平成１４年度　卒業論文

『 画像の位置合わせを行う
最適化アルゴリズムの構築 』

奈良女子大学　理学部　情報科学科

数理情報学講座　和田研究室

157 番　細井絵里子

概要

本論文では、画像の位置合わせを行う最適化アルゴリズムを提案する。位置合わせとは、2 枚の画像を比較し、それぞれの画像に含まれる物体どうしの位置が合うよう、画像を動かすことである。

本論文で提案するアルゴリズムは、画像を縮小して小さな画像から位置合わせを行い、順に画像を元の大きさまで戻して最適な位置合わせを決定するものである。位置合わせの方法は回転と平行移動のみである。具体的なアルゴリズムは、以下のとおりである。

① 2 枚の画像をそれぞれ 1/2 ずつ縮小していく。
② 小さく縮小した画像から順に位置合わせを行う。小さな画像では荒く、画像が大きくなるにつれて細かく行う。画像サイズが元に戻った時に、求めるべき位置合わせの場所が決定される。

本研究の目的は、DaltaViewer で電子顕微鏡画像や光学顕微鏡画像、またはデジタルカメラなどから得られた画像を使用可能にすることである。DaltaViewer とは連続断面画像から 3 次元イメージを再構成するプログラムだが、従来のものは、扱える画像が共焦点レーザー顕微鏡のみであった。他の画像が扱えない原因は、隣接する画像どうしの位置が一致していないことにあったため、それらを解決すべく、本研究の開発に着手した。

結果として、このアルゴリズムを用いた位置合わせは、実用時間内で可能となった。そのためこのアルゴリズムを DaltaViewer に組み込むことで、本研究の目的である、DaltaViewer での電子顕微鏡画像や光学顕微鏡画像、デジタルカメラなどから得られた画像の使用が実現された。

目次

1. はじめに

本研究の背景には、和田昌昭教授が以前より着手してきたアプリケーションプログラム：DaltaViewer の存在がある。DeltaViewer とは、共焦点レーザー顕微鏡によって得られた断面画像から、任意の境界を抽出して曲面データを三次元構築し、それを Open GL を用いてコンピュータ画面上に表示するものである。注目すべきは、扱える画像が“共焦点レーザー顕微鏡画像”のみという点である。なぜ、一般的な顕微鏡画像は使用できないのだろうか？

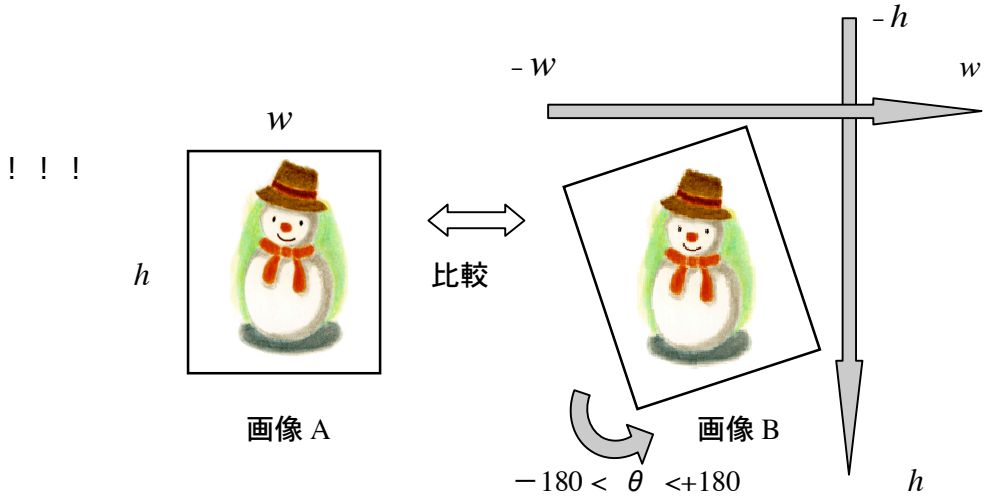
原因として、“隣接する画像どうしの位置関係”があげられる。共焦点レーザー顕微鏡は深さ方向の分解能が存在するため、非破壊・非接触で研究対象の連続断面画像を得ることができる。その結果、画像間の位置合わせは初めから出来ており、DaltaViewer に位置合わせプログラムは必要なかった。対して、電子または光学顕微鏡で得られた断面画像は、人間が実際に研究対象を切断し観察したものである。そのため、どうしても画像どうしの位置にズレが生じてしまう。このことから、DaltaViewer で電子または光学顕微鏡画像を使用可能にするには、画像どうしの位置合わせを行う必要があった。

上記のような理由から、本研究の開発に着手した。そのため本論文では主に電子顕微鏡で得られた物体の連続断面画像を用いて位置合わせを行っていく。

2. 位置合わせとは

本論文で扱う画像はディジタル画像である。ディジタル画像には、縦横に並んだ画素の 1 pixel ごとに RGB 値が定められている。RGB 値とは、色情報を数値化したカラーモデルのひとつで、赤（Red)、緑（Green)、青（Blue）の光の三原色によって色を表現する。つまり、 2 枚の画像の位置合わせをするとは、この RGB の値を出来るだけ近付ければ良い。そこで、画像 B を回転･平行移動させ、画像 A との各ピクセルごとの RGB 値を比較し、その差の 2 乗和が最小となる角度･x 軸 y 軸への水平移動距離を求めれば、画像の位置合わせが出来ると考えられる。

単純に考えるならば、 1 つの画像（画像 B）を－180～180 度回転させ、画像の幅を w、高さを h とすると、x 方向に $-w \sim +w$, y 方向に $-h \sim +h$ 移動させて、元となる画像（画像 A）と比較し、最適な値を見つければ良い。



しかし、この方法では処理するのに莫大な時間がかかってしまう。例をあげて説明する。

Ex) 画像サイズ [500, 500]
位置合わせを行う範囲 [200, 200] とする。
画像を 360 度、x、y 方向にそれぞれ 1000 pixel ずつ動かすので
画像を動かすだけで

$$360 \times 1000 \times 1000 = 360000000 \quad [回]$$

もの計算が必要となる。

また位置合わせを行うための誤差の計算も行わなければならない。
その計算に約 1/12 秒かかるので、全部で

360000000 × 1/12 ≒ 8333 [時間]

もの時間が必要という計算になる。これは約一年の時間に相当し、
これでは実際に扱うには不適当である。

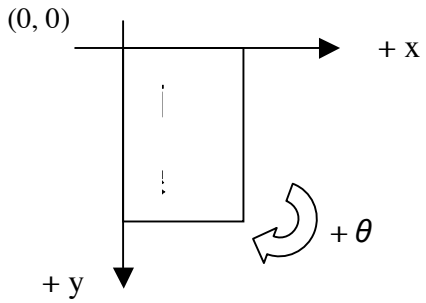
そこで、処理時間短縮のために独自のアルゴリズムを提案した。以下で詳しく説明していくが、その前にまず位置合わせを行うための画像の回転と平行移動・補間について、説明していきたい。

