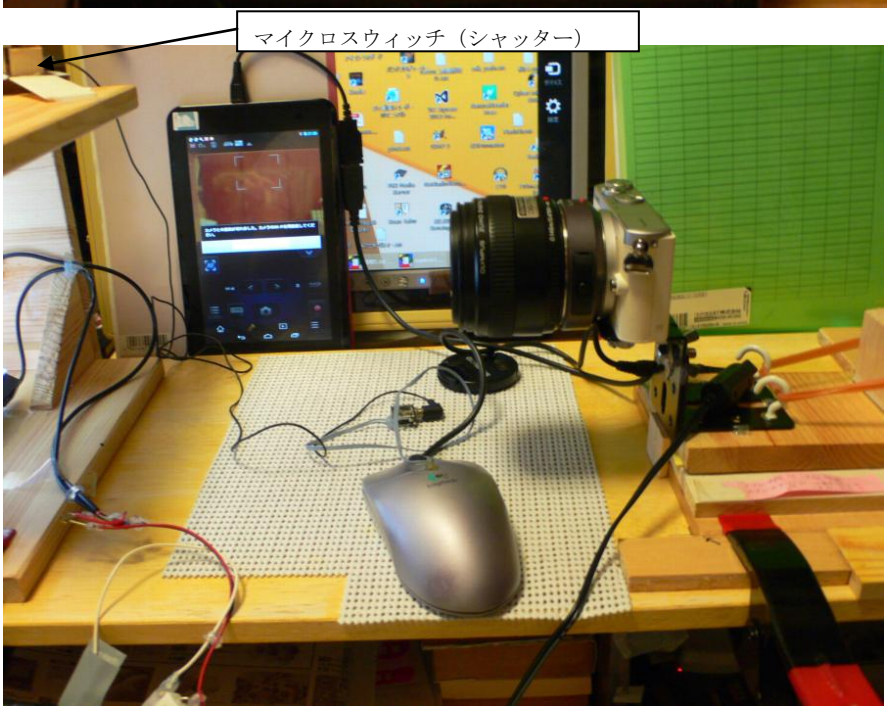


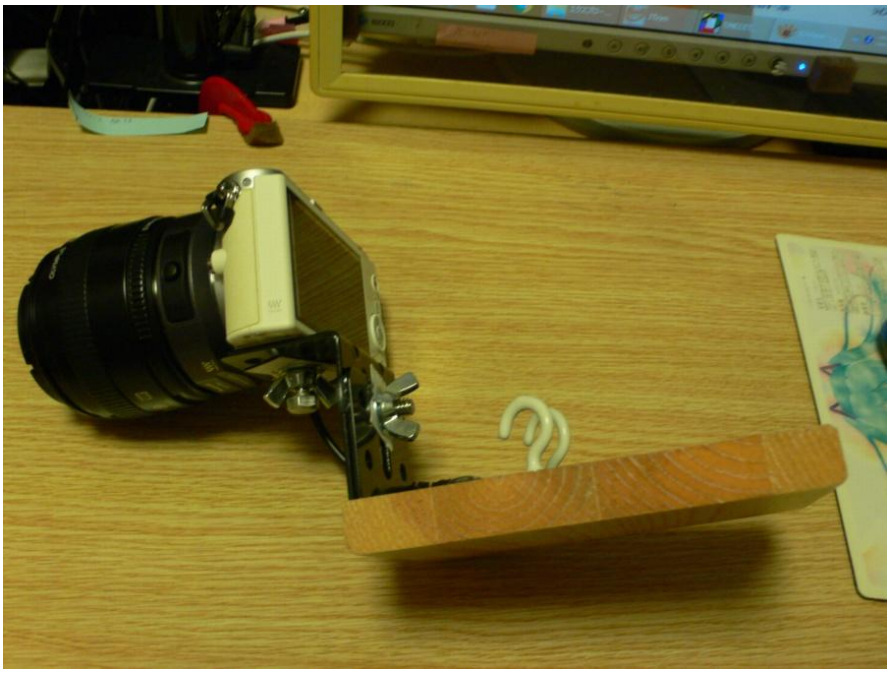
現在、ミラーレス一眼（LUMIX DMC-GM1K)、マクロレンズ（ズイコーデジタル 35mm F3.5 Macro + アダプタ）を使用しています。記録画素数は 4592x3448 ピクセル、シャッタースピード 1/80、絞り 6.3、ISO400 で撮影しました。撮影は、DMC-GM1K をタブレットと WiFi 接続し、マウスをクリックすることにより行いました。実際には利便性を上げるために、マウスの内部のスイッチ部分にケーブルを半田付けし、外部のマイクロスイッチによりシャッターを押すようにしました。フォーカルプレーンシャッターの寿命は 10 万回程度と言われており、カメラの寿命が心配ですが、電子シャッターモードで撮影を行っているため、シャッターの寿命は考慮しなくてよいと思っています。ミラーレス、電子シャッターなので原理上、ミラーの跳ね上がり、シャッターの振動がないので、解像度は良好に保たれると思われます。フォーカスはマニュアルモードを使用したので、撮影時のメカニカルな動作部分はレンズの絞り開閉のみと思われます。サイレントモードで撮影するとかすかな機械音が聞こえますが、これは絞り羽根の動作音を思われます。