平成２１年度　特別研究報告書

イラスト画像に対する対象物抽出

龍谷大学　理工学部　情報メディア学科

学籍番号　　T060548　奥村亮仁

指導教員　三好 力　　教授

内容梗概

本論文では、イラスト画像に対する画像分類の自動化を目標として、その前処理にあたる対象物抽出の自動化を検討する。従来手法として分裂した動的輪郭モデルを用いた対象物抽出について説明する。従来手法の問題点として、エッジの強さと凹形状により、対象物の内部に輪郭が入り込み、対象物全体の抽出しようとすると対象物が欠けてしまう問題があった。この問題点の解決とイラスト画像に対する最適な対象物抽出手法を検討するために、従来手法を拡張して分裂する動的輪郭モデルと領域分割を用いた対象物抽出の手法と、変分ベイズ法による混合正規分布推定と領域分割を用いた対象物抽出の手法を提案手法として説明する。従来手法と 2 つの提案手法について、複数のイラスト画像を用いて比較実験を行い、計算速度や対象物抽出の性能を評価し考察する。

目 次

第1章　はじめに

1.1　研究の背景と目的

今日、インターネットの利用者数の増加と発展[1)]により、多くの情報がやり取りされるようになった。中でも World Wide Web(以下、Web) は、電子メールと並ぶインターネットを利用する主な目的の 1 つとなっており多くの人に利用されている。Web 上には多くの画像が存在し、絵を描いて画像ファイルを個人サイトで公開している人も多い。最近では個人サイトだけでなく、ブログのようなサイトよりも簡単に公開できるものや pixiv 等の絵や漫画のアップロードと公開に特化したサイトなどが存在しており、より公開しやすい環境が整ってきている。多くの場合、公開されているものはイラスト画像と呼ばれる静止画像である。イラスト画像の例を図 1.1 に示す。

公開される画像の数が多くなるとともに、それらを私的に集める人も多くなり、集める量も多くなっていると考えられる。集めた画像を分類する作業は現在ほぼ手作業でするしかない。そこで、この作業を自動化できれば便利であると考えた。しかし、イラスト画像に特化した分類方法だけでなく、前処理となる対象物抽出の方法が見当たらなかった。したがって、まずイラスト画像を対象物抽出するために最適な手法を検討することにした。

分裂する動的輪郭モデル[5)]を従来手法として、従来手法の論文では顕微鏡画像やカメラで撮った動画像などで実験が行われており、イラスト画像に対する検討は行われていない。イラスト画像は風景が描かれたものから物体が描かれたもの、写真と大差がないものまで様々である。写真は多くの場合、イラスト画像より色の数が多く、背景にも対象物にも特徴が多い。一方、顕微鏡画像は背景が単色であり、対象物にしか特徴がなく、対象が細胞のように凹凸が少ないものが多い。イラスト画像は、背景が単色または写真より背景が単純なものが多い。また、顕微鏡画像に比べると、対象物に大きな凹凸が多く対象物が画像を占める割合が大きい。この違いを踏まえた上で、イラスト画像に対して従来手法と提案手法で抽出して比較検討を行い、イラスト画像に合う抽出手法を考える。

本論文は、イラスト画像に対する対象物抽出の自動化を目的として最適な手法を検討する。2 章で、手法の種類を説明し、従来手法の説明とその問題点について述べる。その後、従来手法の問題を解決し、イラスト画像に対する領域抽出を行うための 2 つの手法を提案する。提案手法のうち 1 つは従来手法を発展させ問題解決を図ったもの、もう 1 つはアプローチを変えて混合正規分布を用いて解決を図ったものである。3 章で、2 章で示した手法について比較実験を示し、その結果から問題解決できているかを検討し、イラスト画像に対する対象物抽出はどの手法が一番良い結果と言えるかを考える。

1.2　イラスト画像

1.1 で述べたように、イラスト画像は人が描いた絵で静止画像である。最近の多くの絵に見られる傾向について考える。

図 1.1: イラスト画像の例

イラスト画像には、人物や生物等の一般的にキャラクタと呼ばれる対象となるものと、それ以外の背景が存在する。多くの場合、キャラクタを目立たせるために、キャラクタは輝度が急激に変化する部分が多く、背景は輝度が急激に変化する部分が少ない。したがって、キャラクタにエッジやコーナーが多く現れると考えられる。一般的な写真に比べると、色がまとまって存在しており色の数が少ない。計算機上で描かれた絵が多くなってきており、それについてスキャナで取りこんだ絵に比べるとノイズが少ないと思われる。

本論文で使用するイラスト画像について、特徴が違う画像をいくつか用意して実験を行う。上記で述べた最近の多くの絵に見られる傾向のものだけではなく、その傾向に当てはまらないものも用意して実験を行い、その傾向に当てはまるものと当てはまらないもので従来手法または提案手法について比較検討を行う。

1.3 エッジと輪郭線

1.3.1 エッジについて

エッジとは、画像中の輝度が急激に変化する部分のことである。一例として図 1.2 に元の画像とエッジを検出した画像を示す。

図 1.2: 元画像とエッジ画像

1.3.2　輪郭線について

本論文では、エッジと輪郭線を区別する。エッジは輪郭とも呼ばれ、輪郭線とも呼ばれることがあるが、混同しないためにエッジと輪郭線を定義を以下に示す。

エッジは輝度が急激に変化する部分だけを示し、点の集合であって線ではないとする。一方、輪郭線は対象物を囲む閉曲線であり、対象物の領域と対象物外の領域の間に存在するものとする。

エッジは対象物の領域と対象物外の領域に現れることが多い。これは対象物と対象物外の間は輝度の変化が大きいことが多いからである。しかし、輝度の変化が大きい点を抽出するのがエッジであり、ここで定義した輪郭線とは別である。ここで定義した輪郭線はエッジの有無は関係なく、対象物の輪郭を表す閉曲線である。閉曲線はエッジのように点ではなく線である。

第2章　手法

2.1　対象物抽出の種類

以下に、対象物抽出の手法について述べる。[2) 3) 7)]

2.1.1　エッジ抽出を用いた手法

エッジを追跡し、閉曲線になるようにすれば、輪郭線が得られる。輪郭線の内部を対象物の領域と考えて、内部だけ抽出すれば対象物の領域が得られる。このようにして、エッジ抽出を用いて対象物抽出を行うことを基本とする手法である。しかし、エッジが途切れている場合には追跡できずにうまく抽出できる閉曲線を得る事が出来ない。多くの画像の場合、ノイズやエッジが途切れることが多く、安定した結果が得ることが難しい。

2.1.2　領域分割を用いた手法

領域分割[8)]とは、画像データを局所的な特徴が一様な部分画像に分割する手法をいう。図2.1に色について領域分割を行った例を示す。例では赤、緑、青で表される画像をそれぞれ領域として区別し、それぞれ0、1、2という名前を領域につけている。0から2までのうち、例えば0だけ抜き出せば赤の部分画像が得られる。領域分割、はこのように色のような画像の特徴から領域を作り、名前をつけて部分画像に分割できるようにする処理である。

領域分割に用いられる手法として、近傍の画素を見領域に属しているかどうかを判断していく分割併合法や、特徴の平均を用いてあらかじめ決めた個数のクラスタに分類するK-Means法が従来存在する。ノイズに比較的強いが、分類した結果が対象物と限らないために対象物かどうかを判断するための後処理が必要となる。

2.1.3　動的輪郭モデルを用いた手法

動的輪郭モデル[4)]とは、対象物の近傍に置いた閉曲線を予め定義したエネルギー関数の値が最小になるように閉曲線の制御点を逐次的に動かし、エネルギー関数の値が最小になったとき、閉曲線を対象物の輪郭線とするものである。

図2.2に動的輪郭モデルのイメージを示す。図2.2における矢印は、閉曲線が対象物に引き寄せられるエネルギーを示している。対象物に対して、このエネルギーが最小になるようにモデル化する。エネルギーには曲線の滑らかさやエッジが用いられ、改良された動的輪郭モデルでは、それらに加えて違うエネルギーが用いられることもある。これらのエネルギーからエネルギー関数をつくり、逐次的にエネルギー関数を計算して最小になったとき、対象物の輪郭に沿った閉曲線が得られる。この閉曲線が輪郭線となる。

輪郭線が得られれば、2.1.1と同じように、輪郭線内部を抽出すれば対象物の領域を得る事が出来る。

2.2.1、2.3.1 では、Kass による動的輪郭モデルを改良した手法である分裂する動的輪郭モデル [5] を用いる。

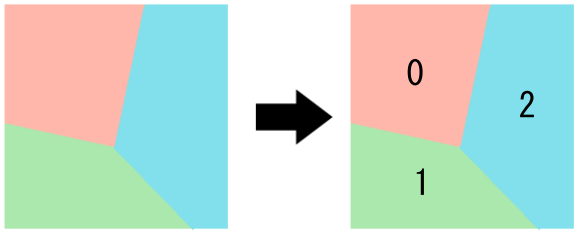

図 2.1: 色による領域分割の例

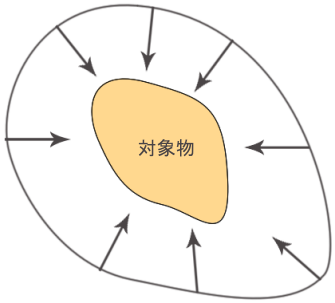

図 2.2: 動的輪郭モデルのイメージ

2.2 従来手法

2.2.1 分裂する動的輪郭モデルを用いた対象物抽出

概要

従来手法である分裂する動的輪郭モデル [5] を用いた対象物抽出について示す。

分裂する動的輪郭モデルは、Kass らによる動的輪郭モデルを改良したものである。Kass らによる動的輪郭モデルで必要であった、初期輪郭の数だけ対象物の数の用意することと、初期輪郭を対象物の近傍に設置することが分裂する動的輪郭モデルでは必要なくなった。これは分裂する動的輪郭モデルでは輪郭が交差することがあり、それを利用して輪郭モデルを分裂させることにより、実現している。

以下に、Kass らによる動的輪郭モデルと分裂する動的輪郭モデルについて示す。

動的輪郭モデル

Kass らによる動的輪郭モデル [4] は、対象物の近傍に置いた閉曲線 $\boldsymbol{v}(s) = (x(s), y(s))(0 \leq s \leq 1)$ を内部エネルギー $E_{int}(\boldsymbol{v}(s))$、画像エネルギー $E_{image}(\boldsymbol{v}(s))$、外部エネルギー $E_{con}(\boldsymbol{v}(s))$ によ

り定義されるエネルギー関数 E_{snakes} を最小化するように閉曲線の制御点を逐次的に動かして閉曲線を変形させ、エネルギー関数が極小状態を形状として輪郭を抽出するモデルである。

$$E_{snakes} = \int_0^1 (E_{int}(\boldsymbol{v}(s)) + E_{image}(\boldsymbol{v}(s)) + E_{con}(\boldsymbol{v}(s)))ds \tag{2.1}$$

E_{int} は、輪郭の滑らかさを表す次式の E_{spline} が用いられることが多い。α、β は重み係数である。

$$E_{spline}(\boldsymbol{v}(s)) = \frac{1}{2}\left(\alpha\left|\frac{d\boldsymbol{v}(s)}{ds}\right|^2 + \beta\left|\frac{d^2\boldsymbol{v}(s)}{ds^2}\right|^2\right) \tag{2.2}$$

E_{image} は、エッジの大きさを表す次式の E_{edge} が用いられることが多い。$I(\boldsymbol{v}(s))$ は画像の輝度、$\nabla I(\boldsymbol{v}(s))$ はエッジ、γ は画像エネルギーに対する重み係数を表す。

$$E_{edge}(\boldsymbol{v}(s)) = -\frac{1}{2}\gamma\left|\nabla I(\boldsymbol{v}(s))\right|^2 \tag{2.3}$$

E_{con} は、外部から強制的にエネルギーを与えるもので必要に応じて使われるパラメータである。

このモデルは、対象物の凹の形をした部分に入り込もうとする力が、隣接する制御点の内部エネルギーの引っ張る力によって入り込もうとする画像エネルギーの力と相殺され、凹の形の抽出が難しい。また、始めに対象物の数だけ閉曲線を作る必要があり、かつ、対象物の近傍に置く必要という問題がある。

分裂する動的輪郭モデル

上記で述べた Kass による Snakes に存在する問題点を解決したものが、分裂する動的輪郭モデル [5] である。分裂する動的輪郭モデルでは、エネルギー関数に面積項 E_{area} を導入で凹の形を抽出でき、閉曲線が分裂する事によって、始めから対象物の数だけ閉曲線を作ることと近傍に置く必要がない。以下、分裂する動的輪郭モデルについて説明する。

閉曲線は制御点が時計回りか反時計回りのどちらと定義されるかを決めておき、閉曲線の制御点を離散点 $\boldsymbol{v_i}(x_i, y_i)(i = 1, 2, ..., n)$ とする。ただし、$\boldsymbol{v}_0 = \boldsymbol{v}_n$、$\boldsymbol{v}_{n+1} = \boldsymbol{v}_1$ とする。分裂する動的輪郭モデルのエネルギー関数 E_{snakes} は以下のように定義される。

$$E_{spline}(\boldsymbol{v}_i) = \frac{1}{2}\sum_{i=1}^{n}(w_{sp1}|\boldsymbol{v}_i - \boldsymbol{v}_{i-1}|^2 + w_{sp2}|\boldsymbol{v}_{i+1} - 2\boldsymbol{v}_i + \boldsymbol{v}_{i-1}|^2) \tag{2.4}$$

$$E_{area}(\boldsymbol{v}_i) = \frac{1}{2}\sum_{i=1}^{n} w_{area}(x_i(y_{i+1} - y_i) - y_i(x_{i+1} - x_i)) \tag{2.5}$$

$$E_{dist}(\boldsymbol{v}_i) = \frac{1}{2}\sum_{i=1}^{n} w_{dist}|d_{avg} - |\boldsymbol{v}_i - \boldsymbol{v}_{i-1}||^2 \tag{2.6}$$

$$E_{edge}(\boldsymbol{v}_i) = -\frac{1}{2}\sum_{i=1}^{n} w_{edge}|\nabla I(\boldsymbol{v}_i)|^2 \tag{2.7}$$

$$E_{snakes}(\boldsymbol{v}_i) = E_{spline}(\boldsymbol{v}_i) + E_{area}(\boldsymbol{v}_i) + E_{dist}(\boldsymbol{v}_i) + E_{edge}(\boldsymbol{v}_i)(\boldsymbol{v}_i) \tag{2.8}$$

w_{sp1}、w_{sp2}、w_{area}、w_{dist}、w_{edge}≥0 は、各エネルギー項の重み係数である。E_{dist} は制御点間の距離の平均化、E_{intens} は画像の輝度を表す。全ての $\boldsymbol{v}$ について、$E_{snakes}(\boldsymbol{v}_i)$ が最小になるように $\boldsymbol{v}_i$ を動かす。

面積項 E_{area} によって、Kass らによる動的輪郭モデルでは難しかった凹形状も抽出可能となる。その原理を説明する。式 (2.6) の右辺第 1 項は、y 軸に関して、幅 $(y_{i+1} - y_i)$、高さ x_i の長方形で表される符号付き面積の総和である。右辺第 2 項は、x 軸に関して同様の計算を行っている。輪郭に沿って、最初に定義した回り方のほうが正となり、逆が負となる。この計算が凹形状に輪郭が入り込むエネルギーとなる。