3. 画像の回転・平行移動と補間法

本研究では位置合わせを行うために、画像の回転と平行移動を行う。ここでは、一般的な画像の回転と平行移動について、またそれらを出力する際の補間方法について述べる。

3.1 回転

一般的にディジタル画像は、画像の左上を原点として考えている。そのため回転方向も通常の反時計回りとは逆の”時計回り”となる。



そのため、通常通り反時計回りに画像を θ 回転させるには、$-\theta$ と入力しなければならない。これをふまえて考えるなら、入力点(x1, y1)を原点(0, 0)を中心として θ 度回転させた場合の出力点の座標(x2, y2)は

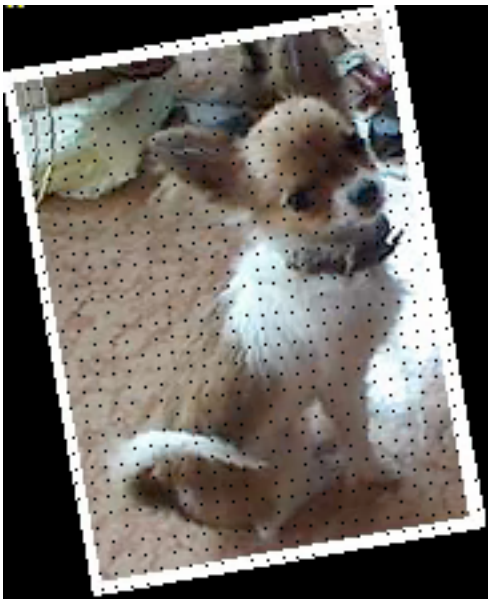
$$x2 = x1 * \cos(-\theta) - y1 * \sin(-\theta)$$
$$y2 = x1 * \sin(-\theta) + y1 * \cos(-\theta)$$

となる。

この式を使って画像を回転させると、画像の左上を中心とした回転になる。これを画像の中心(cx, cy)を回転の中心とする画像にしたい場合、式は

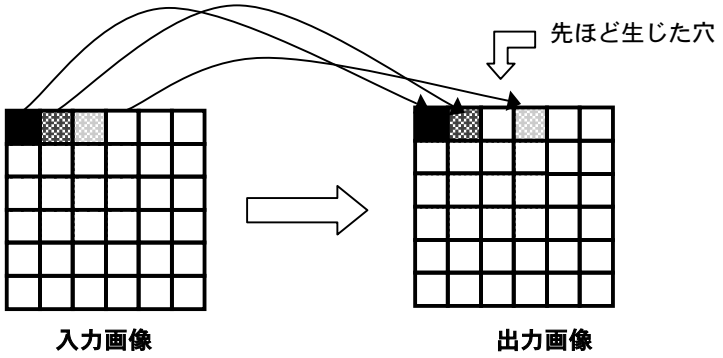
$$x2 = (x1-cx) * \cos(-\theta) - (y1-cy) * \sin(-\theta) + cx$$
$$y2 = (x1-cx) * \sin(-\theta) + (y1-cy) * \cos(-\theta) + cy$$

となる。これで画像の中心を原点とした回転が出来るようになった。しかし実際に実行すると、下図のように変換後の画像に穴が生じてしまった。

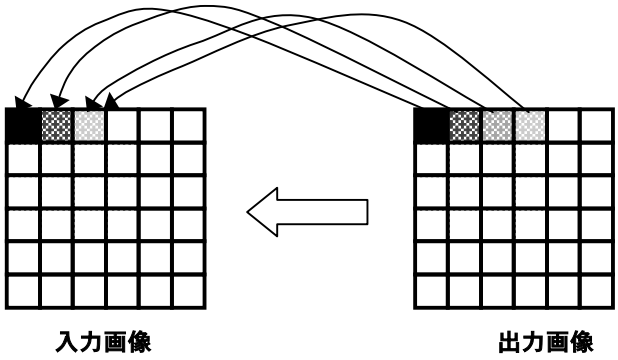


. **サンプル画像** **回転処理後画像(角度:10 度)**

これは、入力画像の位置から出力画像の座標の計算をしたために生じた結果である。というのも、画像は回転しているので、入力座標(x1,y1)から求められた出力座標(x2, y2)は、実数である。しかし画像は整数座標に RGB 値が格納されているため、ここでは(x2,y2)の値をそれぞれ切り捨てにして整数とし、その整数が表す座標に(x1,y1)の値が格納される。そのためすべての(x2,y2)座標に値が格納されるとは限らない。よって、画像を出力する際穴が生じてしまうのである。



そこでこの現象を回避するには、入力画像から出力画像を求めるのではなく、下図のように出力画像の座標に対応する入力画像の座標を逆算すればよい。なぜなら、出力画像の座標から、逆に入力画像の座標を求めることによって、必ず出力座標にはなんらかの値が格納されるからである。



このとき用いる式は、

$$x1 = (x2-cx) * \cos(\theta) - (y2-cy) * \sin(\theta) + cx$$
$$y1 = (x2-cx) * \sin(\theta) + (y2-cy) * \cos(\theta) + cy$$

となる。

3.2　平行移動

平行移動は、単純に画像を移動させたい分だけ動かせばよい。入力画像の座標を(x1, y1)、出力画像の座標を(x2, y2)とし、平行移動する距離を(u, v)とするなら、式は

$$x2 = x1 - u$$
$$y2 = y1 - v$$

となる。この式を用いることで画像は、入力画像より左に u ピクセル、上方向に v ピクセル平行移動して出力される。

3.3　補間

しかし以上のような方法で、回転・平行移動を行うとエッジの目立った汚い画像となってしまう。そこで補間をかける必要がある。補間とは画像を拡大・縮小させたり回転させる際、増加または減少した画素（補間画素）に与える階調値を原画像の画素の階調値から計算することをいう。主に３つの方法がある。ここでは、その１つである Bilinear を用いるが、参考までに他の方法についても述べておく。

3.3.1 ニアレストネイバー法(Nearest Neighbor)

原画像の画素をそのまま自然数倍する方法で、最近接点割り当て (nearest neibor assignment),最近内挿法 (nearest neibor interpolation) とも呼ばれる。例えば、２×２の原画像を４×４の画像に拡大するには、原画像の各要素を同じ階調値の２×２画素に拡大すれば良い。同様に、２n×２n の画像に拡大するには、原画像の各画素を同じ階調値のｎ×ｎ画素に拡大すれば良い。

例　　2×2　→　　4×4

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 2 & 2 & 3 & 3 \\ 2 & 2 & 3 & 3 \end{bmatrix}$$

式で表すとすると、入力画像 f(i, j)をＡ倍したときの画像 $f'(m,n)$ の階調値は

$$f'(m,n) = f(\lfloor m/A + 1/2 \rfloor, \lfloor n/A + 1/2 \rfloor)$$

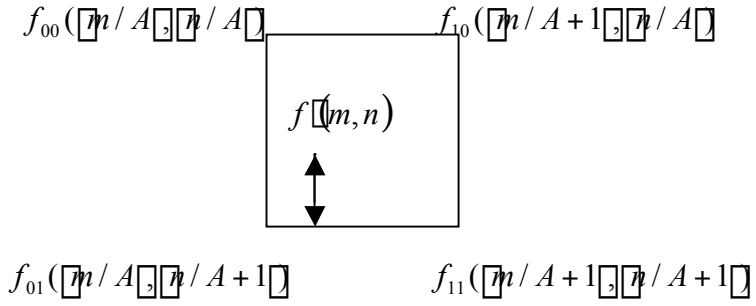
という式で求められる。ここで $\lfloor m/A + 1/2 \rfloor$ は、簡単に言い表すと、ｍ／Ａを四捨五入することを示す。