ピーキングの色によるフォーカスの適正確認と撮影範囲は、タブレット上で確認できます。照明は LED を取り付けることにより行いました。これが非常な難問となりました。はじめは市販の 100V、４W の棒状の LED を箱の下に適当に置いて撮影をしました。大きな照明ムラができてしまい、LED の位置を変えてもほとんど改善しませんでした。Scan Tailor を用いると照明ムラを除去できましたが、一部の活字がかすれるなど不都合がたまに発生することがありました。照明ムラはできるだけなくすことが重要と思いました。現在は LED 照明を 2 個配置することにより以前に比べかなり改善しました。アクリル板に不要な反射が起きない様に箱の上部面の一部、LED の後部に黒色の紙を貼り付けました。LED および黒色紙の位置は、試行錯誤で決める必要があります。コピーを速くする手順として以下のようにしました。ページ番号は見ない。和書の場合、最終ページから初めて、奇数ページのみをまずコピーする。次に偶数ページのみを初ページから初めて、最終ページまでをコピーする。原則として、ページのめくりは重力に沿うように上から下に自然に落下する様にする。ファイル番号とページ番号はバラバラになりますが、フリーソフト「ファイル名変更君」を利用することにより、ファイル番号をページ番号に変更することが可能です。本のコピー、プロ