面積項 E_{area} を導入すると、閉曲線が交差する現象が発生する。これは、交差される部分で算出される面積項が負になることによって起こるものである。この交差を切り離して閉曲線を分裂させる。図 2.3 に交差時の分裂の様子を示す。交差の判定は次式の実数 p、q について $0 \leq p \leq 1$、$0 \leq q \leq 1$ を満たすかどうかを調べる。

$$p(\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) + \boldsymbol{v}_i = q(\boldsymbol{v}_{j+1} - \boldsymbol{v}_j) + \boldsymbol{v}_j \tag{2.9}$$

交差している場合、$\boldsymbol{v}_i$ と $\boldsymbol{v}_{j+1}$、$\boldsymbol{v}_{i+1}$ と $\boldsymbol{v}_j$ を連結して、輪郭モデルを 2 つにして、最初に決めた回り方になるようにする。

このモデルでは、制御点の生成と消滅も行う。制御点の生成は、$\boldsymbol{v}_i$、$\boldsymbol{v}_{i+1}$ 間の距離が閾値 D_{TH} 以上であれば、$\boldsymbol{v}_i$、$\boldsymbol{v}_{i+1}$ 間に制御点を生成する。制御点の消滅は、$\boldsymbol{v}_i\boldsymbol{v}_{i-1}$ と $\boldsymbol{v}_i\boldsymbol{v}_{i+1}$ のなす角 θ について、予め決めた閾値 θ_{TH} を用いて $cos\theta > \theta_{TH}$ を満たす $\boldsymbol{v}_i$ を消滅させる。また、分裂した輪郭モデルの制御点の数が 5 未満の場合、対象が小さすぎるかつ $E_{spline}(\boldsymbol{v}_i)$ の微分が定義できないために輪郭モデル自体を消滅させる。

収束判定として、制御点の移動した個数が C_{TH} 以下であり、新たな制御点の生成・消滅がないとき、または更新回数 t が t_{max} 以上になった場合に、動的輪郭モデルの更新を終了して抽出が完了したとする。

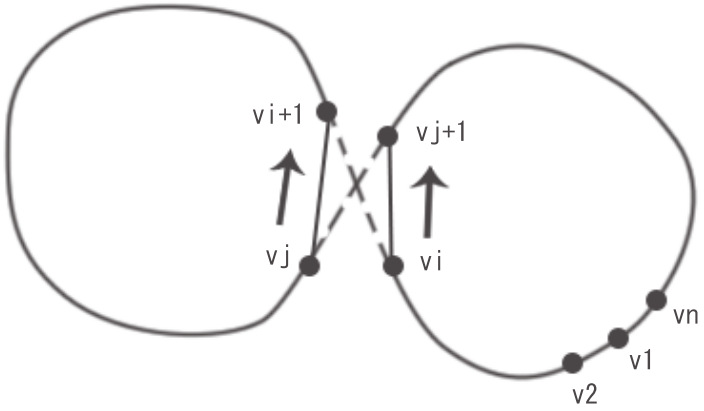

図 2.3: 交差した動的輪郭モデルの分裂

抽出手順

この手法における抽出手順を以下に示す。

[STEP1] 初期輪郭の設定

制御点が n 個の閉曲線 $\boldsymbol{v}_i(x_i(t), y_i(t))(i = 1, 2, ..., n)$ を、$t = 0$ として画像中の対象物全てを包含するように設定する。また、生成と消滅の閾値 D_{TH}、θ_{TH}、収束判定の閾値 C_{TH}、t_{max} を設定する。

[STEP2] 輪郭モデルの変形

全制御点の位置を式 (2.8) とグリーディアルゴリズム [6] によって、1 回更新 $t \leftarrow t+1$ とし、それにより移動した制御点の個数を C_{move} とする。

[STEP3] 輪郭モデルの交差判定

全制御点について、式 (2.9) を解いて $0 \leq p \leq 1$、$0 \leq q \leq 1$ を満たす場合は、交差があるので STEP4、それ以外の場合は STEP5 を行う。

[STEP4] 輪郭モデルの分裂

交差している線分 $\boldsymbol{v}_i\boldsymbol{v}_{i+1}$ と $\boldsymbol{v}_j\boldsymbol{v}_{j+1}$ について、$\boldsymbol{v}_i$ と $\boldsymbol{v}_{j+1}$、$\boldsymbol{v}_{i+1}$ と $\boldsymbol{v}_j$ を連結し輪郭モデルを 2 つにして、どちらも制御点の順番を時計回りになるようにする。

[STEP5] 制御点の生成・消滅判定

$|\boldsymbol{v}_{i+1} - \boldsymbol{v}_i| > D_{TH}$ を満たす場合、$\boldsymbol{v}_{i+1}$、$\boldsymbol{v}_i$ 間に新たな制御点を生成する。また、$cos\theta > \theta_{TH}$ を満たす場合、$\boldsymbol{v}_i$ を消滅させる。

[STEP6] 抽出完了判定 $C_{move} \geq C_{TH}$ かつ制御点の生成、消滅がなかったとき、または $t \leq t_{max}$ のとき、抽出できたとして終了する。それ以外の場合は、STEP2 に戻る。

問題点

この手法の問題点として、エッジが弱い部分では対象物の輪郭に沿わずに対象物の内部に入り込むことがあり、内部のより強いエッジに沿ってしまう。内部に入り込んだ輪郭をそのまま抽出に用いると対象物の一部が欠けてしまうことがある。また、面積項によって凹の形になれることにより対象物の内部に入りやすくなる。

この様子を図 2.4 に示す。図 2.4 では、黒い線が輪郭、黄色と水色の境界が強いエッジ、背景と物体の境界が弱いエッジとする。強いエッジのほうがエネルギーが大きいために、輪郭が内部に引っ張られていることを示している。

理想として対象物全体がほしいので、出来る限り欠けないように改良する必要がある。

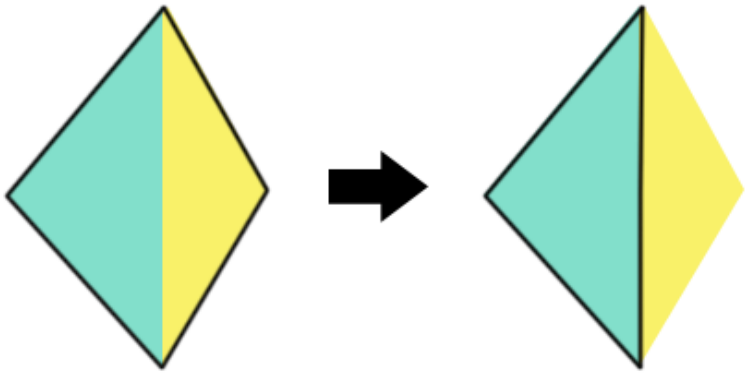

図 2.4: 従来手法の問題点

2.3 提案手法

2.3.1 分裂する動的輪郭モデルと領域分割を用いた対象物抽出

概要

イラスト画像について、経験的に写真に比べるとほぼ同じ色が固まって存在する。このことから局所的な特徴として色を用いて領域分割を行えば、対象物の輪郭に沿った領域を得ることができると考えられる。しかし、分割された領域が対象物か背景か、対象物または背景が分割されたものかどうかがわからない。この分割された領域と分裂する動的輪郭モデルで抽出した輪郭線を組み合わせれば、動的輪郭モデルによって対象物が欠ける問題をなくしつつ、領域が対象物かどうかを判断することが可能となると考えられる。

2.2.1 で述べた問題点を解決するために、従来手法に領域分割を加えた抽出手法について示す。

領域分割は Hill-Climb を用いた手法を用いる。この手法は、事前に決めるパラメータ数が少なくて済み、クラスタ数を決める必要がなく、予備知識の必要もないので、自動化するのに最適であると思われる。また、単純なアルゴリズムなので高速と謳われている。

領域が対象物かどうかの判断には、輪郭線内部に領域がどれだけの面積で入っているかを調べて領域全体の面積との比で判断すればよい。

分裂する動的輪郭モデルの説明は 2.2.1 で示している。以下に、Hill-Climb を用いた領域分割と抽出手順を示す。

Hill-Climb を用いた領域分割

本提案手法では、Hill-Climb を用いた領域分割 [9] を用いて領域分割を行う。この領域分割手法は、HSV 色空間で H、S、V それぞれのヒストグラムを求めて、ヒストグラムをグラフで表したときに頂上となる部分（ピーク）を調べる。調べたピークにヒストグラムを属させて、ピークの数だけ領域ができるようにするものである。

パラメータの数は H、S、V を分割するための量子数だけであり、色のヒストグラムから求めるので予備知識が必要ではない。また、ピークの数が領域となるので事前に領域の数を決める必要がない。

以下に、この分割手法のアルゴリズムを示す。

[STEP1]

入力画像を HSV 色空間にする。H、S、V それぞれについて、あらかじめ決めた量子数に分割しヒストグラムを計算する。

[STEP2]

H、S、V それぞれのヒストグラムを、H、S、V の順で STEP3 と STEP4 の処理を行う。V の処理が完了したとき、STEP5 を行う。

[STEP3]

移動が行われていないヒストグラム、かつ、ヒストグラムのピクセル数が 0 でないものを始点とし、現在位置を p する。

[STEP4]

図 2.5 にこのステップの様子を示す。グラフはヒストグラムであり、矢印は p の移動を示している。

p と隣り合うヒストグラムのピクセル数を比較して、大きい方に p を移動させる。同じ場合は、隣り合うヒストグラムのさらに隣のヒストグラムと比較し、大きい方に p を移動させる。p の隣り合うヒストグラムが両方とも小さい場合、その位置記録して始点から p までのヒストグ

ラムをそのピークに属させる。すべてのヒストグラムについて移動が行われたかどうかを調べる。行われたとき、STEP2 に戻る。それ以外の場合、STEP3 に戻る。
[STEP5]
入力画像のピクセルと対応するピークによって、画像を分割して終了する。

抽出手順

本提案手法における抽出手順を以下に示す。
[STEP1]
入力画像について、分裂する動的輪郭モデルを用いて輪郭抽出を行う。輪郭抽出後の輪郭の数を N_{snakes} として、輪郭を $S_i(i = 1, 2, ..., N_{snakes})$ とする。
[STEP2]
入力画像について、Hill-Climbing を用いた領域分割を行う。分割された領域の数を N_{area} として、分割された領域を $a_j(j = 1, 2, ..., N_{area})$ とする。
[STEP3]
$i = 0$、$j = 0$ とする。
[STEP4]
S_i の内部に入っている a_j の面積を求めて、その面積が a_j の面積の閾値 P_{TH} 以上の場合、その領域を抽出して STEP6 を行う。それ以外の場合は STEP5 を行う。
[STEP5]
$i \leftarrow i + 1$ とする。$i > N_{snakes}$ のとき、$j \leftarrow j + 1$ とする。その後、STEP6 を行う。
[STEP6]
$j > N_{area}$ の場合、全ての a_j について判定したとなるので抽出完了とする。それ以外の場合、$i = 0$ として STEP4 に戻る。

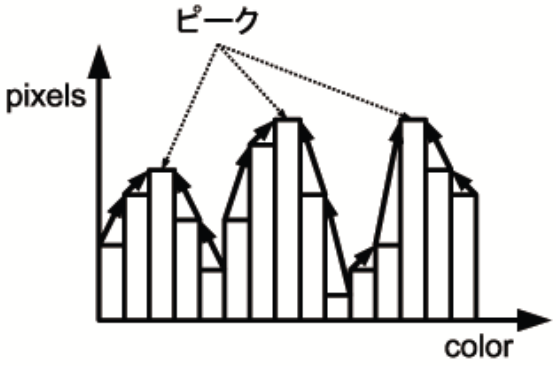

図 2.5: ピークを特定する様子

2.3.2　変分ベイズ法による混合正規分布推定と領域分割を用いた対象物抽出

概要

2.2.1 で述べた問題点を解決とイラスト画像に対する対象物抽出のために、動的輪郭モデルからアプローチを変えたは領域分割を用いた対象物抽出の一種である手法について示す。

2.3.1 で示したように、領域分割だけでは分割された領域は、対象物の領域か背景の領域かがわからない。2.3.1 では、分裂する動的輪郭モデルを用いて領域が対象物かどうかを判断していた。本手法では、分裂する動的輪郭モデルの代わりに変分ベイズ法による混合正規分布推定[13) 16)]を用いる。混合正規分布の推定はパターン認識等で用いられるのが一般的である。ここでは、対象物を特定できそうな特徴量を使って分布の推定を行い、その分布をそのまま領域にできるのではないかと考える。

特徴量はエッジではなくコーナーを用いて対象物を推定する。コーナーとは、画像の特徴をよく表す点または角を表すものである。コーナーを検出した画像を図 2.6 に示す。エッジではデータ点が多すぎでありノイズも考えられる。コーナーならエッジの重要な部分だけとなり、ノイズも少なくなる。

動的輪郭モデルでは不必要な対象物ではない部分にエッジが多く含まれている場合、輪郭線が不必要な部分に沿ってしまいうまく抽出できない。この手法では、そのような不必要な部分について、エッジに頼らないので動的輪郭モデルよりも高い精度で抽出できると考えられる。

領域分割で用いる Hill-Climb を用いた領域分割については、2.3.1 で説明している。以下に変分ベイズ法の基礎とそれを用いた混合正規分布の説明を示し、それと領域分割を用いた抽出手順について示す。

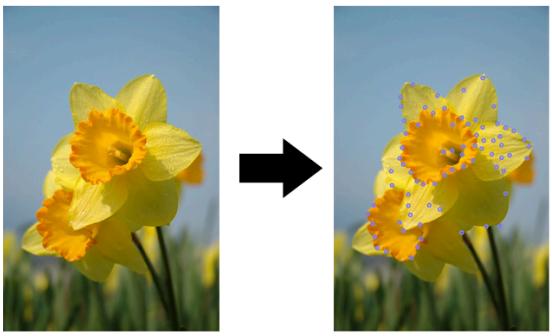

図 2.6: コーナー検出

ベイズ学習

ベイズ学習[11)15)]とは、ほしい確率分布の未知パラメータをそれに関する確率分布を考え、未知パラメータの不確定性を表現してほしい確率分布を推定するものである。未知パラメータの確率分布は、データが観測される前を事前分布、その後を事後分布と呼ぶ。学習データ D、事前分布 $p(\theta)$、事後分布 $p(\theta|D)$ として、D が観測された後の事後分布 $p(\theta|D)$ はベイズの定理

より以下のようになる。

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \tag{2.10}$$

$$p(D) = \int p(D|\theta)p(\theta)d\theta \tag{2.11}$$

またベイズ学習では、事後分布 $p(\theta|D)$ を用いて分布を予測することが出来る。これを予測分布と呼び、未知データ x^* として次式で定義される。

$$p(\boldsymbol{x}^*|D) = \int p(\boldsymbol{x}^*|\theta)p(\theta|D)d\theta \tag{2.12}$$

式 (2.12) は、仮説 $p(\boldsymbol{x}^*|\theta)$ を事後分布 $p(\theta|D)$ で重み付き平均したものである。これにより、$\boldsymbol{x}^*$ が予測分布に入るか否かを確率的に予測することができる。