この方法の利点は、オリジナル画像データを壊さず、アルゴリズムが簡単なことである。そのため非常にコンピュータ向きであり、高速に処理できる。しかし、画素をそのまま拡大しているので拡大率が大きくなるにつれてエッジ部でのジャギーが目立つようになってしまう。

3.3.2　バイリニア法(bilinear)

本研究で扱っている補間方法である。線形補間 (linear interpolation) などとも呼ばれ、１次補間の一種である。補間画素の周囲４つの画素の階調値から線形近似によって、階調値を求める方法である。

入力画像を f(i, j)とし、その画像をＡ倍するとする。Ａ倍された画像 f'(m, n)の階調値を求めるには、



$$\delta_x = m/A - \lfloor m/A \rfloor$$
$$\delta_y = n/A - \lfloor n/A \rfloor$$

と定義すると、

$$f'(m,n) = \begin{bmatrix} f_{00} & f_{10} \\ f_{01} & f_{11} \end{bmatrix} \begin{bmatrix} 1-\delta_x \\ \delta_x \end{bmatrix}^T \begin{bmatrix} 1-\delta_y \\ \delta_y \end{bmatrix}$$

で求められる。

この方法では、頂点系列で表現された平均化により図形の境界が滑らかになる効果があるが、オリジナルデータは壊され、図形の境界線がぼやけてしまう。さらに、拡大倍率が大きくなると補間される画素の数が原画像の画素数よりもはるかに多くなる。そのため、拡大倍率が大きくなるに従い、階調の平均化、すなわち画像のぼやけが顕著になってしまう。

3.3.3 バイキュービック法(Bicubic)

キュービックコンボリューション (cubic convolution) とも呼ばれる。補間画素の周囲 16 個の画素の階調値から 3 次補間関数をもちいて補間画素の階調値を計算する方法である。

原画像の画素を

$$\begin{matrix} p_{00}(x_0,y_0,g_{00}) & p_{10}(x_1,y_0,g_{10}) & p_{20}(x_2,y_0,g_{20}) & p_{30}(x_3,y_0,g_{30}) \\ p_{01}(x_0,y_1,g_{01}) & p_{11}(x_1,y_1,g_{11}) & p_{21}(x_2,y_1,g_{21}) & p_{31}(x_3,y_1,g_{31}) \\ p_{02}(x_0,y_2,g_{02}) & p_{12}(x_1,y_2,g_{12}) & p_{22}(x_2,y_2,g_{22}) & p_{32}(x_3,y_2,g_{32}) \\ p_{03}(x_0,y_3,g_{03}) & p_{13}(x_1,y_3,g_{13}) & p_{23}(x_2,y_3,g_{23}) & p_{33}(x_3,y_3,g_{33}) \end{matrix}$$

とする。ただし g_{kl} は画素 (x_k, y_l) の階調値を表わすものとする。また補間画素を $p(x,y,g)$ とする。求めたい諧調値は

$$g = \begin{pmatrix} C(y_0 - y) & C(y_1 - y) & C(y_2 - y) & C(y_3 - y) \end{pmatrix} = \begin{vmatrix} g_{00} & g_{10} & g_{20} & g_{30} \\ g_{01} & g_{11} & g_{21} & g_{31} \\ g_{02} & g_{12} & g_{22} & g_{32} \\ g_{03} & g_{13} & g_{23} & g_{33} \end{vmatrix} \begin{vmatrix} C(x_0 - x) \\ C(x_1 - x) \\ C(x_2 - x) \\ C(x_3 - x) \end{vmatrix}$$

ただし、関数 C(x)は

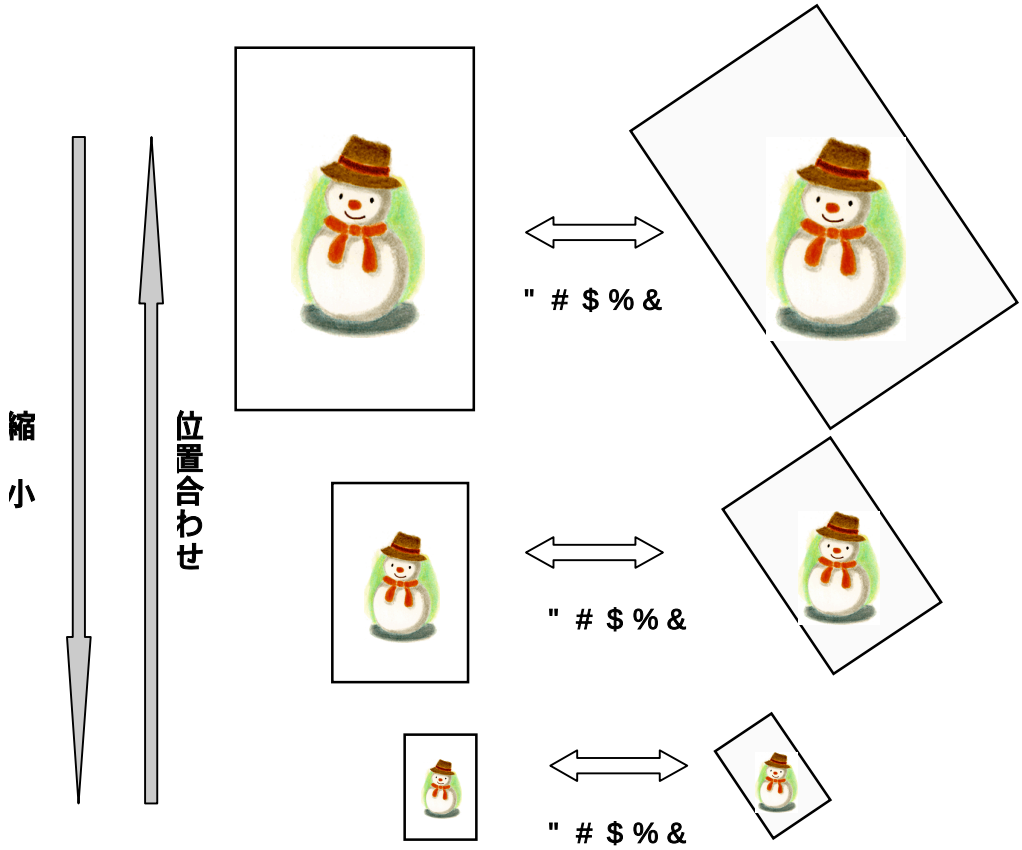
$$C(x) = \sin(\pi x)/(\pi x) \approx \begin{cases} 1 - 2|x|^2 + |x|^3 & 0 \le |x| < 1 \\ 4 - 8|x| + 5|x|^2 - |x|^3 & 1 \le |x| < 2 \\ 0 & 2 \le |x| \end{cases}$$

によって定義される。これは標本化定理で現れる関数の多項式近似である。

バイキューブ法では、平滑化効果によるジャギーは消え、鮮鋭化効果でバイリニア法ほどには画像がぼけない。一般的に最も結果の良い補間法とされている。しかし、計算量が多く、処理速度が遅い、画像に若干の揺らぎが生じるなどの欠点も見受けられる。