グラミングしている時は懐メロ、映画などの録画番組を見ながら行っています。



p

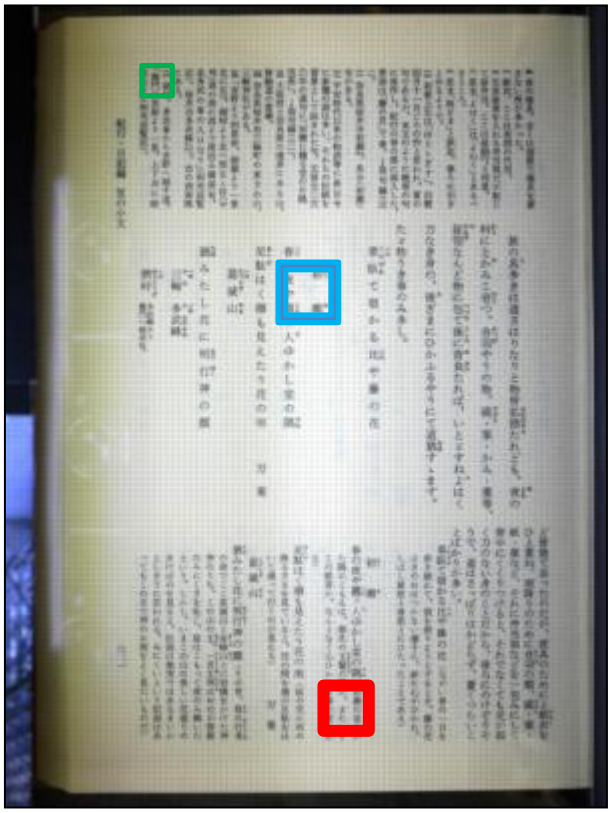




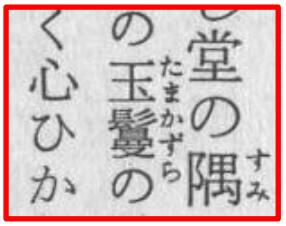
撮影箱を用いた場合の解像度

デジカメで撮影したものの解像度が、フラットベッドスキャナー（EPSON GT-8300UF）の 300dpi と同等ならスキャナーの替わりとして使用できると考えていました。解像度を比較した結果を図に示します。作成した撮影箱は最大 A5 サイズまでしか撮影できませんが、解像度は 300dpi を超えており、400dpi とほぼ同等と思われます。解像度はカメラの性能に依存しており、840 万画素の Panasonic DMC-LX1 では 300dip に達していないと思います。被写体サイズ、画像ドット数から計算するとコンデジ、ミラーレスそれぞれ見かけ上、解像度は 325dpi、522dpi となります。スキャナーの画像と比べるとデジカメの画像は輪郭にシャープさが無く、実際の解像度は見かけの解像度の 8 掛けと考えた方がよいと思います。カメラの解像度は画像の中心部が最も高く、周辺部で低下することが知られています。マクロレンズを用いた場合、周辺部でも十分な解像度を保持していることがわかります。用いたコンデジは 2005 年発売のものなので、現在の高画素のコンデジを用いて撮影すれば、おそらく 300dpi は優に超えると思います。

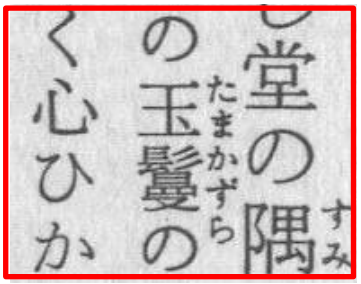
デジカメの解像度



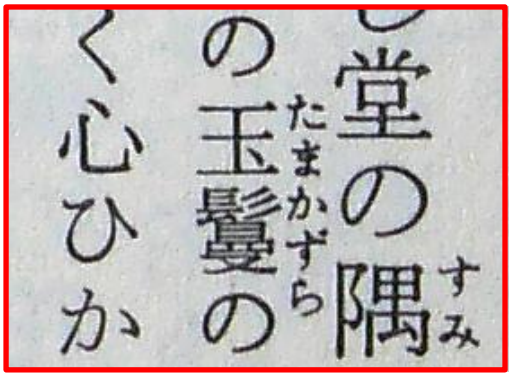
1600万画素ミラーレス＋マクロレンズ
日本古典文学全集、小学館
縦：217mmmm