式 (2.11) について、右辺の分母は θ によらない、および、$-\log p$ が情報量であることから、次式のようにできる。

$$-\log p(\theta|D) \propto -\log p(D|\theta) - \log p(\theta) \tag{2.13}$$

式 (2.13) は、

事後の情報量$\propto$学習データの情報量 + 事前の情報量

といえ、事前の情報量に学習データの情報量が付加されて事後の情報量となっており、直感的に自然な情報論的解釈ができる。

変分ベイズ法

式 (2.12) は、一般に解析的に求めることが困難である。したがって、何らかの近似が必要となる。ここで変分法を用いて式 (2.12) の近似を行うのが、変分ベイズ法 [12] である。

変分法 [14] とは、関数を関数の関数 (汎関数) によって関数を近似する手法である。微分法と似ており、微分法では変数を変化させて極値を求めるのに対して、変分法では汎関数を変化させて極値を求める。

変分法を用いて導くための汎関数を導出を説明してアルゴリズムを示す。

潜在変数 Z、混合分布モデルにおける混合要素数 m として、すべての未知量を周辺化した周辺尤度 $L(D)$ は次式で表される。

$$L(D) = \log p(D) = \log \sum_m \sum_z \int p(D, Z, \theta, m)d\theta \tag{2.14}$$

新たな分布 q を導入して対数関数に対する Jensen の不等式を適用すると、$L(D)$ の下限値 $F[q]$ が得られる。

$$F[q] = \sum_m \sum_z \int q(Z, \theta, m) \log \frac{p(D, Z, \theta, m)}{q(Z, \theta, m)} d\theta \tag{2.15}$$

$L(D) - F[q]$ より、カルバック・ライブラダイバージェンス $\mathrm{KL}(q(Z, \theta, m|D), p(Z, \theta, m|D))$ を求めることができ、次式の関係となる。

$$L(D) = F[q] + \mathrm{KL}(q(Z, \theta, m|D), p(Z, \theta, m|D)) \tag{2.16}$$

式 (2.16) について、$F[q]$ を最大化することは KL を最小化する事と同義であり、KL を最小化することで真の事後分布の最良の近似となる。ここで $F[q]$ を汎関数と考えると、$F[q]$ の極値問題となって変分法を用いて $L(D)$ を近似することができる。

$p(D, Z, \theta, m)$ を分解した式を示し、$q(Z, \theta, m)$ を未知パラメータごとに分解した形を仮定する。

$$p(D, Z, \theta, m) = p(m)p(D, Z|m)\prod_i p(\theta_i|m) \tag{2.17}$$

$$q(Z, \theta, m) = q(m)q(Z|m)\prod_i q(\theta_i|m) \tag{2.18}$$

式 (2.18)、(2.18) を式 (2.15) に代入して、$q(Z|m)$、$q(\theta_i|m)$ を求めると、以下の式を得る。ただし、表記 $\langle f(x)\rangle_{p(x)}$ は $f(x)$ の $p(x)$ に関する期待値である。

$$q(Z|m) = C\exp\langle\log p(D, Z|\theta, m)\rangle_{q(\theta|m)} \tag{2.19}$$

$$q(\theta_i|m) = C'p(\theta_i|m)\exp\langle\log p(D, Z|\theta, m)\rangle_{q(Z|m),q(\theta_{-1}|m)} \tag{2.20}$$

C は $\sum_Z q(Z|m) = 1$ となるための定数、C' は $\int q(\theta_i|m)d\theta_i = 1$ となるための定数である。また、θ_{-1} は θ_i 以外のパラメータ集合を示す。式 (2.20)、(2.20) は相互に依存関係があるので、反復アルゴリズムによって逐次推定する。反復ステップ数を t として、アルゴリズムを以下に示す。

[STEP1]
分布 $q(\theta|m)^{(0)} = \prod_i p(\theta_i|m)^{(0)}$ を設定して、$t \leftarrow 0$ とする。
[STEP2] 以下を収束または更新回数が t_{max} になるまで繰り返す。
式 (2.20) より、

$$q(Z|m)^{(t+1)} = C\exp\langle\log p(D, Z|\theta, m)\rangle_{q(\theta|m)^{(t)}}$$

式 (2.20) より $i = 1, ..., I$ について、

$$q(\theta_i|m)^{(t+1)} = C'p(\theta_i|m)\exp\langle\log p(D, Z|\theta, m)\rangle_{q(Z|m)^{(t+1)},q(\theta_{-1}|m)^{(t)}}$$

を計算して、$t \leftarrow t + 1$ とする。

変分ベイズ法による混合正規分布推定

m 個の要素数を持つ d 次元の混合正規分布の確率密度関数は次式で表される。

$$p(\boldsymbol{x};\theta) = \sum_{i=1}^{m}\alpha_i N(\boldsymbol{x};\boldsymbol{\mu}_i, \boldsymbol{S}_i^{-1}) \tag{2.21}$$

$$N(\boldsymbol{x};\boldsymbol{\mu}, \boldsymbol{S}_i^{-1}) = (2\pi)^{-\frac{d}{2}}|\boldsymbol{S}|^{\frac{1}{2}}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{S}(\boldsymbol{x}-\boldsymbol{\mu})\right\} \tag{2.22}$$

ただし、入力ベクトル $\boldsymbol{x}$、平均ベクトル $\boldsymbol{\mu}$、共分散行列の逆行列である精度行列 $\boldsymbol{S}$ である。
$\boldsymbol{\alpha} = \{\alpha_i\}_{i=1}^m$、$\boldsymbol{\mu} = \{\boldsymbol{\mu}_i\}_{i=1}^m$、$\boldsymbol{S} = \{\boldsymbol{S}_i\}_{i=1}^m$ として、未知パラメータ θ の結合分布は以下にように分解できる。

$$p(\theta) = p(m)p(\boldsymbol{\alpha}|m)p(\boldsymbol{S}|m)p(\boldsymbol{\mu}|\boldsymbol{S}, m) \tag{2.23}$$

式 (2.20)、(2.20) を式 (2.23) に適用すると $p(\boldsymbol{\alpha}|\boldsymbol{m})$、$p(\boldsymbol{S}|m)$、$p(\boldsymbol{\mu}|\boldsymbol{S}, m)$ を求める事が出来る。
導出した結果を整理して、混合正規分布の推定のために変分ベイズ法を適用したアルゴリズムは、以下の通りとなる。
[STEP1] 初期化

事前分布の超パラメータ ϕ_0、ξ_0、η_0、$\boldsymbol{\nu}_0$、$\boldsymbol{B}_0$ と $\bar{N}_i^{(0)} \leftarrow \frac{N}{m}$ を設定する。その後、事後分布の超パラメータを以下のように設定する。

$i = 1, ..., m$ に対して、

$$
\begin{aligned}
&\phi_i^{(0)} \leftarrow \phi_0 \\
&\bar{\boldsymbol{\mu}_i}^{(0)} \leftarrow \boldsymbol{\nu_0} \\
&\eta_i^{(0)} \leftarrow \eta_0, \boldsymbol{B}_i^{(0)} \leftarrow \boldsymbol{B}_0 \\
&f_{\boldsymbol{\mu}_i}^{(0)} \leftarrow \eta_0 + \bar{N}_i^{(0)} + 1 - d \\
&\Sigma_{\boldsymbol{\mu}_i}^{(0)} \leftarrow \frac{\boldsymbol{B}_i^{(0)}}{(\bar{N}_i^{(0)} + \xi_0) f_{\boldsymbol{\mu}_i}^{(0)}}
\end{aligned}
$$

[STEP2] 潜在変数の事後分布の更新

$i = 1, ..., m$、$n = 1, ..., N$ に対して、次の計算を行う。

$$
\begin{aligned}
\bar{z}_i^n &= \frac{\exp \gamma_i^n}{\sum_{j=1}^m \exp \gamma_j^n} \\
\gamma_i^n &\leftarrow \Psi(\phi_0 + \bar{N}_i^{(t)}) - \Psi(m\phi_0 + \sum_{i=1}^m \bar{N}_i^{(t)}) \\
&+ \frac{1}{2} \sum_{j=1}^d \Psi(\frac{\eta_0 + \bar{N}_i^{(t)} + 1 - j}{2}) - \frac{1}{2} \log |\boldsymbol{B}_i^{(t)}| \\
&- \frac{1}{2} \mathrm{Tr} \left\{ (\eta_0 + \bar{N}_i^{(t)}) (\boldsymbol{B}_i^{(t)})^{-1} \left(\frac{f_{\boldsymbol{\mu}_i}^{(t)}}{f_{\boldsymbol{\mu}_i}^{(t)} - 2} \Sigma_{\boldsymbol{\mu}_i}^{(t)} + (\boldsymbol{x}_n - \bar{\boldsymbol{\mu}_i}^{(t)}) ((\boldsymbol{x}_n - \bar{\boldsymbol{\mu}_i}^{(t)})^T \right) \right\}
\end{aligned}
$$

[STEP3] パラメータの事後分布の更新

以下の計算を行う。

$$
\begin{aligned}
\bar{N}_i^{(t)} &\leftarrow \sum_{n=1}^N \bar{z}_i^n \\
\bar{\boldsymbol{x}}_i^{(t)} &\leftarrow \sum_{n=1}^N \boldsymbol{x}_n \\
\boldsymbol{C}_i^{(t)} &\leftarrow \sum_{n=1}^N \bar{z}_i^n (\boldsymbol{x}_n - \bar{\boldsymbol{x}}_i^{(t)}) (\boldsymbol{x}_n - \bar{\boldsymbol{x}}_i^{(t)})^T \\
\phi_i^{(t)} &\leftarrow \phi_0 + \bar{N}_i^{(t)} \\
\eta_i^{(t)} &\leftarrow \eta_0 + \bar{N}_i^{(t)} \\
\bar{\boldsymbol{\mu}}_i^{(t)} &\leftarrow \frac{\bar{N}_i^{(t)} \bar{\boldsymbol{x}}_i^{(t)} + \xi_0 \boldsymbol{\nu}_0}{\bar{N}_i^{(t)} + \xi_0} \\
f_{\boldsymbol{\mu}_i}^{(t)} &\leftarrow \eta_i^{(t)} + 1 - d \\
\boldsymbol{B}_i^{(t)} &\leftarrow \boldsymbol{B}_0 + \bar{\boldsymbol{C}}_i^{(t)} + \frac{\bar{N}_i^{(t)} \xi_0}{\bar{N}_i^{(t)} + \xi_0} (\bar{\boldsymbol{x}}_i^{(t)} - \boldsymbol{\nu}_0) (\bar{\boldsymbol{x}}_i^{(t)} - \boldsymbol{\nu}_0)^T \\
\Sigma_{\boldsymbol{\mu}_i}^{(t)} &\leftarrow \frac{\boldsymbol{B}_i^{(t)}}{(\bar{N}_i^{(t)} + \xi_0) f_{\boldsymbol{\mu}_i}^{(t)}}
\end{aligned}
$$

[STEP4]

$t \leftarrow t+1$ とする。収束したとき終了し、それ以外の場合は STEP2 に戻る。
このアルゴリズムにより、事後分布を求めることが出来る。
事後分布を求めた後、予測分布を求める。式 (2.12) より、

$$p(\boldsymbol{x}^*|D,m) = \sum_{i=1}^{m} \langle \alpha_i \rangle_{q(\boldsymbol{\alpha}|m)} \left\langle N(\boldsymbol{x}^*; \boldsymbol{\mu}_i, \boldsymbol{S}_i^{-1}) \right\rangle_{q(\boldsymbol{\mu}_i, \boldsymbol{S}_i|m)} \tag{2.24}$$

となる。

抽出手順

変分ベイズ法による混合正規分布推定を用いた対象物抽出の手順について示す。
[STEP1]
混合正規分布の混合数 m を事前に決めておく。入力画像からコーナー点を検出して座標を 0.0 から 1.0 の範囲にする。
[STEP2]
コーナーの座標点をデータ点として、前述の変分ベイズ法による混合正規分布の手法で事後分布を求め、予測分布を計算できるようにする。入力画像の幅と高さをそれぞれ w、h として、座標 (0,0) から (w-1,h-1) まで未知データとして予測分布を計算して、そのときの値が閾値 P_{TH} 以上ならば対象物の領域候補とする。
[STEP3]
入力画像を Hill-Climbing を用いた領域分割をする。この領域分割手法のパラメータは事前に決めておく。
[STEP4]
STEP2 で分割した領域について、STEP1 で求めた分布領域に入っている面積を求める。このときの面積とその領域全体の面積の比が PA_{TH} 以上ならば対象物の領域として抽出する。
図 2.7 にこのステップの様子の例を示す。分割された領域 0 から 2 があり、青い線で示した推定した分布領域をそれに重ねる。PA_{TH} が 0.5 として、領域 1 以外は領域が分布領域の半分以上と重なっているため、重なった領域の面積と領域全体の面積の比は 0.5 以上であるといえ、領域 1 以外を抽出する。
抽出処理が完了したら終了する。

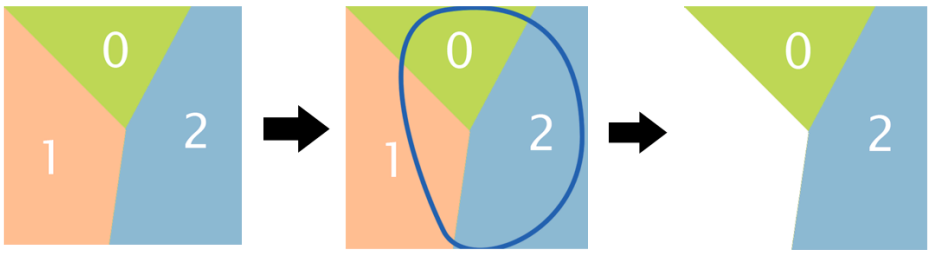

図 2.7: 変分ベイズ法を用いた提案手法における抽出手順

第3章　実験

3.1　実験方法

従来手法と 2 つの提案手法を比較、検討するために動的輪郭モデルと変分ベイズ法それぞれ領域を決定するプログラム、Hill-Climb を用いた領域分割を行うプログラムと領域分割しないプログラムを作りプラグインとする。入力画像とプラグイン化した手法をプログラムに入力すると抽出結果を出力するプログラムを作成した。ライブラリは OpenCV-1.1pre、コンパイラ、リンカは VisualC++2008Standard を用いた。動的輪郭モデル、変分ベイズ法のプログラムはともに自身で書いたコードに関してマルチスレッドのような最適化は行っていない。

3.2　パラメータの決定

3.2.1　Hill-Climb を用いた領域分割のパラメータ

2 つの提案手法で共に使う Hill-Climb を用いた領域分割のパラメータを決定する。

2.3.1 の Hill-Climb を用いた領域分割で示したように、パラメータは H、S、V それぞれの量子数である。

パラメータが大きすぎると領域の数が多くなりすぎてノイズや意味のない領域が取り込まれやすくなり、逆に小さすぎると分かれてほしい領域が一緒になってしまう。この手法の論文 [9] では、色の多様性を確保するために H を S と V に比べて量子数を多くしており、H、S、V それぞれ 16、8、8 としている。