4. 最適化アルゴリズム

では、次に私の提案したアルゴリズムについて述べていく。そのアルゴリズムとは、画像を縮小して位置合わせを行う方法である。下図のように 2 枚の画像を縮小していき、小さな画像から順に位置合わせを行う。縮小するにつれて画像はぼやけていくので、小さな画像では粗く調べることでおおまかな角度・水平移動距離を求め、画像が大きくなるにつれて細かく調べることで、より詳細な値を導くことが出来る。



では、このアルゴリズムを４つの段階

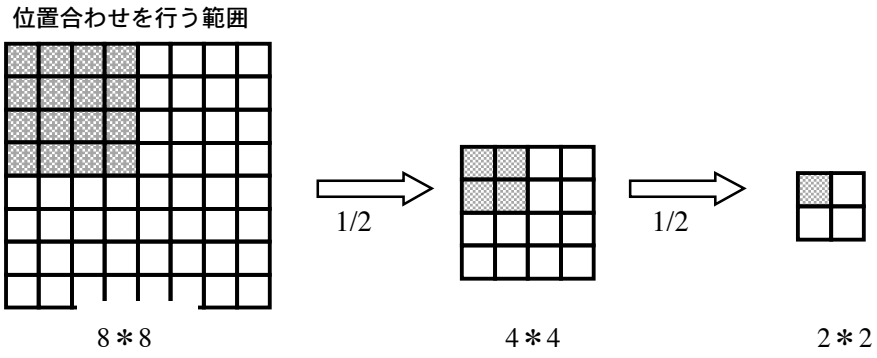
1　画像の縮小
2　最初の位置合わせ
3　最適解かどうかの判定
4　最適な位置の決定

に分けてより詳しく説明する。

4.1　画像の縮小

まず初めに画像の縮小を行う。縮小倍率は、基本は 1 / 2 とする。ただし、画像の幅・高さが奇数の場合、 2 で割って四捨五入した値を縮小した画像の幅・高さとする。つまり、サイズ[15*15] の画像を縮小する場合、次のサイズは[8*8]となる。このときの縮小倍率は 8/15 である。

これを、位置合わせを行う範囲が 2＊2 の大きさになるまで続ける。ここで、n 段階縮小した画像を A_n, B_n、それぞれの画像の幅・高さをそれぞれ w_n, h_n とする。また画像 A_{n-1} から A_n に縮小した時の縮小倍率を S_n とする。

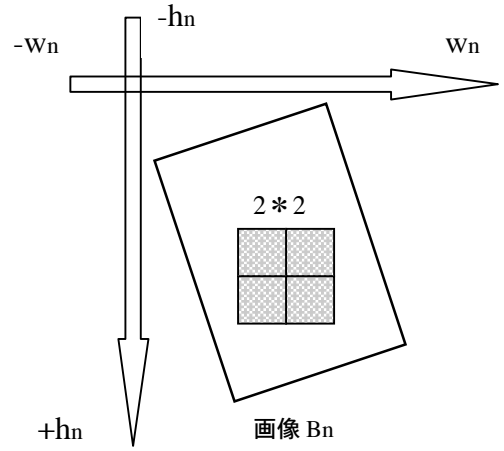


縮小方法としては、様々なものがある。本研究では先ほど述べた Bilinear を用いて縮小する。

4.2　最初の位置合わせ

画像の縮小ができたら、いよいよ最初の位置合わせを行う。最初の位置合わせは、位置合わせを行う範囲が 2*2 のサイズまで縮小した画像で行う。最初なので、画像 B_n を-180 ～ +180 度、x 方向には $-w_n \sim +w_n$ 、y 方向には $-h_n \sim +h_n$ 回転・平行移動させて行う。そして画像 A_n と比較して、誤差が最小となる角度、x 軸 y 軸への水平移動距離を求める。

では、実際にどのように誤差を求めているのか以下で詳しく説明する。



画像 B_n を

$$-180 < \theta < +180$$

$$-w_n < x < +w_n$$

$$-h_n < y < +h_n$$

の範囲で回転・平行移動させ、
画像 A_n と比較する

4.2.1 調べる度合い

まずどれくらいの細かさで調べているかだが、それはユーザーが指定する精度に比例してくる。精度が 1.0 とは、1pixel 単位で位置合わせがされているという意味である。

精度を P とするなら、x 方向 y 方向にはそれぞれ P pixel ずつ平行移動させて誤差の計算を行えば良い。回転方向も同様に P pixel ずつ動かさなければならないが、
そのための計算は以下のとおりである。

$r * \theta$ = 精度　　r : 位置合わせを行う範囲の半径
θ : 角度（単位はラジアン）

θ の単位はラジアンなので、角度に直すには θ に、180.0 / π を掛ければよい。最初の位置合わせでは、位置合わせを行う範囲が 2*2 まで縮小した画像なので、上記の r は 1.0 となる。精度を 0.5 とするなら、角度 degree は

$$\text{degree} = 0.5 * 180.0 / 1.0 * \pi$$
$$= 28.64788978$$

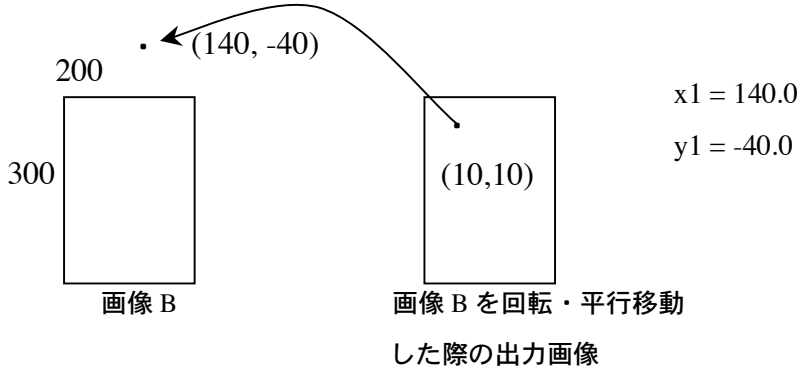
となる。以上のことから画像サイズが大きくなればなるほど、調べる角度の度合いは細かくなっていくといえる。

4.2.2 誤差の計算

次は、誤差の計算である。最初にも述べたとおり、ここでは 2 枚の画像の位置合わせを行いたい範囲の RGB 値をそれぞれ比較して、その差の二乗和を誤差とする。

単純に引き算をして計算していけばよいが、一つ注意点が存在する。それは画像を回転・平行移動させた際、求めた座標が元の画像にない場合である。

具体例をあげて説明する。画像 B を 90 度回転させ、そこから x 方向 y 方向に 100 pixel ずつ平行移動させた際、出力画像の座標(10, 10)が元の画像(画像 B）のどの位置にあたるか計算すると以下のようになる。



ここで求められた座標(140.0, -40.0)は元の画像(画像 B)には存在しない。しかしこのような場合にも、誤差の計算は行わなければならない。では、どうしたらよいだろうか？

ここでは、画像 B と位置合わせを行う画像 A の値を参照する。元の画像に求めた座標が存在しないということは、出力座標には画像 B の情報はいっさい含まれていないといえる。そのような場所での位置合わせを避けるために、画像 A の値と最も離れた値をその出力座標に格納しておけば、間違ってもその場所で位置合わせが行われることはない。そこで、本研究では、