スキャナー300dpi

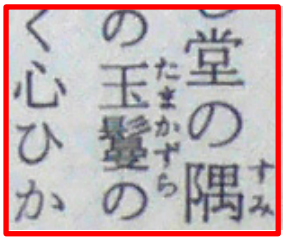


スキャナー400dpi

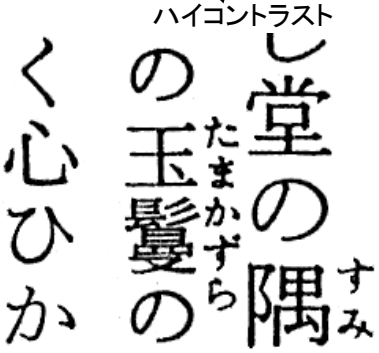


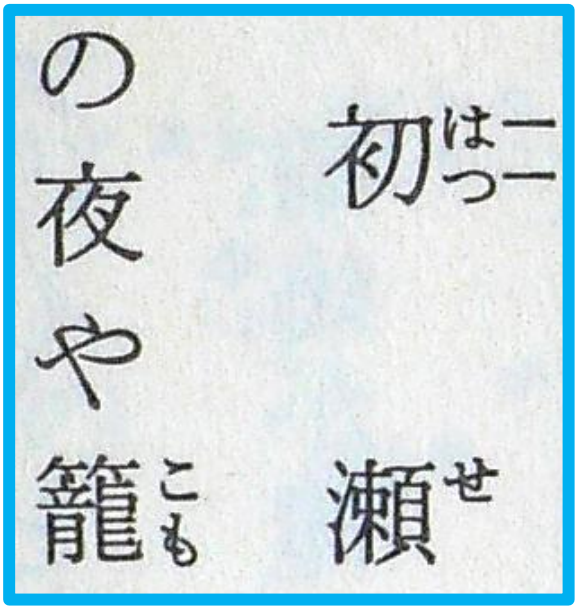
1600万画素ミラーレス＋マクロレンズ

ハイコントラスト

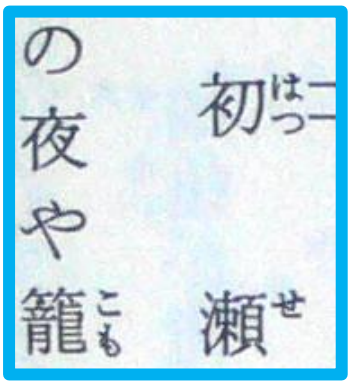


840万画素コンパクトデジカメ





1600万画素ミラーレス＋マクロレンズ



840万画素コンパクトデジカメ

1600万画素ミラーレス
＋マクロレンズ

作者プロフィール

年金暮らしの老人です。現在 68 才、62 才で退職してから仕事はしていません。趣味は、読書、テレビ、音楽鑑賞、楽器（ピアノ、ギター）、プログラミングなど。DTM は PC-9801 用のミュージ郎 55 の頃から行っています。現在は Singer Song Write Lite9 を主に使ってピアノ、ギターの練習に役立てています。VST のソフト音源にＰＭＩベーゼンドルファー２９０を使用しています。このソフト音源、購入して 10 年程経過していると思うのですが、Windows8.1 でもインストール、オーソライズできました。DTM を楽器練習に用いると、任意の範囲でのリピート、テンポ、音程のシフト調整が自由にできるのが利点です。ハード音源として 88pro を持っています。最近、ソースネクストからバンドプロデューサー（河合楽器）というソフトを特価で購入し、耳コピーの練習しています。スプートニックの「空の終列車」を練習したいと思ったのですが、楽譜が販売されていないので、このソフトで耳コピーを試みています。昔、nifty に MIDI のフォーラムがあり、MIDI ファイルの著作権管理が今ほど厳しくないため、多数の楽曲の MIDI ファイルが打ち込みの技を競う形でアップロードされていました。ベンチャーズなど多数の MIDI ファイルをダウンロードしましたが、「空の終列車」はダウンロードするのを忘れたみたいです。今年、Kindle Paperwhite を購入したのを機会に、英語の多読を趣味にしています。キンドルストアーは、洋書に安いものがたくさんあるのが魅力です。例えばイアン・フレミングの００７シリーズ１２冊組が２００円などです。英語力はそれほど高くないので、グリム童話、アンデルセン等の易しいものと、中級のシドニィ・シェルダン、アガサ、上級のハクスリー、オーウェル、ドイルなどを混ぜて読んでいます。中級以上はときどき日本語訳を参照しながら読んでいます。今の英語力では読んで内容を楽しむより、英語文を読解できたと確認できるのが楽しい段階です。在職中は英文の技術資料を読むのは仕事の一部でしたが、文学作品はそれに比べると難しいと思います。技術論文を自動翻訳すると、翻訳日本語で概略を理解できることが多いと思いますが、文学作品を自動翻訳すると意味不明の日本語をはき出すことがあります。

ハクスリーの「すばらしい新世界」の出だしの部分、読んでも意味が分からなかった箇所です。

「Cold for all the summer beyond the panes, for all the tropical heat of the room itself, a harsh thin light glared through the windows, hungrily seeking some draped lay figure, some pallid shape of academic goose-flesh, but finding only the glass and nickel and bleakly shining porcelain of a laboratory.」google のサイトで翻訳すると「すべての夏の間、寒さのために、部屋全体の熱帯熱、窓を照らした激しい薄い光、腹が立った寝たきりの姿、学問的なガチョウの肉の一部、 ニッケルと実験室の白濁陶器。」と日本語文章が良くなく、あまり参考になりません。本を電子化して将来のために保存をすることは当方の寿命を考えると、あまり意味がないのではと思うようになりました。１０インチのタブレットを利用していますが、大きな字で読むことができるのが電子化の一番のメリットと思っていま

す。タブレットで拡大できないときは、A4 でプリントすると読みやすくなります。テレビなどで老人を拝見すると元気な方が多いのですが、個人としては、あと５年、１０年いや２０年なのかわかりませんが、７０を過ぎると死は身近なものになると考えているので、「つひに行く道とはかねて聞きしかど　昨日今日とは思はざりしを。 死は前よりしも來らず、かねて後に迫れり。沖の干潟遥かなれども、磯より潮の滿つるが如し。死ぬる時節には死ぬがよく候。これはこれ災難をのがるる妙法にて候 」など先人の言葉が身近に感じられるようになりました。