しかし、イラスト画像において、この数値をそのまま使うとあまりうまく領域分割ができなかった。これは、イラスト画像では H だけでなく S と V も重要で、H の値が同じでも S の値または V の値が違うことがあるからである。2 つの提案手法において、先に対象物の領域を決めているために対象物の中の領域数が多少多くなったとしても問題はないと思われる。ノイズを拾わない程度により細かく分割したいので、論文の値を元に S と V の量子数を上げて 16、16、16 とした。

3.2.2　分裂する動的輪郭モデルのパラメータ

従来手法と提案手法で用いる分裂する動的輪郭モデルのパラメータを決定する。

2.2.1 で示したように、必要なパラメータは初期輪郭の制御点の個数 n、更新回数の最大値 t_{max}、移動した制御点の個数 C_{TH}、制御点を追加するための閾値 D_{TH}、制御点を削除するための閾値 θ_{TH} エネルギー関数の重み w_{sp1}、w_{sp2}、w_{area}、w_{dist}、w_{edge} である。以下に、実験で用いるそれぞれのパラメータの値を示す。

$$n = 300$$
$$t_{max} = 1000$$

$$
\begin{aligned}
C_{TH} &= 10 \\
D_{TH} &= 50.0 \\
\theta_{TH} &= 0.999 \\
w_{sp1} &= 1.0 \\
w_{sp2} &= 1.0 \\
w_{area} &= 3.0 \\
w_{dist} &= 1.0 \\
w_{edge} &= 2.0
\end{aligned}
$$

イラスト画像には、凹の形をした部分とエッジが多くあると考えられるためにエネルギー関数の重みである面積項のパラメータ w_{area} とエッジのエネルギーのパラメータ w_{edge} を他のパラメータよりも強くしている。

3.2.3 変分ベイズ法を用いた混合正規分布推定のパラメータ

提案手法で用いる変分ベイズ法を用いた混合正規分布推定のパラメータを決定する。コーナーの検出には、OpenCV の cvGoodFeaturesToTrack を使用している。

コーナー検出のパラメータは、許容最低品質、許容最低距離、平均化ブロックのサイズである。変分ベイズ法のパラメータは、ϕ_0、ξ_0、η_0、$\boldsymbol{\nu}_0$、$\boldsymbol{B}_0$、t_{max} である。以下に、実験に用いるパラメータの値を示す。

- コーナー検出のパラメータ

$$
\begin{aligned}
\text{許容最低品質} &= 0.01 \\
\text{許容最低距離} &= 10 \\
\text{平均化ブロックのサイズ} &= 3
\end{aligned}
$$

- 変分ベイズ法のパラメータ

$$
\begin{aligned}
m &= 7 \\
\phi_0 &= \frac{N}{m} \\
\xi_0 &= 1 \\
\eta_0 &= d + 2 \\
\boldsymbol{\nu}_0 &= \hat{\boldsymbol{x}}_i \\
\boldsymbol{B}_0 &= \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i)^2 \boldsymbol{I}
\end{aligned}
$$

変文ベイズ法のパラメータは、以下のように決めた。[17) 18)]

- ϕ_0：N_i の初期値と同程度の値が妥当である。
- ξ_0：分母が無意味に大きくなるのを防ぐ。
- η_0：$\Gamma(\log((\eta_0 + 1 - d)/2))$ という式が含まれており、$\Gamma(0)$、Γ(負の値) とならないようにしている。

- $\boldsymbol{\nu_0}$：真の平均ベクトルと大幅に異なると、尤度が極端に低くなって学習できなくなるために、ある程度データ点に近づけた値を用いている。$\hat{\boldsymbol{x}}_i$ は、KMeans 法で変分ベイズ法のクラスタ数 m と同じクラスタ数でデータ点をクラスタリングして求めた値であり、ランダムに決めた値に比べると真の平均ベクトルにより近い値になっていると考えられるので、これを初期値としている。

- $\boldsymbol{B}_0$：値が大きすぎると $\boldsymbol{B}_i$ が $\boldsymbol{B}_0$ に依存して学習を妨げるので、データ点と平均ベクトルの分散を用いて学習の妨げになりにくいと思われる値にしている。

3.3 抽出画像

従来手法、分裂する動的輪郭モデルと領域分割を用いた手法、変分ベイズ法と領域分割を用いた手法で対象物抽出を行った結果を示す。図 3.1、図 3.2 ともに左上が元画像、右上が従来手法の結果、左下が分裂する動的輪郭モデルと領域分割を用いた手法の結果、右下が変分ベイズ法と領域分割を用いた手法の結果である。

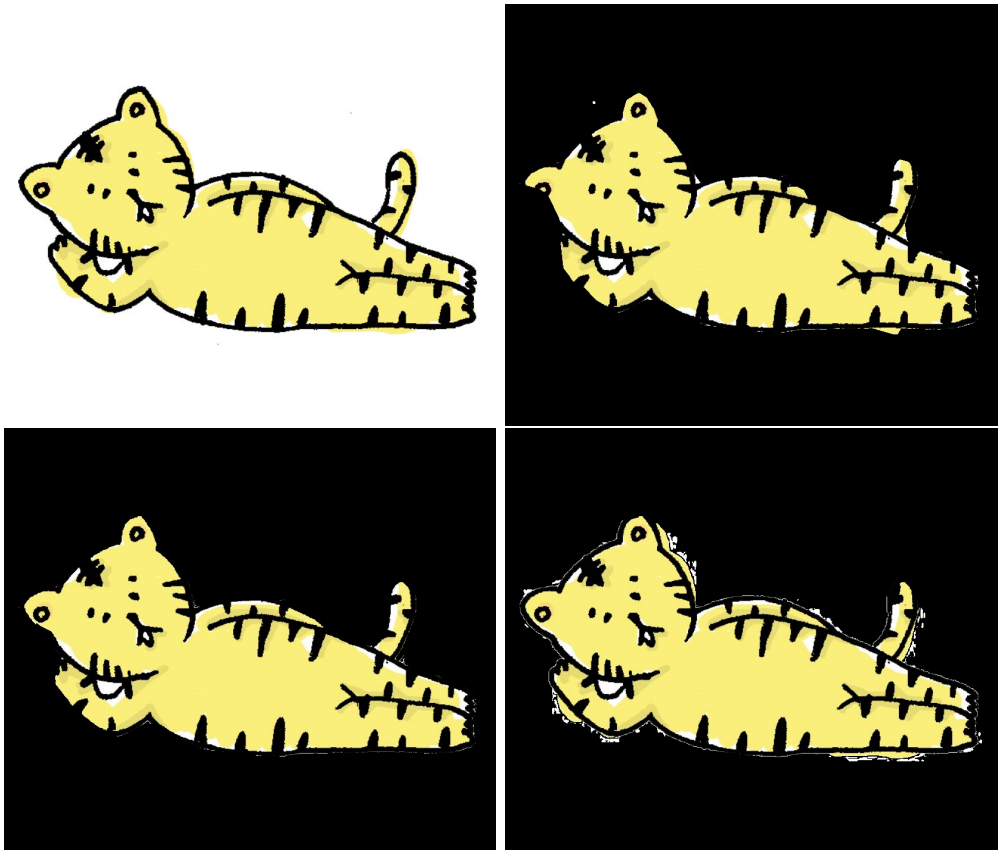

図 3.1: 単純な背景のイラスト画像抽出結果

図 3.1 について、背景が 1 色で単純な背景のイラスト画像とする。従来手法で抽出した結果は対象物が少し欠けてしまっているのに対して、分裂する動的輪郭モデルと領域分割を用いた手法では、その欠けた部分をうまく抽出できていることがわかる。一方、変分ベイズ法と領域分割を用いた手法では背景が少し抽出結果に現れてしまっている。したがって、このイラスト画像の場合、分裂する動的輪郭モデルと領域分割を用いた手法が一番良いと考えられる。

図 3.2: 複雑な背景のイラスト画像抽出結果

図 3.2 について、背景に対象物と別の模様が含まれているためにこれを複雑な背景のイラスト画像とする。このとき対象物は人物である。従来手法では、対象物のほぼ全体が含まれているものの模様に沿って抽出してしまっている。分裂する動的輪郭モデルと領域分割を用いた手法では、従来手法よりは模様の部分がなくなっているもののまだ大きく含まれている。変分ベイズ法と領域分割を用いた手法では、欠けてしまっている部分があるものの他の手法に比べて対象物だけになっている。したがって、このイラスト画像の場合、変分ベイズ法と領域分割を用いた手法が一番良いと考えられる。

3.4 性能比較

従来手法、分裂する動的輪郭モデルと領域分割を用いた手法、変分ベイズ法と領域分割を用いた手法の性能について検討する。本実験では、単純な背景と複雑な背景を区別したイラスト画像をそれぞれ 13 個、9 個用意してプログラムを実行し抽出を行う。手作業で抽出した正解の画像とそれぞれの手法を適用したプログラムによって抽出した画像を、0 から 1 の範囲で 1 を正解の画像と完全一致として比較し、正答率を求めた。ここで、単純な背景とは、背景が 1 色だけのものや単純な図形が 1 個程度存在する背景、複雑な背景とは、背景にエッジが多く含まれていたりする単純な背景以外の背景とする。

図 3.3、図 3.4 について、横軸はデータの番号、縦軸は正答率を示す。赤の+点は従来手法、緑の×点は分裂する動的輪郭モデルと領域分割を用いた手法、青い*印は変分ベイズ法と領域分割を用いた手法を示す。データを分かりやすくするために、変分ベイズ法と領域分割を用いた手法について昇順で並び替えている。

単純な背景のイラスト画像について検討する。図 3.3 より、従来手法と分裂する動的輪郭モ

デルと領域分割を用いた手法を比較すると、この提案手法はどのデータでも従来手法より高い値を示しており、この提案手法は単純な背景のイラスト画像において、従来手法より高い精度で抽出可能であるといえる。また、変分ベイズ法と領域分割を用いた手法とそれ以外の手法を比較すると、分裂する動的輪郭モデルと領域分割を用いた手法のほうが高い正答率を示すことが多いが、従来手法よりも高い正答率を示すことが多い。したがって、単純な背景のイラスト画像において、分裂する動的輪郭モデルと領域分割を用いた手法が一番精度よく抽出できるといえる。また、変分ベイズ法と領域分割を用いた手法を比較すると、従来手法に比べると精度が高く、分裂する動的輪郭モデルと領域分割を用いた手法よりは精度が低いといえる。

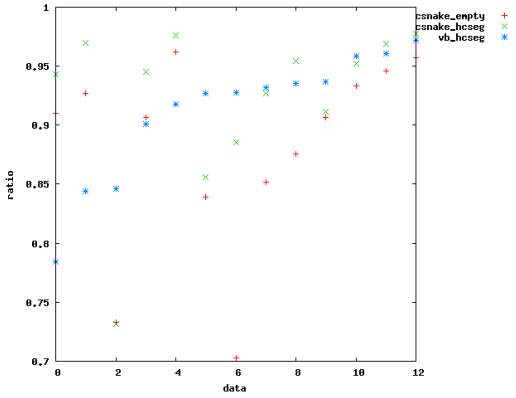

図 3.3: 単純な背景のイラスト画像における正答率

複雑な背景のイラスト画像について検討する。図 3.4 より、単純な背景のイラスト画像に比べると全体的に低い正答率となっている。各手法を見ると、変分ベイズ法と領域分割を用いた手法が他の手法に比べて目立って高い正答率になっている。従来手法と分裂する動的輪郭モデルと領域分割を用いた手法について、従来手法のほうが高いまたは従来手法とほぼ等しい正答率となっており、分裂する動的輪郭モデルと領域分割を用いた手法は従来手法に比べて精度が悪くなっている。したがって、複雑な背景のイラスト画像において変分ベイズ法と領域分割を用いた手法が最も精度が高く抽出できるといえる。

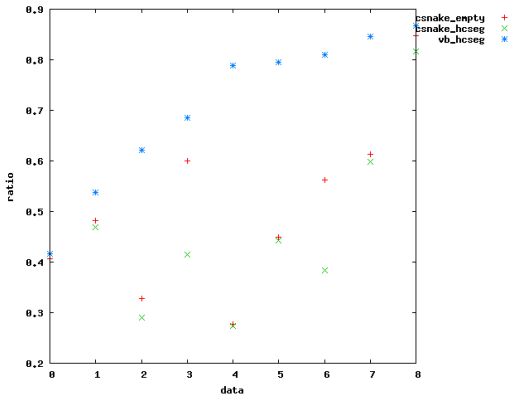

図 3.4: 複雑な背景のイラスト画像における正答率

3.5　処理速度比較

3.5.1　従来手法と分裂する動的輪郭モデルと領域分割を用いた手法の比較

従来手法である分裂する動的輪郭モデルだけを用いた手法と提案手法である分裂する動的輪郭モデルと領域分割を用いた手法の処理時間を検討する。本実験では、548 個のイラスト画像についてプログラムを実行し、それぞれの手法で抽出を行ってその処理時間を記録した。どちらの手法も分裂する動的輪郭モデルを使用しているために更新回数は同じになるので記録していない。図 3.5 は横軸に画像のピクセル数、縦軸にミリ秒単位の時間を取ったもので、全体のグラフとデータが集中している部分を拡大したグラフを示す。それぞれ従来手法を用いた手法は赤の＋点、提案手法は緑の×点を示す。

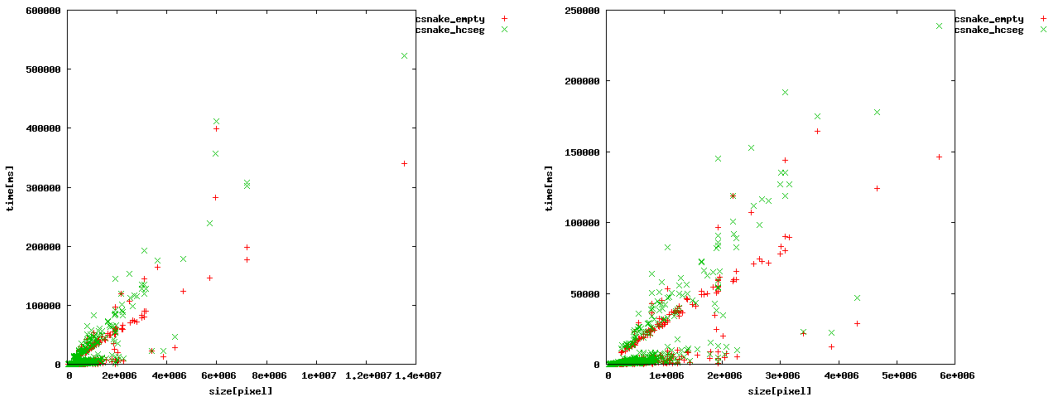

図 3.5: 従来手法と動的輪郭モデルと領域分割を用いた手法における画像のピクセル数と処理時間

図 3.5 より、ほとんどの場合、提案手法が従来手法よりも処理に時間がかかっている。これは、従来手法に領域分割処理を加えたためものが提案手法なので、提案手法の処理時間は領域分割の処理の時間分が従来手法の処理時間に加えられていたものと考えられる。