(画像 A の RGB の値 / 255.0) ＞ 0.5 → 出力座標には 0
(画像 A の RGB の値 / 255.0) ＜ 0.5 → 出力座標には 255.0

を格納することにする。

こうすることで、誤差の計算はほぼ正確にできるようになった。

4.3 最適解かどうかの判定

次に最初の位置合わせができたら、その求めた値が最適解かどうかの判定を行う。が、その前になぜこのような判定を行う必要があるのかについて述べる。

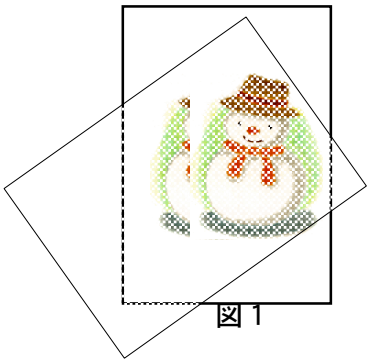


図 1

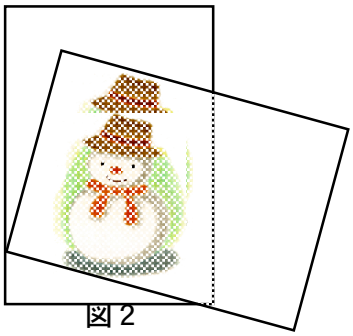


図 2

' *) 4.2 0 8 9 6 : ; < = > ? @ A B 0 1 2An 3 1 2Bn 4
! ! ! " # $ % & 5 6 7 - ⇒ C D E F G

' ( ) ' * + , - . / 0 1 2An 3 1 2Bn 4 " # $ % & 5
⇒ C D E H H H

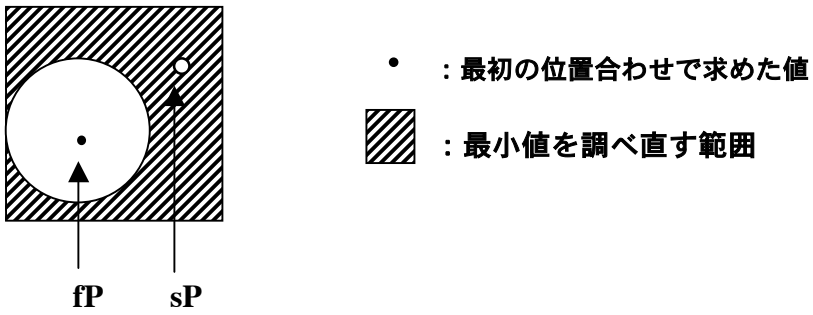
図 1 は、最初の位置合わせで求めた角度･水平移動距離で、位置合わせしたものである。それに対し、図 2 は図 1 以外の場所で位置合わせしたものである。この時、図 1 の誤差の値は最小値である。しかしここで、図 2 の誤差の値も、最小値とほぼかわらないとしたらどうだろうか？どちらが最適な位置合わせの場所なのだろうか？もしかしたら、誤差が最小である図 1 よりも図 2 のほうが適しているかもしれない。

このように、最小に近い値が多く存在すると、どの場所が一番適しているのか一概には判断がつけられない。そこで、最初の位置合わせで求めた値が本当に最適な場所なのか判断する必要があるのである。

ではその具体的な判定の方法について述べる。

4.3.1 判定方法

まずはじめに、最初の位置合わせで求めた値以外の場所で、もう一度最小値を調べ直す。このとき最初の位置合わせで求めた値の周りは、調べる必要がない。



こうして得られた最小値を sMin とし、その場所 sP する。最初の位置合わせで求めた場所が fP でその最小値を fMin とすると、fMin と sMin の差が大きければ最適な位置合わせの場所は fP と確定できる。

しかし、fMin と sMin の差が小さければ fP と sP どちらが最適な位置合わせの場所なのか判断がつかないので、このような場合はその縮小サイズでの位置合わせはあきらめ、画像サイズを一段階前の縮小サイズに戻し、もう一度最初の位置合わせをやり直す必要があるといえる。では、fMin と sMin の差をどのように設定すればこのような判定ができるのだろうか？

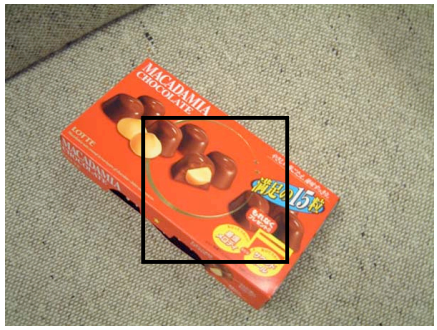
それは、“画像の種類”に左右される。そのため一概にはいえない。そこでここでは電子顕微鏡画像とデジタルカメラから得られた画像を使ってそれぞれの場合はどんな結果になるのか実験を行った。以下にその結果を添付する。

4.3.2　実験結果

位置合わせは Agreement Point を中心とする Range の範囲で行う。Range をどの大きさまで縮小して位置合わせを行ったか、Size で表す。Fmin と Smin はそれぞれ fMin、sMIn の点の誤差を％で示したものである。Magunification は Smin が Fmin の何倍であるかを示した値である。それぞれ位置合わせができたら○、出来なかったら×で表す。

(1) デジタルカメラから得られた画像　サイズ[600,450]

Agreement Point [300, 225]

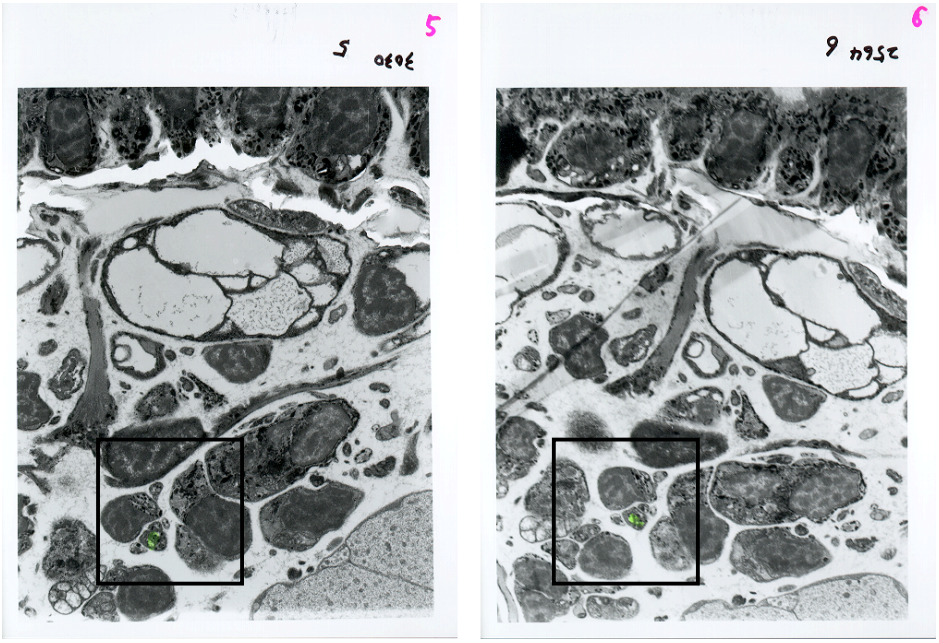


Range	Size	Fmin	Smin	Magnigication	○or×
50*50	2*2	0.288579	1.22082	4.230453359	○
100*100	2*2	0.0874714	0.253175	2.894374618	×
	4*4	0.437789	1.33896	3.058459669	○
200*200	2*2	0.105277	0.478662	4.54669111	○
	4*4	0.253712	0.863201	3.402286845	○