3.5.2　提案手法同士の比較

分裂する動的輪郭モデルと領域分割を用いた手法と変分ベイズ法と領域分割を用いた手法の処理時間を検討する。本実験では、3.5.1 と同様に、548 個のイラスト画像についてプログラムを実行し、それぞれの手法で抽出を行ってその処理時間と更新回数を記録した。図 3.6 は横軸に画像のピクセル数、縦軸にミリ秒単位の時間を取ったもの、図 3.7 は横軸に画像のピクセル数、縦軸に更新回数を取ったもので、全体のグラフとデータが集中している部分を拡大したグラフを示す。それぞれ分裂する動的輪郭モデルを用いた手法は赤の＋点、変分ベイズ法を用いた手法は緑の×点を示す。

図 3.6 と図 3.7 より、動的輪郭モデルは画像のピクセル数が 1000000 以下のとき、収束が速く処理している時間が短い。変分ベイズ法は画像のピクセルが同条件の場合、動的輪郭モデルに比べると収束が遅く、更新回数の最大値である 1000 回になることが多い。また、処理時間も動的輪郭モデルに比べると遅い。しかし、変分ベイズ法は画像のピクセル数が多くなっても処理時間の増大はあまり見られず、対数関数的な増大をしている。一方、動的輪郭モデルは画像のピクセル数に比例して処理時間が増えている。また、収束した場合は処理時間が短いが、収束せずに更新回数の最大値になった場合、変分ベイズ法より処理が遅くなっている。

上記より、画像のピクセル数が 1000000 以下の時は分裂する動的輪郭モデルのほうが収束が速いために変分ベイズ法より処理時間が短いことが多く、画像のピクセルが 1000000 より多い場合は分裂する動的輪郭モデルは収束しにくくなるので、変分ベイズのほうが分裂する動的輪郭モデルより安定していて処理時間が短くなることが多いといえる。

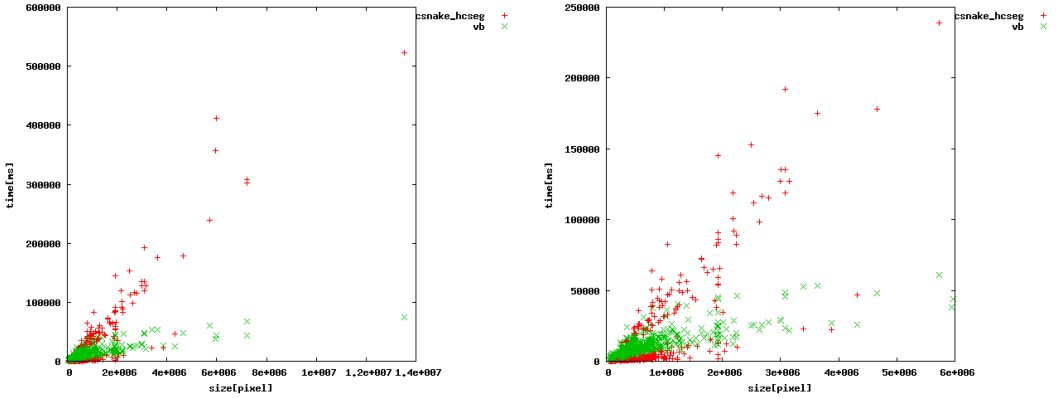

図 3.6: 提案手法同士の画像のピクセル数と処理時間

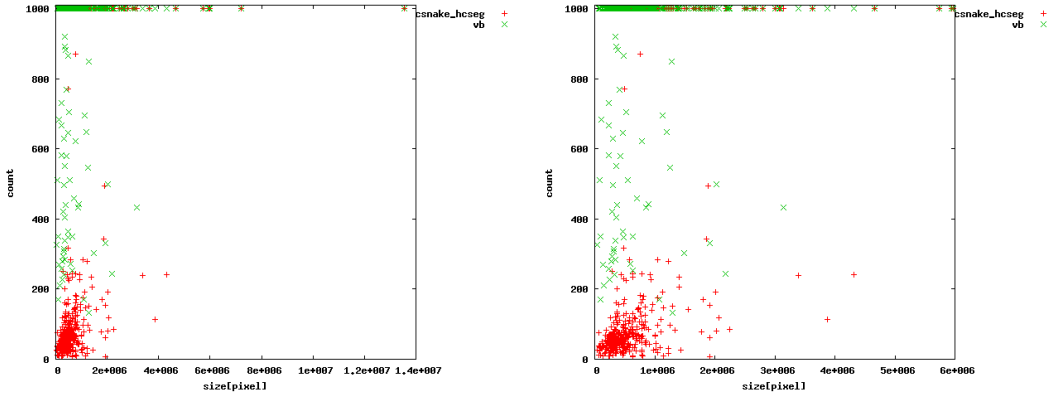

図 3.7: 提案手法同士の画像のピクセル数と更新回数

3.5.3 性能比較時のイラスト画像による各手法の比較

各手法において 3.4 で用いた画像を抽出したときの処理時間を検討する。本実験では、3.4 で用いた単純な背景のイラスト画像 13 個と複雑な背景のイラスト画像 9 個についてプログラムを実行し、それぞれの手法で抽出を行ってその処理時間と更新回数を記録した。図 3.8、図 3.9 は横軸に画像のピクセル数、縦軸にミリ秒単位の時間を取ったもの、図 3.10、図 3.11 は横軸に画像のピクセル数、縦軸に更新回数を取ったものである。更新回数に関して、従来手法と提案手法 1 では同じなので動的輪郭モデルと変文ベイズ法の比較とした。従来手法を赤の＋点、分裂する動的輪郭モデルと領域分割を用いた手法を緑の×点、変分ベイズ法と領域分割を用いた手法を青の*点を示す。

図 3.8、図 3.9 より、処理時間について 3.5.2 と同じ傾向を示している。また、図 3.10、図 3.11 より、更新時間は総じて動的輪郭モデルが変分ベイズ法よりも低い値を示している。ほぼすべて

のデータにおいて動的輪郭モデルは収束しているので、動的輪郭モデルの処理時間が短くなっていると思われる。

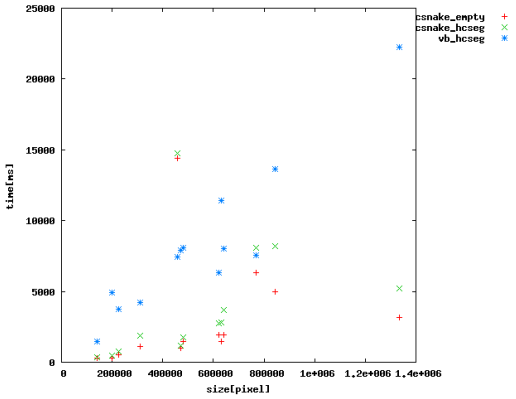

図 3.8: 単純な背景のイラスト画像のピクセル数と処理時間

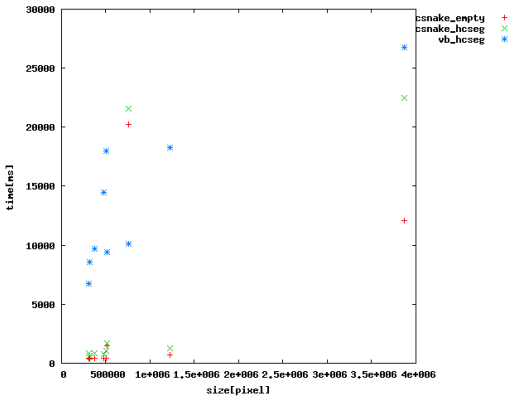

図 3.9: 複雑な背景のイラスト画像のピクセル数と処理時間

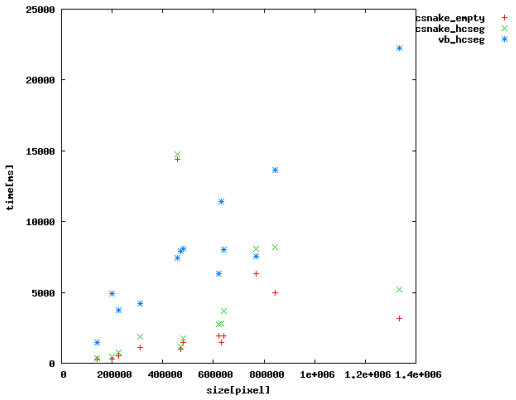

図 3.10: 単純な背景のイラスト画像のピクセル数と更新回数

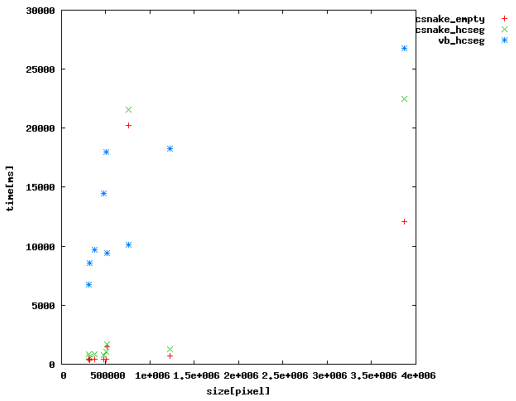

図 3.11: 複雑な背景のイラスト画像のピクセル数と更新回数

3.6 考察

3.3、3.4、3.5 から、イラスト画像から対象物を抽出するのに最適な手法を考える。

単純な背景のイラスト画像と複雑な背景のイラスト画像に対して、変分ベイズ法と領域分割を用いた手法が比較的安定した正答率で抽出ができると言える。しかし、画像が多く分布する画像のピクセル数が 1000000 以下の場合には処理が動的輪郭モデルを用いた手法に比べると長い。一方、動的輪郭モデルと領域分割を用いた手法は単純な背景のイラスト画像に対して変分ベイズ法より正答率が高く、画像のピクセル数が 1000000 以下の場合に処理時間も短い。しかし、複雑な背景のイラスト画像に対しては、従来手法の正答率が動的輪郭モデルと領域分割を用いた手法の正答率を上回ることがあり、正答率も低くなっている。

したがって、動的輪郭モデルと領域分割を用いた手法は、単純な背景のイラスト画像ならば短い時間で高い精度の対象物抽出ができると考えられる。一方、変分ベイズ法と領域分割を用いた手法は、処理に時間がかかるもののどんなイラスト画像でも他の手法と比べると対象物を抽出できると考えられる。

第4章　おわりに

本論文では、イラスト画像に対する対象物の自動抽出について提案を行った。また、従来手法について対象物の凹の部分に抽出する領域が食い込んでしまう問題を領域分割を用いることで対処する手法と変分ベイズ法による混合正規分布推定と領域分割によって対処する手法を提案した。

考察より、イラスト画像に対する対象物抽出について最適な手法は、時間はかかるものの、安定して高い精度で抽出できる変分ベイズ法による混合正規分布推定と領域分割を用いた手法であると考えられる。しかし、単純な背景を持つイラスト画像と複雑な背景を持つイラスト画像を区別できるとすると、使う手法を切り替えることにより動的輪郭モデルと領域分割を用いた手法を単純な背景の場合に用いれば、より高速で高い精度で抽出できると考えられる。

今後の検討課題として、本論文では固定していたパラメータについて、さらなるパラメータについて検討することや画像に対して最適なパラメータを自動決定させられるようにすることが考えられる。また、対象物の領域を決定する手法の改良や領域分割の手法の改良などにより、さらなる抽出精度の向上が考えられる。

今後、これらの自動抽出手法を利用したイラスト画像の自動分類へと研究を進める。

謝辞

本研究にあたり、多くの助言、指導をしてくださった三好力教授に心からお礼申し上げます。
また、様々なコメントをしてくださった三好研究室の諸氏に厚くお礼申し上げます。

参考文献

1) 総務省,
情報通信白書 平成 21 年度版 (2009) p.120-126.

2) 副島 義貴,
場景変動を考慮した移動物体の追跡に関する研究 (1998) p.5-15.

3) 玉木 徹,
画像中の物体および人物領域の抽出手法に関する研究 (2000).

4) Michael Kass, Andraw Witkin, and Demetri Terzopoulos,
Snakes: Active Contour Models (1988).

5) 荒木 昭一, 横矢 直和, 岩佐 英彦, 竹村 治雄,
複数物体の抽出を目的とした交差判定により分裂する動的輪郭モデル (1996).

6) Donna J. Williams and Mubark Shah,
A Fast Algorithm for Active Contours and Curvature Estimation (1991).

7) 田村 秀行,
コンピュータ画像処理 (オーム社, 2002).

8) 高木幹雄, 下田陽久,
新編 画像解析ハンドブック (東京大学出版会, 2004) p.1651.

9) 大橋 巧, Zaher AGHBARI, 牧之内 顕文,
Hill-Climbing を用いたイメージセグメンテーション (2003).

10) 上田 修功,
EM アルゴリズムの新展開:変分ベイズ法 (2002).

11) 上田 修功,
ベイズ学習 [II] -ベイズ学習の基礎- (2002).

12) 上田 修功,
ベイズ学習 [III] -変分ベイズ学習の基礎- (2002).

13) 上田 修功,
ベイズ学習 [IV・完] -変分ベイズ学習の応用例- (2002).

14) 物理のかぎしっぽ,
変分法 1
http://www12.plala.or.jp/ksp/mathInPhys/variations1/ (2005).

15) C・M・ビショップ著, 元田 浩/栗田 多喜夫/樋口 知之/松本 裕治/村田 昇監訳,
パターン認識と機械学習 上 ベイズ理論による統計的予測
(シュプリンガー・ジャパン, 2007).

16) C・M・ビショップ著, 元田 浩/栗田 多喜夫/樋口 知之/松本 裕治/村田 昇監訳,
パターン認識と機械学習 下 ベイズ理論による統計的予測
(シュプリンガー・ジャパン, 2007) p.175-200.

17) 荒木佑季,
変分ベイズ学習による混合正規分布推定 -検証と改善- (2008).

18) 荒木佑季,
変分ベイズ学習の具体的使用法 (2008) p.1-4.

付 録A　実験データ

表 A.1: 単純な背景のイラスト画像における正答率

	従来手法	提案手法 1	提案手法 2
データ 0	0.910136	0.942983	0.784312
データ 1	0.926592	0.969531	0.843829
データ 2	0.733302	0.731397	0.845957
データ 3	0.906790	0.945234	0.900966
データ 4	0.961775	0.975896	0.917881
データ 5	0.838931	0.856255	0.926719
データ 6	0.702614	0.885206	0.927477
データ 7	0.851537	0.926643	0.931956
データ 8	0.875592	0.954468	0.935380
データ 9	0.906559	0.911811	0.937017
データ 10	0.933133	0.951985	0.958349
データ 11	0.945757	0.968957	0.960921
データ 12	0.957281	0.977680	0.972155

表 A.2: 複雑な背景のイラスト画像における正答率

	従来手法	提案手法 1	提案手法 2
データ 0	0.405982	0.416835	0.416389
データ 1	0.481940	0.468552	0.538081
データ 2	0.328235	0.290786	0.620935
データ 3	0.600108	0.415341	0.684892
データ 4	0.276384	0.273272	0.788040
データ 5	0.449311	0.442492	0.794607
データ 6	0.562603	0.384189	0.810012
データ 7	0.612497	0.597832	0.845892
データ 8	0.848257	0.816647	0.867638

表 A.3: 単純な背景のイラスト画像における画像サイズと処理時間

	画像サイズ	従来手法	提案手法 1	提案手法 2
データ 0	456702	14430.681068	14771.047583	7417.076316
データ 1	767832	6328.068983	8063.586346	7573.61729
データ 2	197984	266.573535	470.526132	4903.957526
データ 3	140800	314.791742	366.991908	1476.88829
データ 4	480000	1460.069772	1763.377128	8080.127295
データ 5	640000	1906.967618	3699.919996	7998.411824
データ 6	224950	516.796649	783.161302	3757.330407
データ 7	630000	1451.748712	2798.350136	11439.809931
データ 8	845500	4984.581542	8177.241011	13650.570095
データ 9	310400	1130.491489	1877.412774	4219.670812
データ 10	472207	987.571584	1190.45066	7893.232118
データ 11	1335000	3171.344491	5184.285752	22254.650758
データ 12	622400	1960.256099	2760.049182	6328.738946

表 A.4: 複雑な背景のイラスト画像における画像サイズと処理時間

	画像サイズ	従来手法	提案手法 1	提案手法 2
データ 0	504000	436.810955	1052.494073	18008.899074
データ 1	480000	410.981802	743.91762	14498.065617
データ 2	3871820	12050.528593	22487.074733	26754.20366
データ 3	510000	1465.28497	1718.649009	9408.153154
データ 4	378000	419.97273	848.442958	9701.67123
データ 5	1228800	675.135934	1291.398946	18256.400698
データ 6	320400	375.995828	640.585803	8584.32126
データ 7	307200	404.589745	870.77733	6773.769705
データ 8	756900	20207.831478	21564.455773	10094.762863

表 A.5: 単純な背景のイラスト画像における更新回数

	従来手法 提案手法 1	提案手法 2
データ 0	1000	1000
データ 1	162	1000
データ 2	29	1000
データ 3	36	270
データ 4	90	1000
データ 5	66	1000
データ 6	44	1000
データ 7	74	1000
データ 8	125	1000
データ 9	66	1000
データ 10	59	1000
データ 11	83	1000
データ 12	92	1000

表 A.6: 複雑な背景のイラスト画像における更新回数

	従来手法 提案手法 1	提案手法 2
データ 0	26	1000
データ 1	24	1000
データ 2	113	1000
データ 3	89	1000
データ 4	29	1000
データ 5	16	1000
データ 6	31	1000
データ 7	36	1000
データ 8	870	1000

付 録B　ソースコード

ソースコード B.1: amlab.cpp

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <stdexcept>
5  #include <windows.h>
6  #include <string>
7  #include "nclr_cv_image.hpp"
8  #include "area_detect.hpp"
9  #include "image_divide.hpp"
10 #include "timecounter.hpp"
11
12 using namespace Nclr::OpenCV;
13
14 int main(int argc, char **argv)
15 {
16     HMODULE dll_ad = NULL;
17     HMODULE dll_imgdiv = NULL;
18
19     ad_ptr ad = NULL;
20     imgdiv_ptr imgdiv = NULL;
21
22     time_counter tc;
23
24     try {
25         if( argc != 4 )
26             throw std::runtime_error( "invalid_arguments\n" );
27
28         dll_ad = LoadLibrary( argv[2] );
29         if( dll_ad == NULL )
30             throw std::runtime_error( "cannot_load_area_detect_dll\n" );
31         dll_imgdiv = LoadLibrary( argv[3] );
32         if( dll_imgdiv == NULL )
33             throw std::runtime_error( "cannot_load_image_divide_dll\n" );
34
35         ad = (ad_ptr)GetProcAddress( dll_ad, (LPCSTR)1 );
36         if( ad == NULL )
37             throw std::runtime_error( "cannot_get_area_detect_address\n" );
38         imgdiv = (imgdiv_ptr)GetProcAddress( dll_imgdiv, (LPCSTR)1 );
39         if( imgdiv == NULL )
40             throw std::runtime_error( "cannot_get_image_divide_address\n" );
41
42         Image<unsigned char, 3> srcimg( argv[1] );
43         if( !srcimg.isvalid() )
44             throw std::runtime_error( "cannot_open_image_file\n" );
45
46         std::string filename = argv[1];
47         filename.erase( filename.begin() + filename.rfind( '.' ), filename.end() );
48
49         std::string dllname[2];
50         for( int i = 0; i < 2; ++i ) {
51             dllname[i] = argv[2 + i];
52             dllname[i].erase( dllname[i].begin() + dllname[i].find( '.' ),
53                               dllname[i].end() );
54         }
55         filename += "_" + dllname[0] + dllname[1];
56         filename += ".txt";
57
58         FILE *txtfp = fopen( filename.c_str(), "w" );
59         int t = 0;
60
```

```
61          Image<unsigned char, 1> area( srcimg.getwidth(), srcimg.getheight() );
62          cvZero( area );
63
64          tc.begin();
65          (*ad)( &srcimg, &area, &t );
66          fprintf( txtfp, "area_detect_time_=_%f[ms]\n", tc.end() );
67          fprintf( txtfp, "area_detect_cnt_=_%d\n", t );
68
69          Image<unsigned char, 3> areaimg( srcimg.getwidth(), srcimg.getheight() );
70          for( int y = 0; y < srcimg.getheight(); ++y ) {
71              for( int x = 0; x < srcimg.getwidth(); ++x ) {
72                  for( int c = 0; c < 3; ++c )
73                      areaimg.data( x, y, c ) = ~( srcimg.data( x, y, c ) ^ area.data( x, y ) );
74              }
75          }
76
77          Image<unsigned short, 1> data_impl( srcimg.getwidth(), srcimg.getheight() );
78          imgdiv_info id_info;
79          id_info.data = &data_impl;
80
81          tc.begin();
82          (*imgdiv)( &srcimg, &id_info );
83          fprintf( txtfp, "image_divide_time_=_%f[ms]\n", tc.end() );
84          fclose( txtfp );
85
86          Image<unsigned char, 3> result( srcimg.getwidth(), srcimg.getheight() );
87          cvZero( result );
88
89          if( id_info.area_cnt[0] != srcimg.getwidth() * srcimg.getheight() )
90          {
91              std::vector<int> target_cnt( id_info.area_cnt.size(), 0 );
92              for( int y = 0; y < srcimg.getheight(); ++y ) {
93                  for( int x = 0; x < srcimg.getwidth(); ++x ) {
94                      if( area.data( x, y ) == 255 )
95                          ++target_cnt[id_info.data->data( x, y )];
96                  }
97              }
98
99              for( int y = 0; y < srcimg.getheight(); ++y ) {
100                 for( int x = 0; x < srcimg.getwidth(); ++x ) {
101                     double ratio = (double)target_cnt[id_info.data->data( x, y )]
102                         / id_info.area_cnt[id_info.data->data( x, y )];
103                     if( ratio >= 0.5 ) {
104                         for( int c = 0; c < 3; ++c )
105                             result.data( x, y, c ) = srcimg.data( x, y, c );
106                     }
107                 }
108             }
109         }
110         else {
111             for( int y = 0; y < srcimg.getheight(); ++y ) {
112                 for( int x = 0; x < srcimg.getwidth(); ++x ) {
113                     if( area.data( x, y ) == 255 ) {
114                         for( int c = 0; c < 3; ++c )
115                             result.data( x, y, c ) = srcimg.data( x, y, c );
116                     }
117                 }
118             }
119         }
120
121         /*
122         cvNamedWindow( "area" );
123         cvNamedWindow( "result" );
124         cvShowImage( "area", areaimg );
125         cvShowImage( "result", result );
126
127         cvWaitKey( 0 );
128
129         cvDestroyWindow( "result" );
130         cvDestroyWindow( "area" );
131         */
132
133         filename.erase( filename.begin() + filename.rfind( '.' ), filename.end() );
```

```
134             filename += "_result.jpg";
135             cvSaveImage( filename.c_str(), result );
136             filename.erase( filename.begin() + filename.rfind( '_' ), filename.end() );
137             filename += "_area.jpg";
138             cvSaveImage( filename.c_str(), areaimg );
139         }
140         catch(std::runtime_error &e) {
141             fprintf( stderr, "%s", e.what() );
142         }
143
144         if( dll_imgdiv != NULL )
145             FreeLibrary( dll_imgdiv );
146         if( dll_ad != NULL )
147             FreeLibrary( dll_ad );
148
149         printf( "amlab_exit\n" );
150
151         return 0;
152     }
```

ソースコード B.2: nclr_cv_misc.hpp

```
1   #ifndef NCLR_CV_MISC_HPP_
2   #define NCLR_CV_MISC_HPP_
3
4   #include <cv.h>
5
6   namespace Nclr
7   {
8       namespace OpenCV
9       {
10          template <typename T>
11          inline T &ImgData(IplImage *img, int x, int y, int offset = 0)
12          {
13              return ((T *)( img->imageData + img->widthStep * y ))[x * img->nChannels +
                    offset];
14          }
15
16          template <typename T>
17          inline const T &ImgData(const IplImage *img, int x, int y, int offset = 0)
18          {
19              return ((T *)( img->imageData + img->widthStep * y ))[x * img->nChannels +
                    offset];
20          }
21
22          inline bool EqualImgSize(const IplImage *lhs, const IplImage *rhs)
23          {
24              return lhs->width == rhs->width && lhs->height == rhs->height;
25          }
26
27      }; // namespace OpenCV
28  }; // namespace Nclr
29
30  #endif // NCLR_CV_MISC_HPP_
```

ソースコード B.3: nclr_cv_mat.hpp

```
1   #ifndef NCLR_CV_MAT_HPP_
2   #define NCLR_CV_MAT_HPP_
3
4   #include <cv.h>
5
6   namespace Nclr
7   {
8       namespace OpenCV
9       {
10          namespace
11          {
12              template <typename T> struct MatrixType;
13
14              template <> struct MatrixType<float> { enum { val = CV_32F }; };
15              template <> struct MatrixType<double> { enum { val = CV_64F }; };
```

```
16          };
17
18          template <typename Type>
19          class Matrix
20          {
21          public:
22              Matrix(int row, int col);
23              ~Matrix();
24              Matrix(const Matrix<Type> &rhs);
25              Matrix<Type> &operator=(const Matrix<Type> &rhs);
26
27              Type get(int row, int col) const;
28              void set(int row, int col, Type val);
29
30              int getrows() const { return mat_->rows; }
31              int getcols() const { return mat_->cols; }
32
33              CvMat *gethandle() const { return mat_; }
34              operator CvMat *() const { return mat_; }
35
36          private:
37              CvMat *mat_;
38          };
39
40          //
                ============================================================================
41
42          template <typename Type>
43          Matrix<Type>::Matrix(int row, int col) :
44              mat_( NULL )
45          {
46              mat_ = cvCreateMat( row, col, MatrixType<Type>::val );
47              cvZero( mat_ );
48          }
49
50          template <typename Type>
51          Matrix<Type>::~Matrix()
52          {
53              cvReleaseMat( &mat_ );
54          }
55
56          template <typename Type>
57          Matrix<Type>::Matrix(const Matrix<Type> &rhs)
58          {
59              mat_ = cvCreateMat( rhs.getrows(), rhs.getcols(), MatrixType<Type>::val );
60              for( int y = 0; y < rhs.getcols(); ++y ) {
61                  for( int x = 0; x < rhs.getrows(); ++x )
62                      set( x, y, rhs.get( x, y ) );
63              }
64          }
65
66          template <typename Type>
67          Matrix<Type> &Matrix<Type>::operator=(const Matrix<Type> &rhs)
68          {
69              //cvReleaseMat( &mat_ );
70
71              //mat_ = cvCreateMat( rhs.getrows(), rhs.getcols(), MatrixType<Type>::val );
72              for( int y = 0; y < rhs.getcols(); ++y ) {
73                  for( int x = 0; x < rhs.getrows(); ++x )
74                      set( x, y, rhs.get( x, y ) );
75              }
76
77              return *this;
78          }
79
80          template <typename Type>
81          Type Matrix<Type>::get(int row, int col) const
82          {
83              return cvmGet( mat_, row, col );
84          }
85
86          template <typename Type>
87          void Matrix<Type>::set(int row, int col, Type val)
```

```
88          {
89              cvmSet( mat_, row, col, val );
90          }
91      };
92  };
93
94  #endif // NCLR_CV_MAT_HPP_
```

ソースコード B.4: nclr_cv_image.hpp

```
1   #ifndef NCLR_CV_IMAGE_HPP_
2   #define NCLR_CV_IMAGE_HPP_
3
4   #include <cv.h>
5   #include <highgui.h>
6   #include "nclr_cv_misc.hpp"
7
8   namespace Nclr
9   {
10      namespace
11      {
12          template <typename DepthType> struct Depth;
13
14          template <> struct Depth<unsigned char> { enum { val = IPL_DEPTH_8U }; };
15          template <> struct Depth<unsigned short> { enum { val = IPL_DEPTH_16U }; };
16          template <> struct Depth<char> { enum { val = IPL_DEPTH_8S }; };
17          template <> struct Depth<short> { enum { val = IPL_DEPTH_16S }; };
18          template <> struct Depth<long> { enum { val = IPL_DEPTH_32S }; };
19          template <> struct Depth<float> { enum { val = IPL_DEPTH_32F }; };
20          template <> struct Depth<double> { enum { val = IPL_DEPTH_64F }; };
21
22      }; // namespace
23
24      namespace OpenCV
25      {
26          //
                ////////////////////////////////////////////////////////////////////
27          //
28          template <typename DepthType, int Channels>
29          class Image
30          {
31          public:
32              Image(int width, int height);
33              Image(const char *filename);
34              Image(const Image<DepthType, Channels> &rhs);
35              Image<DepthType, Channels> &operator=(const Image<DepthType, Channels> &rhs);
36              ~Image();
37
38              bool isvalid() const { return img_ != NULL; }
39
40              int getwidth() const { return img_->width; }
41              int getheight() const { return img_->height; }
42              int getchannels() const { return Channels; }
43              int getdepth() const { return Depth<DepthType>::val; }
44
45              DepthType &data(int x, int y, int offset = 0);
46              const DepthType &data(int x, int y, int offset = 0) const;
47
48              IplImage *gethandle() { return img_; }
49              const IplImage *gethandle() const { return img_; }
50              operator IplImage *() { return img_; }
51              operator const IplImage *() const { return img_; }
52
53          private:
54              void load(const char *filename, Image<unsigned char, 3> &dest);
55              void load(const char *filename, Image<unsigned char, 1> &dest);
56
57          private:
58              IplImage *img_;
59          };
60
61          //
```