この結果から、デジタルカメラから得られた画像は、Smin が Fmin の約.3 以上なら確実に位置合わせが出来るといえる。

(2)　電子顕微鏡から得られたイソアワモチ幼動物の断面画像　サイズ [493,693]

Agreement Point [150,580]：ここには白黒以外のはっきりとした特徴がある



Range	Size	Fmin	Smin	Magnigication	○or×
100*100	2*2	1.19358	1.23242	1.03254076	×
	4*4	1.83044	2.12903	1.163124713	○
	8*8	1.9579	2.9623	1.512998621	○
200*200	2*2	0.21889	0.622815	2.845333272	○
	4*4	1.95217	3.2826	1.681513393	○

この結果から、この場合は Smin が Fmin の.約 1.5 以上なら確実に位置合わせ出来るといえた。

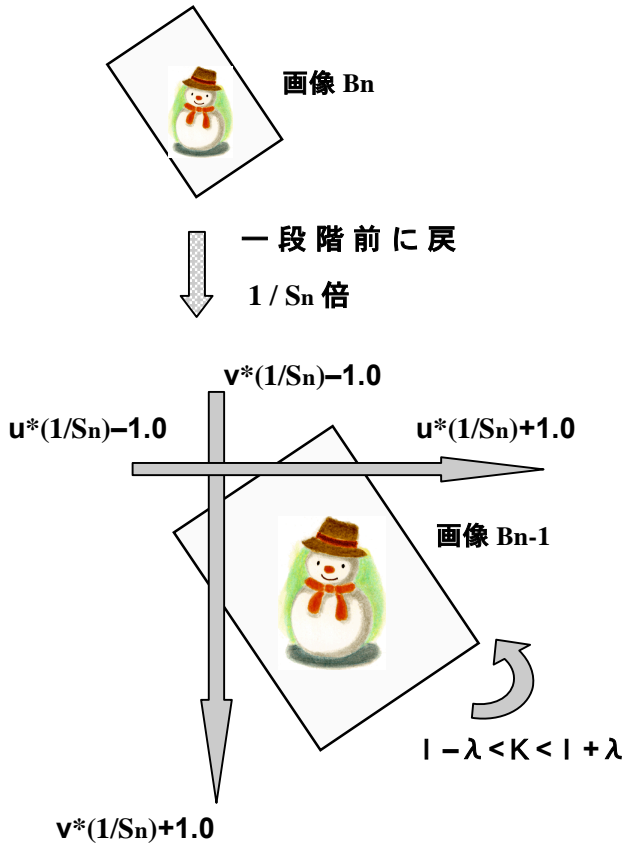
4.4 最適な位置の決定

以上のような方法で最初の位置合わせが確定出来たなら、次はより詳しい位置合わせを行う。

まず初めに 2 枚の画像サイズを一段階前の縮小サイズに戻す。

縮小サイズをもどしたら、最初の位置合わせで求めた角度･水平移動距離に、はばをもたせた範囲で、もう一度誤差が最小となる角度・x 軸 y 軸への水平移動距離を求める。

なお、ここで述べた“はば”とは以下のとおりである。φ, u, v は最初の位置合わせで求めた値である。S_nは画像B_{n-1}から画像B_nに縮小した際の縮小倍率である。



I − J < K < I + J

u*(1/Sn) ! − ! 1.0 ! < ! x ! < ! u*(1/Sn) ! + ! 1.0

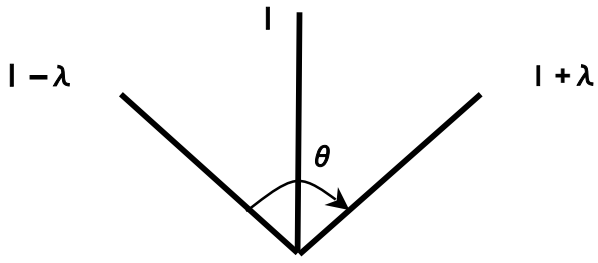
v*(1/Sn) ! − ! 1.0 ! < ! y ! < ! v*(1/Sn) ! + ! 1.0

λは一段階前に戻した画像での、動かす角度の度合いである。φ−λからφ+λの範囲をλずつ動かすので、3点を調べる計算になる。そこで、その3点を

Z[0] = φ − λ

Z[1] = φ

Z[2] = φ + λ



とおく。

x・y 方向の調べる範囲は、画像サイズを一段階前に戻すので、最初の位置合わせで求めた値(u,v)に 1/Sn 掛けた値から±1.0 を調べる範囲とする。1.0 pixel ずつ調べるので、角度と同様それぞれ 3 点ずつ調べることになる。その 3 点をそれぞれ

X[0] = u*(1/Sn)–1.0　　Y[0] = v*(1/Sn)–1.0

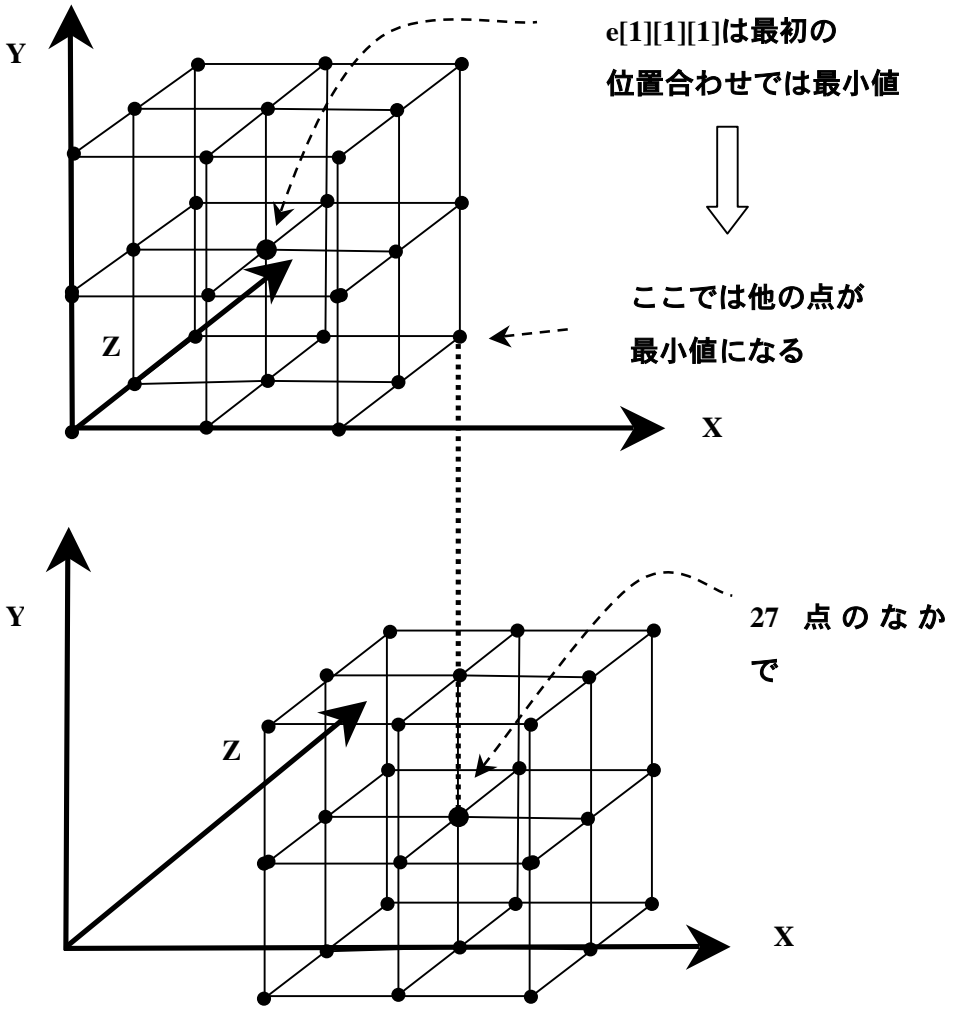
X[1] = u*(1/Sn)　　Y[1] = v*(1/Sn)