```
62          //
63          template <typename DepthType, int Channels>
64          Image<DepthType, Channels>::Image(int width, int height) :
65              img_( NULL )
66          {
67              img_ = cvCreateImage( cvSize( width, height ),
68                                    Depth<DepthType>::val,
69                                    Channels );
70          }
71
72          template <typename DepthType, int Channels>
73          Image<DepthType, Channels>::Image(const char *filename) :
74              img_( NULL )
75          {
76              load( filename, *this );
77          }
78
79          template <typename DepthType, int Channels>
80          Image<DepthType, Channels>::Image(const Image<DepthType, Channels> &rhs) :
81              img_( NULL )
82          {
83              img_ = cvCreateImage( cvGetSize( rhs.img_ ),
84                                    Depth<DepthType>::val,
85                                    Channels );
86
87              if( rhs.img_ != NULL && img_ != NULL )
88                  cvCopy( rhs.img_, img_ );
89          }
90
91          template <typename DepthType, int Channels>
92          Image<DepthType, Channels> &Image<DepthType, Channels>::operator=(const Image<
                DepthType, Channels> &rhs)
93          {
94              cvReleaseImage( &img_ );
95
96              img_ = cvCreateImage( cvGetSize( rhs.img_ ),
97                                    Depth<DepthType>::val,
98                                    Channels );
99
100             if( rhs.img_ != NULL && img_ != NULL )
101                 cvCopy( rhs.img_, img_ );
102         }
103
104         template <typename DepthType, int Channels>
105         Image<DepthType, Channels>::~Image()
106         {
107             cvReleaseImage( &img_ );
108         }
109
110         template <typename DepthType, int Channels>
111         DepthType &Image<DepthType, Channels>::data(int x, int y, int offset)
112         {
113             return ImgData<DepthType>( img_, x, y, offset );
114         }
115
116         template <typename DepthType, int Channels>
117         const DepthType &Image<DepthType, Channels>::data(int x, int y, int offset) const
118         {
119             return ImgData<DepthType>( img_, x, y, offset );
120         }
121
122         template <typename DepthType, int Channels>
123         void Image<DepthType, Channels>::load(const char *filename, Image<unsigned char, 3> &
                dest)
124         {
125             dest.img_ = cvLoadImage( filename, CV_LOAD_IMAGE_COLOR );
126         }
127
128         template <typename DepthType, int Channels>
129         void Image<DepthType, Channels>::load(const char *filename, Image<unsigned char, 1> &
                dest)
130         {
```

```
131                 dest.img_ = cvLoadImage( filename, CV_LOAD_IMAGE_GRAYSCALE );
132             }
133
134     }; // namespace OpenCV
135 }; // namespace Nclr
136
137 #endif // NCLR_CV_IMAGE_HPP_
```

ソースコード B.5: timecounter.hpp

```
1  #ifndef TIMECOUNTER_HPP_
2  #define TIMECOUNTER_HPP_
3
4  #include <windows.h>
5
6  class time_counter
7  {
8  public:
9      time_counter()
10     {
11         QueryPerformanceFrequency( &freq_ );
12     }
13
14     void begin()
15     {
16         QueryPerformanceCounter( &before_ );
17     }
18
19     double end()
20     {
21         LARGE_INTEGER after;
22         QueryPerformanceCounter( &after );
23
24         return ( after.QuadPart − before_.QuadPart ) ∗ 1000.0 / freq_.QuadPart;
25     }
26
27 private:
28     LARGE_INTEGER freq_, before_;
29 };
30
31 #endif // TIMECOUNTER_HPP_
```

ソースコード B.6: hcseg.hpp

```
1  #ifndef HCSEG_HPP_
2  #define HCSEG_HPP_
3
4  #include <vector>
5  #include "nclr_cv_image.hpp"
6
7  typedef Nclr::OpenCV::Image<unsigned short, 1> Segment;
8  typedef std::vector<CvScalar> SegPeaks;
9  typedef std::vector<int> SegCount;
10
11 int HCSegment(const Nclr::OpenCV::Image<unsigned char, 3> &src,
12                 Segment &segment,
13                 SegPeaks &peaks,
14                 SegCount &segcnt,
15                 int qh, int qs, int qv);
16
17 #endif // HCSEG_HPP_
```

ソースコード B.7: hcseg.cpp

```
1  #include "hcseg.hpp"
2
3  using namespace Nclr::OpenCV;
4
5  namespace
6  {
7      int HillClimb(int p, int prev, int next, const std::vector<unsigned int> &hist, bool flg)
8      {
```

```
 9          --prev;
10          ++next;
11
12          if( flg ) {
13              if( prev < 0 )
14                      prev = hist.size() - 1;
15              else if( next >= hist.size() )
16                  next = 0;
17          }
18          else {
19              if( prev < 0 ) {
20                  if( hist[p] > hist[next] )
21                      return p;
22                  else
23                      return next;
24              }
25              else if( next >= hist.size() ) {
26                  if( hist[p] > hist[prev] )
27                      return p;
28                  else
29                      return prev;
30              }
31          }
32
33          if( hist[p] > hist[prev] && hist[p] > hist[next] )
34              return p;
35
36          if( hist[prev] > hist[next] )
37              return ++prev;
38          else if( hist[prev] < hist[next] )
39              return --next;
40
41          return HillClimb( p, prev, next, hist, flg ) < p ? prev : next;
42      }
43
44      int HillClimb(int p, const std::vector<unsigned int> &hist, bool flg)
45      {
46          int prev = p - 1, next = p + 1;
47
48          if( flg ) {
49              if( prev < 0 )
50                      prev = hist.size() - 1;
51              else if( next >= hist.size() )
52                  next = 0;
53          }
54          else {
55              if( prev < 0 ) {
56                  if( hist[p] > hist[next] )
57                      return p;
58                  else
59                      return next;
60              }
61              else if( next >= hist.size() ) {
62                  if( hist[p] > hist[prev] )
63                      return p;
64                  else
65                      return prev;
66                  }
67          }
68
69          if( hist[p] > hist[prev] && hist[p] > hist[next] )
70              return p;
71
72          if( hist[prev] > hist[next] )
73              return prev;
74          else if( hist[prev] < hist[next] )
75              return next;
76
77          return HillClimb( p, prev, next, hist, flg );
78      }
79
80      void HillClimb(int h, int s, int v, const std::vector<unsigned int> hist[3],
81                              std::vector<CvScalar> &moved)
82      {
```

```
83          moved.clear();
84
85          int th = h, ts = s, tv = v;
86          int pth = h, pts = s, ptv = v;
87
88          moved.push_back( cvScalar( th, ts, tv, 0 ) );
89          for(;;) {
90              th = HillClimb( th, hist[0], true );
91              ts = HillClimb( ts, hist[1], false );
92              if( ts > 1 )
93                  tv = HillClimb( tv, hist[2], false );
94
95              if( pth == th && pts == ts && ptv == tv ) {
96                  break;
97              }
98
99              moved.push_back( cvScalar( th, ts, tv, 0 ) );
100             pth = th;
101             pts = ts;
102             ptv = tv;
103         }
104     }
105
106     inline bool operator==(const CvScalar &lhs, const CvScalar &rhs)
107     {
108         return lhs.val[0] == rhs.val[0] && lhs.val[1] == rhs.val[1] && lhs.val[2] == rhs.val[2];
109     }
110
111 }; // namespace
112
113 int HCSegment(const Image<unsigned char, 3> &src, Segment &segment, SegPeaks &peaks,
        SegCount &segcnt,
114                 int qh, int qs, int qv)
115 {
116     using namespace Nclr::OpenCV;
117
118     Image<float, 3> src_n( src.getwidth(), src.getheight() );
119     Image<unsigned short, 3> src_s( src.getwidth(), src.getheight() );
120     cvConvert( src, src_n );
121     for( int y = 0; y < src.getheight(); ++y ) {
122         for( int x = 0; x < src.getwidth(); ++x ) {
123             for( int i = 0; i < 3; ++i )
124                 src_n.data( x, y, i ) /= 255.0f;
125         }
126     }
127     cvCvtColor( src_n, src_n, CV_BGR2HSV );
128     for( int y = 0; y < src.getheight(); ++y ) {
129         for( int x = 0; x < src.getwidth(); ++x )
130             src_n.data( x, y, 0 ) /= 360.0f;
131     }
132
133     int quant[3] = { qh, qs, qv };
134     float quant_f[3];
135     std::vector<unsigned int> hist[3];
136     for( int i = 0; i < 3; ++i ) {
137         quant_f[i] = 1.0f / quant[i];
138         hist[i].resize( quant[i], 0 );
139     }
140
141     for( int y = 0; y < src.getheight(); ++y ) {
142         for( int x = 0; x < src.getwidth(); ++x ) {
143             if( segment.data( x, y ) == -1 )
144                 continue;
145
146             for( int i = 0; i < 3; ++i ) {
147                 float val = src_n.data( x, y, i );
148                 float q = 0;
149                 for( int j = 0; j < quant[i]; ++j ) {
150                     if( q <= val && val <= q + quant_f[i] ) {
151                         src_s.data( x, y, i ) = j;
152                         if( i != 0 || ( i == 0 && src_n.data( x, y, 0 ) != 0 ) )
153                             ++hist[i][j];
154                         break;
```

```
155                         }
156                         q += quant_f[i];
157                     }
158                 }
159             }
160         }
161
162         for( int i = 0; i < 3; ++i ) {
163             int test = 0;
164             for( int j = 0; j < hist[i].size(); ++j ) {
165                 if( hist[i][j] == 0 )
166                     ++test;
167             }
168             if( test == hist[i].size() )
169                 return 1;
170         }
171
172         peaks.clear();
173         std::vector<unsigned short> histpeak( quant[0] * quant[1] * quant[2], -1 );
174         std::vector<CvScalar> moved;
175         for( int v = 0; v < quant[2]; ++v ) {
176             for( int s = 0; s < quant[1]; ++s ) {
177                 for( int h = 0; h < quant[0]; ++h ) {
178                     std::size_t i;
179                     HillClimb( h, s, v, hist, moved );
180                     for( i = 0; i < peaks.size(); ++i ) {
181                         if( peaks[i] == moved.back() )
182                             break;
183                     }
184                     if( i >= peaks.size() ) {
185                         peaks.push_back( moved.back() );
186                         i = peaks.size() - 1;
187                     }
188                     histpeak[h + s * quant[0] + v * quant[0] * quant[1]] = i;
189                 }
190             }
191         }
192
193         for( std::size_t i = 0; i < peaks.size(); ++i ) {
194             for( int j = 0; j < 3; ++j )
195                 peaks[i].val[j] /= quant[j];
196         }
197
198         segcnt.clear();
199         segcnt.resize( peaks.size(), 0 );
200         for( int y = 0; y < src.getheight(); ++y ) {
201             for( int x = 0; x < src.getwidth(); ++x ) {
202                 if( segment.data( x, y ) == -1 )
203                     continue;
204                 int pos = src_s.data( x, y, 0 ) + src_s.data( x, y, 1 ) * quant[0]
205                     + src_s.data( x, y, 2 ) * quant[0] * quant[1];
206                 segment.data( x, y, 0 ) = histpeak[pos];
207                 ++segcnt[histpeak[pos]];
208             }
209         }
210
211         return 0;
212     }
```

ソースコード B.8: variate_bayes.hpp

```
1  #ifndef VARIATE_BAYES_HPP_
2  #define VARIATE_BAYES_HPP_
3
4  #define _USE_MATH_DEFINES
5  #include <math.h>
6  #include <float.h>
7  #include "nclr_cv_mat.hpp"
8  #include <boost/math/special_functions/digamma.hpp>
9
10 using namespace Nclr::OpenCV;
11 using namespace boost::math;
12
```

```
13 struct AVBParams
14 {
15     double phi, xi, eta;
16     Matrix<double> nu, B;
17
18     AVBParams(int d) :
19         nu( d, 1 ), B( d, d )
20     { }
21 };
22
23 struct VBParams
24 {
25     double phi, eta, f, Ni;
26     Matrix<double> mu, B, sigma;
27
28     VBParams(int d) :
29         mu( d, 1 ), B( d, d ), sigma( d, d )
30     { }
31 };
32
33 class VariateBayes
34 {
35 public:
36     VariateBayes(int d, int cnum);
37     ~VariateBayes();
38
39     int Learn(const std::vector<Matrix<double> > &data,
40               double param = 20.0,
41               int tcnt = 1000,
42               int errmax = 100);
43     int Learn(const VariateBayes &vb,
44               const std::vector<Matrix<double> > &data,
45               double params = 20.0,
46               int tcnt = 1000,
47               int errmax = 100);
48     double Predict(const Matrix<double> &x);
49     double Gaussian(const Matrix<double> &x);
50
51     int KMeans(const std::vector<Matrix<double> > &data,
52                std::vector<Matrix<double> > &centroid,
53                int tcnt = 1000);
54
55 private:
56     int Init(const std::vector<Matrix<double> > &data,
57              double param);
58     int EStep(const std::vector<Matrix<double> > &data,
59               std::vector<std::vector<double> > &z);
60     int MStep(const std::vector<Matrix<double> > &data,
61               const std::vector<std::vector<double> > &z);
62
63 private:
64     const int D;
65     std::vector<VBParams> clusters_;
66     AVBParams p0_;
67 };
68
69 //
       ==========================================================================

70
71 VariateBayes::VariateBayes(int d, int cnum) :
72     D( d ),
73     clusters_( cnum, VBParams( d ) ),
74     p0_( d )
75 { }
76
77 VariateBayes::~VariateBayes()
78 { }
79
80 int VariateBayes::Init(const std::vector<Matrix<double> > &data,
81                        double param)
82 {
83     std::vector<Matrix<double> > center( clusters_.size(), Matrix<double>( D, 1 ) );
84     KMeans( data, center );
```

```
85
86      p0_.phi = data.size() / (double)clusters_.size();
87      p0_.xi = 1.0;
88      p0_.eta = D + 2.0;
89
90      for( std::size_t n = 0; n < data.size(); ++n ) {
91          for( int k = 0; k < D; ++k )
92              p0_.nu.set( k, 0, p0_.nu.get( k, 0 ) + data[n].get( k, 0 ) );
93      }
94      for( int k = 0; k < D; ++k )
95          p0_.nu.set( k, 0, p0_.nu.get( k, 0 ) / data.size() );
96
97      for( std::size_t n = 0; n < data.size(); ++n ) {
98          for( int k = 0; k < D; ++k ) {
99              double val = data[n].get( k, 0 ) - p0_.nu.get( k, 0 );
100             p0_.B.set( k, k, p0_.B.get( k, k ) + param * val * val );
101         }
102     }
103     for( int k = 0; k < D; ++k )
104         p0_.B.set( k, k, p0_.B.get( k, k ) / data.size() );
105
106     for( std::size_t i = 0; i < clusters_.size(); ++i ) {
107         clusters_[i].phi = p0_.phi;
108         clusters_[i].Ni = p0_.phi;
109         clusters_[i].eta = p0_.eta;
110         clusters_[i].B = p0_.B;
111         clusters_[i].f = p0_.eta + clusters_[i].Ni + 1.0 - D;
112         for( int k = 0; k < D; ++k ) {
113             clusters_[i].mu.set( k, 0, center[i].get( k, 0 ) );
114             clusters_[i].sigma.set( k, k, clusters_[i].B.get( k, k )
115                                     / ( clusters_[i].f * ( clusters_[i].Ni + p0_.xi ) ) );
116         }
117     }
118
119     return 0;
120 }
121
122 int VariateBayes::KMeans(const std::vector<Matrix<double> > &data,
123                          std::vector<Matrix<double> > &centroid,
124                          int tcnt)
125 {
126     std::vector<int> group( data.size(), -1 );
127     std::vector<int> prevgroup( data.size(), -1 );
128     std::vector<int> groupcnt( centroid.size(), 0 );
129
130     for( std::size_t n = 0; n < data.size(); ++n )
131         group[n] = rand() % centroid.size();
132
133     int t;
134     for( t = 0; t < tcnt; ++t ) {
135         for( std::size_t i = 0; i < centroid.size(); ++i ) {
136             cvZero( centroid[i] );
137             groupcnt[i] = 0;
138         }
139         for( std::size_t n = 0; n < data.size(); ++n ) {
140             for( int k = 0; k < D; ++k )
141                 centroid[group[n]].set( k, 0, centroid[group[n]].get( k, 0 ) + data[n].get( k, 0 ) );
142             ++groupcnt[group[n]];
143         }
144         for( std::size_t i = 0; i < centroid.size(); ++i ) {
145             for( int k = 0; k < D; ++k )
146                 centroid[i].set( k, 0, centroid[i].get( k, 0 ) / groupcnt[i] );
147         }
148
149         for( std::size_t n = 0; n < data.size(); ++n ) {
150             double dmin = DBL_MAX;
151             for( std::size_t i = 0; i < centroid.size(); ++i ) {
152                 double d = 0;
153                 for( int k = 0; k < D; ++k ) {
154                     double val = data[n].get( k, 0 ) - centroid[i].get( k, 0 );
155                     d += val * val;
156                 }
157                 d = sqrt( d );
```