X[2] = u*(1/Sn)+1.0　　Y[2] = v*(1/Sn)+1.0

とおく。

回転も x 軸 y 軸への平行移動もそれぞれ 3 点調べるので、3×3×3 で、27 回誤差の計算を行う。X[i],Y[j],Z[k]の点の誤差の値を e[i][j][k]とする。

ここで、注目したいのが X[1]Y[1]Z[1]の点である。これは最初の位置合わせで誤差が最小になった点である。しかし、画像サイズを一段階前に戻したことで、他の点が最小値となるかもしれない。



もし他の点が最小になったなら、再びその周りの 27 点の誤差の計算を行い、e[1][1][1]が 27 点の中で最小になるまでこの作業を繰り返す。

この作業が完了したなら、27 点内をさらに細かく調べていく。その方法とは 27 点を通る関数を作りその最小値を求める方法である。以下にその方法を示す。

① まず初めに 27 点を通る関数を、ラグランジュの補間多項式を用いて作成する。この関数を $f = \sum C_{i,j,k} x^i y^j z^k$ とおく。

② 最小値を求める必要があるので、f の関数をそれぞれ x, y, z で偏微分する。2 回微分も行う。それぞれを以下のようにおく

$$\frac{\partial f}{\partial x} = f_x = \sum_{i=0.1} (i+1) C_{i+1,j,k} x^i y^j z^k \qquad \frac{\partial^2 f}{\partial x^2} = f_{xx} = \sum_{i=1} i(i+1) C_{i-1,j,k} x^{i-1} y^j z^k$$

$$\frac{\partial f}{\partial y} = f_y = \sum_{j=0.1} (j+1) C_{i,j+1,k} x^i y^j z^k \qquad \frac{\partial^2 f}{\partial y^2} = f_{yy} = \sum_{j=1} j(j+1) C_{i,j-1,k} x^i y^{j-1} z^k$$

$$\frac{\partial f}{\partial z} = f_z = \sum_{k=0.1} (k+1) C_{i,j,k+1} x^i y^j z^k \qquad \frac{\partial^2 f}{\partial z^2} = f_{zz} = \sum_{k=1} k(k+1) C_{i,j,k-1} x^i y^j z^{k-1}$$

$$\frac{\partial f}{\partial x \partial y} = f_{xy} = \sum_{\substack{i=0.1 \\ j=0,1}} (i+1)(j+1) C_{i+1,j+1,k} x^i y^j z^k$$

$$\frac{\partial f}{\partial x \partial z} = f_{xz} = \sum_{\substack{i=0.1 \\ k=0,1}} (i+1)(k+1) C_{i+1,j,k+1} x^i y^j z^k$$

$$\frac{\partial f}{\partial y \partial z} = f_{yz} = \sum_{\substack{j=0.1 \\ k=0,1}} (j+1)(k+1) C_{i,j+1,k+1} x^i y^j z^k$$

③ 27 点の中心を(x0, y0, z0)とする。その点で関数の傾きが最小となる方向は

$$(-f_x(x0,y0,z0),\ -f_y(x0,y0,z0),\ -f_z(x0,y0,z0))$$

となる

④ 次にその方向に向かう直線を考える。

$$(x(t), y(t), z(t)) = (x0, y0, z0) + t * (f_x(x0,y0,z0)), (f_y(x0,y0,z0)), (f_z(x0,y0,z0))$$

とおく。ただし t<0 とする。

⑤　この直線上で関数 f の 1 階および 2 階微分を計算してみると

$$\left(\frac{d}{dt}\right) f\left(x(t),y(t),z(t)\right) = f_x(x,y,z) * f_x(x0,y0,z0) + f_y(x,y,z) * f_y(x0,y0,z0) + f_z(x,y,z) * f_z(x0,y0,z0)$$

$$\left(\frac{d^2}{dt^2}\right) f\left(x(t),y(t),z(t)\right) = f_{xx}(x,y,z) * f_x(x0,y,0,z0)^2 + f_{yy}(x,y,z) * f_{yy}(x0,y0,z0)^2 + f_{zz}(x,y,z) * f_{zz}(x0,y0,z0)^2 + 2f_{xy}(x,y,z) * f_x(x0,y0,z0) f_y(x0,y0,z0) + 2f_{xz}(x,y,z) * f_x(x0,y0,z0) f_z(x0,y0,z0) + 2f_{yz}(x,y,z) * f_y(x0,y0,z0) f_z(x0,y0,z0)$$

となる。よって t=0 のところでは

$$\left(\frac{d}{dt}\right) f\left(x(0),y(0),z(0)\right) = f_x(x0,y0,z0)^2 + f_y(x0,y0,z0)^2 + f_z(x0,y0,z0)^2$$

$$\left(\frac{d^2}{dt^2}\right) f\left(x(0),y(0),z(0)\right)$$
$$= f_{xx}(x0,y0,z0) * f_x(x0,y0,z0)^2 + f_{yy}(x0,y0,z0) * f_{yy}(x0,y0,z0)^2 + f_{zz}(x0,y0,z0) * f_{zz}(x0,y0,z0)^2 + 2f_{xy}(x0,y0,z0) * f_x(x0,y0,z0) f_y(x0,y0,z0) + 2f_{xz}(x0,y0,z0) * f_x(x0,y0,z0) f_z(x0,y0,z0) + 2f_{yz}(x0,y0,z0) * f_y(x0,y0,z0) f_z(x0,y0,z0)$$

となる。

⑥　t の関数として t=0 での 1 階および 2 階微分が分かったので、あとはそれを元に 1 階微分が 0 になる t の値を求めれば良い。ただし 2 階微分というのは、t が 1 増えるごとに 1 階微分の値がどれだけ変化するかという量なので、

$$t = -\frac{\frac{df}{dt}}{\frac{d^2 f}{dt^2}}$$

のところで 1 階微分の値が 0、つまり f の最小が求まる。

以上のような方法で最小値がもとまったら、またさらに画像サイズを一段階前の縮小サイズに戻し、同様の作業を行う。これを画像サイズがもとの大きさになるまで繰り返す。元の大きさに戻った時に得られる角度・水平移動距離が求めるべき位置合わせの値である。

以上が私の提案したアルゴリズムの概要である。実際にこのアルゴリズムを用いて作ったプログラムを本論文の最後に付け加えておく。

5. 結果

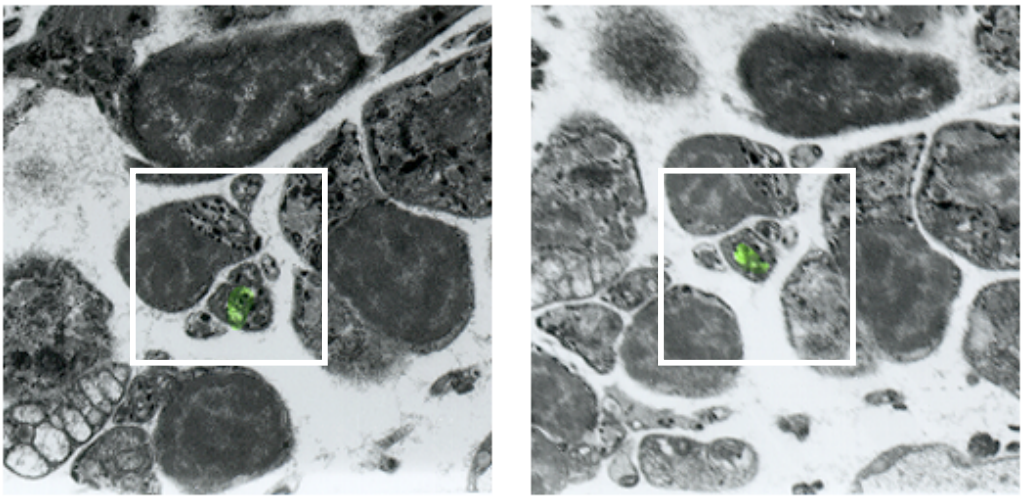
ではこのアルゴリズムを利用して、位置合わせした結果を表示する。

扱った画像‥‥電子顕微鏡画像（イソアワモチ幼動物の外套の断面画像）

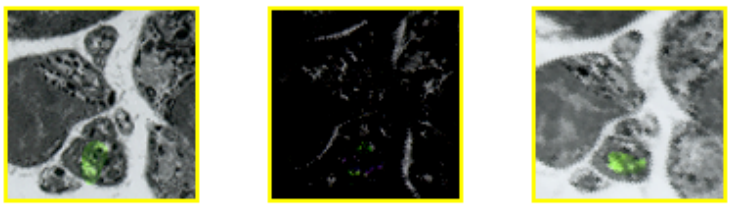
サイズ [250*250]

位置合わせを行った範囲‥‥白い枠で囲まれている部分

サイズ [100*100]



位置合わせ



図（１）　図（２）　図（３）

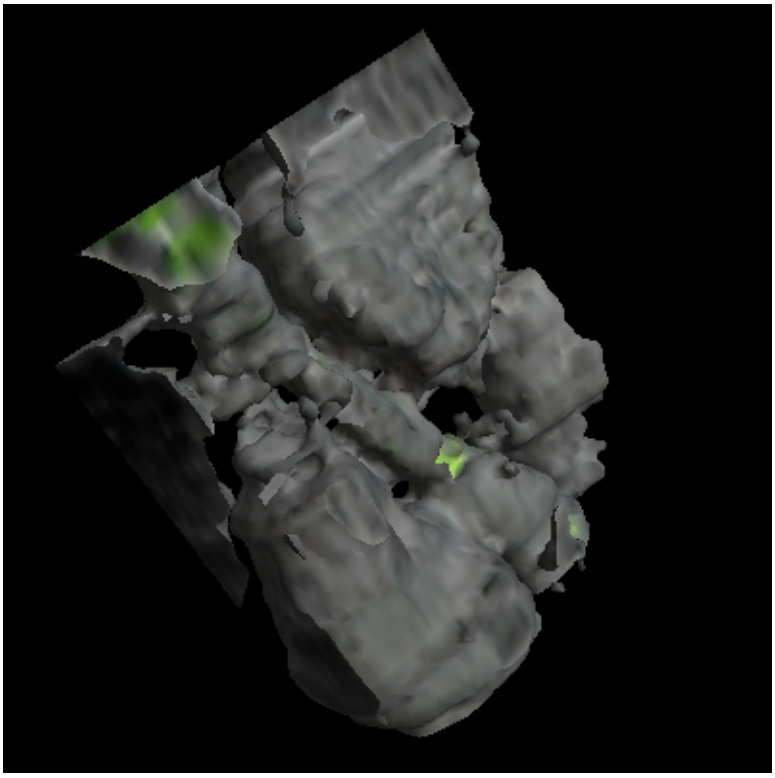
図(1)：画像*A*の位置合わせを行った範囲

図(2)：画像*B*に回転・平行移動を行い画像*A*との位置合わせを行ったもの

図(3)：図(1)と図(2)の誤差を表す画像　一致するほど黒くなる

またこのアルゴリズムを実際に DaltaViewer に組み込みこんだ。ではその結果も表示する。

この画像は 39 枚の電子顕微鏡画像を読みこみ、それぞれ位置合わせを行ってから、境界を抽出し、３D 構築したものである。なお、ここで扱う画像も先ほどと同じイソアワモチ幼動物の外套の断面画像である。



このように、それぞれの断面画像の位置合わせを行ったおかげで、電子顕微鏡から得られた画像も３D 構築することができるようになった。

6. 考察

本研究で提案したアルゴリズムを利用することで、実用時間内での位置合わせが可能となった。その結果、大量の画像データを扱う DaltaViewer での位置合わせに用いることができ、DaltaViewer で電子顕微鏡画像やまたは一般のデジタルカメラによる画像の使用も可能となった。

しかし、まだまだ問題点も多い。本研究の位置合わせは回転と平行移動のみだが、実際には画像を拡大・縮小、または何らかの画像処理をしてからでないと位置合わがうまくいかない画像もある。また短時間になったとはいえ、まだ実行時間がかかりすぎる問題もある。これらが、今後の課題となっていくだろう。

謝辞

この研究を行うにあたって、ご指導いただいた和田昌昭教授には本当に深く感謝しております。和田先生には言葉では言い表せないほど、多大なご迷惑をおかけしてしまいました。なかなか理解することの出来ない私に、何度も何度も適切なアドバイスをしてくださり、どれだけ助けになったか分かりません。本当にありがとうございました。

またいろいろ相談にのってもらい、いつも力になってもらった、同研究室の伊佐友江さん、藤原佐知子さん、許潤玉さんにも深く感謝しております。

また同室として一緒に頑張ってきた山下ゼミの永井さゆりさん、部坂芙美さん、吉田ちはるさん、若野雅実さんにも深く感謝いたします。

参考文献・参考 URL

文献

- 姜秀子：修士論文「 DaltaViewer プロジェクト 」(2001)
- 増田将宣：修士論文「 等高線表現に基づく画像の補間 」(2001)

URL

- 酒井理雄：「 C++言語による画像回転処理について 」
  http://tsugu.no-ip.com:8080/sotuken/rotation/
- 「 画像補間法 」
  http://www2.nsknet.or.jp/~numada/ip05.html