```
158                     if( d < dmin ) {
159                         dmin = d;
160                         group[n] = i;
161                     }
162                 }
163             }
164
165             std::size_t cntequal;
166             for( cntequal = 0; cntequal < data.size(); ++cntequal ) {
167                 if( group[cntequal] != prevgroup[cntequal] )
168                     break;
169             }
170             if( cntequal == data.size() )
171                 break;
172
173             prevgroup = group;
174         }
175
176         return t;
177     }
178
179     int VariateBayes::EStep(const std::vector<Matrix<double> > &data,
180                             std::vector<std::vector<double> > &z)
181     {
182         const double M = clusters_.size();
183
184         double nsum = 0;
185         for( std::size_t j = 0; j < clusters_.size(); ++j )
186             nsum += clusters_[j].Ni;
187
188         Matrix<double> invBi( D, D );
189         Matrix<double> xu( D, D );
190         Matrix<double> tmp( D, D );
191         for( std::size_t i = 0; i < clusters_.size(); ++i ) {
192             double a0 = digamma( p0_.phi + clusters_[i].Ni ) - digamma( M * p0_.phi + nsum );
193             double a1 = 0;
194             for( std::size_t j = 0; j < (std::size_t)D; ++j )
195                 a1 += digamma( ( p0_.eta + clusters_[i].Ni + 1.0 - j ) / 2.0 );
196             double a2 = log( cvDet( clusters_[i].B ) );
197             double a3;
198             for( std::size_t n = 0; n < data.size(); ++n ) {
199                 cvInv( clusters_[i].B, invBi, CV_SVD );
200                 cvMulTransposed( data[n], xu, 0, clusters_[i].mu );
201                 cvScale( clusters_[i].sigma, tmp, clusters_[i].f / ( clusters_[i].f - 2.0 ) );
202                 cvAdd( tmp, xu, tmp );
203                 cvMatMul( invBi, tmp, tmp );
204                 cvScale( tmp, tmp, p0_.eta + clusters_[i].Ni );
205                 CvScalar tr = cvTrace( tmp );
206                 a3 = tr.val[0];
207
208                 z[i][n] = exp( a0 + a1 / 2.0 - a2 / 2.0 - a3 / 2.0 );
209             }
210         }
211
212         for( std::size_t n = 0; n < data.size(); ++n ) {
213             double rsum = 0;
214             for( std::size_t j = 0; j < clusters_.size(); ++j )
215                 rsum += z[j][n];
216             for( std::size_t i = 0; i < clusters_.size(); ++i ) {
217                 z[i][n] /= rsum;
218                 if( !_finite( z[i][n] ) )
219                     return 1;
220             }
221         }
222
223         return 0;
224     }
225
226     int VariateBayes::MStep(const std::vector<Matrix<double> > &data,
227                             const std::vector<std::vector<double> > &z)
228     {
229         for( std::size_t i = 0; i < clusters_.size(); ++i ) {
230             Matrix<double> xi( D, 1 );
```

```
231             Matrix<double> tmpc( D, 1 );
232             for( std::size_t n = 0; n < data.size(); ++n ) {
233                 cvScale( data[n], tmpc, z[i][n] );
234                 cvAdd( tmpc, xi, xi );
235             }
236             cvScale( xi, xi, 1.0 / clusters_[i].Ni );
237
238             clusters_[i].Ni = 0;
239             for( std::size_t n = 0; n < data.size(); ++n )
240                 clusters_[i].Ni += z[i][n];
241
242             Matrix<double> C( D, D );
243             Matrix<double> tmp( D, D );
244             for( std::size_t n = 0; n < data.size(); ++n ) {
245                 cvMulTransposed( data[n], tmp, 0, xi );
246                 for( int l = 0; l < D; ++l ) {
247                     for( int k = 0; k < D; ++k )
248                         C.set( k, l, C.get( k, l ) + tmp.get( k, l ) * z[i][n] );
249                 }
250             }
251
252             clusters_[i].phi = p0_.phi + clusters_[i].Ni;
253             clusters_[i].eta = p0_.eta + clusters_[i].Ni;
254             for( int k = 0; k < D; ++k ) {
255                 clusters_[i].mu.set( k, 0,
256                                      ( clusters_[i].Ni * xi.get( k, 0 ) + p0_.xi * p0_.nu.get( k, 0 ) )
257                                      / ( clusters_[i].Ni + p0_.xi ) );
258             }
259             clusters_[i].f = clusters_[i].eta + 1.0 - D;
260             cvMulTransposed( xi, tmp, 0, p0_.nu,
261                              ( clusters_[i].Ni * p0_.xi ) / ( clusters_[i].Ni + p0_.xi ) );
262             cvAdd( p0_.B, C, C );
263             cvAdd( C, tmp, clusters_[i].B );
264             cvScale( clusters_[i].B, clusters_[i].sigma,
265                      1.0 / ( ( clusters_[i].Ni + p0_.xi ) * clusters_[i].f ) );
266         }
267
268         return 0;
269     }
270
271     int VariateBayes::Learn(const std::vector<Matrix<double> > &data,
272                             double param,
273                             int tcnt,
274                             int errmax)
275     {
276         std::vector<std::vector<double> > z;
277         for( std::size_t i = 0; i < clusters_.size(); ++i )
278             z.push_back( std::vector<double>( data.size(), 0 ) );
279
280         Init( data, param );
281
282         std::vector<VBParams> prevc = clusters_;
283
284         int t, err = 0;
285         for( t = 0; t < tcnt; ++t ) {
286             if( EStep( data, z ) != 0 ) {
287                 Init( data, param );
288                 t = 0;
289                 if( ++err == errmax )
290                     return -1;
291             }
292             else {
293                 MStep( data, z );
294
295                 std::size_t i;
296                 for( i = 0; i < clusters_.size(); ++i ) {
297                     double val = fabs( clusters_[i].Ni - prevc[i].Ni );
298                     if( val > 1.0e-5 )
299                         break;
300                 }
301                 if( i == clusters_.size() )
302                     break;
303
```

```
304                 prevc = clusters_;
305             }
306         }
307
308         return t;
309     }
310
311     int VariateBayes::Learn(const VariateBayes &vb,
312                             const std::vector<Matrix<double> > &data,
313                             double param,
314                             int tcnt,
315                             int errmax)
316     {
317         std::vector<std::vector<double> > z;
318         for( std::size_t i = 0; i < clusters_.size(); ++i )
319             z.push_back( std::vector<double>( data.size(), 0 ) );
320
321         Init( data, param );
322         clusters_ = vb.clusters_;
323
324         std::vector<VBParams> prevc = clusters_;
325
326         int t, err = 0;
327         for( t = 0; t < tcnt; ++t ) {
328             if( EStep( data, z ) != 0 ) {
329                 Init( data, param );
330                 t = 0;
331                 if( ++err == errmax )
332                     return -1;
333             }
334             else {
335                 MStep( data, z );
336
337                 std::size_t i;
338                 for( i = 0; i < clusters_.size(); ++i ) {
339                     double val = fabs( clusters_[i].Ni - prevc[i].Ni );
340                     if( val > 1.0e-5 )
341                         break;
342                 }
343                 if( i == clusters_.size() )
344                     break;
345
346                 prevc = clusters_;
347             }
348         }
349
350         return t;
351     }
352
353     double VariateBayes::Predict(const Matrix<double> &x)
354     {
355         double p = 0;
356
357         double pnsum = 0;
358         for( std::size_t j = 0; j < clusters_.size(); ++j )
359             pnsum += p0_.phi + clusters_[j].Ni;
360
361         for( std::size_t i = 0; i < clusters_.size(); ++i ) {
362             double a = ( p0_.phi + clusters_[i].Ni ) / pnsum;
363             Matrix<double> mat( D, D );
364             cvScale( clusters_[i].B, mat,
365                      ( clusters_[i].Ni + p0_.xi + 1.0 ) / ( clusters_[i].Ni + p0_.xi ) );
366             cvInv( mat, mat );
367             double d = cvMahalanobis( x, clusters_[i].mu, mat );
368
369             p += a * pow( 1.0 + d * d, -( p0_.eta + clusters_[i].Ni + 1.0 ) / 2.0 );
370         }
371
372         return p;
373     }
374
375     double VariateBayes::Gaussian(const Matrix<double> &x)
376     {
377         double p = 0;
```

```
378     const double pid = pow( 2.0 * M_PI, -( D / 2.0 ) );
379
380     double pnsum = 0;
381     for( std::size_t j = 0; j < clusters_.size(); ++j )
382         pnsum += p0_.phi + clusters_[j].Ni;
383
384     for( std::size_t i = 0; i < clusters_.size(); ++i ) {
385         double a = ( p0_.phi + clusters_[i].Ni ) / pnsum;
386
387         Matrix<double> mat( D, D );
388         cvInv( clusters_[i].B, mat );
389         cvScale( mat, mat, p0_.eta + clusters_[i].Ni );
390
391         double d = cvMahalanobis( x, clusters_[i].mu, mat );
392
393         p += a * pid * sqrt( cvDet( mat ) ) * exp( d * d / (-2.0) );
394     }
395
396     return p;
397 }
398
399 #endif // VARIATE_BAYES_HPP_
```

ソースコード B.9: area_detect.hpp

```
1  #ifndef AREA_DETECT_HPP_
2  #define AREA_DETECT_HPP_
3
4  #include "nclr_cv_image.hpp"
5
6  int __declspec(dllexport) __stdcall area_detect(
7      const Nclr::OpenCV::Image<unsigned char, 3> *srcimg,
8      Nclr::OpenCV::Image<unsigned char, 1> *destimg,
9      int *tcnt);
10
11 typedef int (__stdcall *ad_ptr)(
12     const Nclr::OpenCV::Image<unsigned char, 3> *srcimg,
13     Nclr::OpenCV::Image<unsigned char, 1> *destimg,
14     int *tcnt);
15
16 #endif // AREA_DETECT_HPP_
```

ソースコード B.10: csnake_ad.cpp

```
1  #include "area_detect.hpp"
2
3  #include <vector>
4  #include <cv.h>
5  #include <highgui.h>
6  #include <windows.h>
7  #include <math.h>
8  #include <algorithm>
9  #include <new>
10 #include <string>
11
12 #undef min
13 #undef max
14
15 using namespace Nclr::OpenCV;
16
17 namespace {
18     typedef std::vector<CvPoint> point_vector;
19
20     struct csnake_params
21     {
22         double dist, kurtosis;
23         double wsp1, wsp2, warea, wdist, wedge, wintens;
24     };
25
26     inline double cross(const CvPoint &lhs, const CvPoint &rhs)
27     {
28         return lhs.x * rhs.y - lhs.y * rhs.x;
29     }
```

```
30
31      inline CvPoint operator−(const CvPoint &lhs, const CvPoint &rhs)
32      {
33          return cvPoint( lhs.x − rhs.x, lhs.y − rhs.y );
34      }
35
36      void create_cornerimg(const Image<unsigned char, 1> &src,
37                            Image<unsigned char, 1> &dest)
38      {
39          Image<float, 1> eigimg( src.getwidth(), src.getheight() );
40          Image<float, 1> tmpimg( src.getwidth(), src.getheight() );
41
42          int corner_count = 2000;
43          CvPoint2D32f *corners = (CvPoint2D32f *)cvAlloc( corner_count * sizeof( CvPoint2D32f ) );
44
45          cvGoodFeaturesToTrack( dest, eigimg, tmpimg, corners, &corner_count,
46                                 0.04, 13, NULL, 3 );
47      }
48
49      int move_points(point_vector &pt,
50                  const Image<unsigned char, 1> &srcimg,
51                  const CvSize &wnd,
52                  const csnake_params &params)
53      {
54          const int NB = wnd.width * wnd.height;
55          const int CENTER_X = wnd.width / 2;
56          const int CENTER_Y = wnd.height / 2;
57
58          CvPoint diff;
59          double avg_d = 0;
60          for( std::size_t i = 1; i < pt.size(); ++i ) {
61              diff.x = pt[i − 1].x − pt[i].x;
62              diff.y = pt[i − 1].y − pt[i].y;
63              avg_d += sqrt( (double)diff.x * diff.x + diff.y * diff.y );
64          }
65          diff.x = pt[0].x − pt[pt.size() − 1].x;
66          diff.y = pt[0].y − pt[pt.size() − 1].y;
67          avg_d += sqrt( (double)diff.x * diff.x + diff.y * diff.y );
68          avg_d /= pt.size();
69
70          Image<unsigned char, 1> img( srcimg.getwidth(), srcimg.getheight() );
71          cvCopy( srcimg, img );
72          Image<unsigned char, 1> imgdx( srcimg.getwidth(), srcimg.getheight() );
73          Image<unsigned char, 1> imgdy( srcimg.getwidth(), srcimg.getheight() );
74          cvSmooth( img, img, CV_GAUSSIAN, 5 );
75
76          /*
77            float dxkernel[] = { 1, 0, −1, 2, 0, −2, 1, 0, −1 };
78            float dykernel[] = { 1, 2, 1, 0, 0, 0, −1, −2, −1 };
79            CvMat dxfilter = cvMat( 3, 3, CV_32F, dxkernel );
80            CvMat dyfilter = cvMat( 3, 3, CV_32F, dykernel );
81            cvFilter2D( img, imgdx, &dxfilter );
82            cvFilter2D( img, imgdy, &dyfilter );
83          */
84          cvSobel( img, imgdx, 1, 0 );
85          cvSobel( img, imgdy, 0, 1 );
86
87          int moved = 0;
88          for( std::size_t p = 0; p < pt.size(); ++p ) {
89              std::vector<double> energy( NB, 0 );
90
91              int left = std::min( pt[p].x, wnd.width / 2 );
92              int right = std::min( srcimg.getwidth() − pt[p].x, wnd.width / 2 );
93              int top = std::min( pt[p].y , wnd.height / 2 );
94              int bottom = std::min( srcimg.getheight() − pt[p].y, wnd.height / 2 );
95
96              int pp = p + 1, pm = p − 1;
97              if( p == 0 )
98                  pm = pt.size() − 1;
99              else if( p == pt.size() − 1 )
100                 pp = 0;
101
102             for( int j = −top; j <= bottom; ++j ) {
```

```
103                 for( int i = -left; i <= right; ++i ) {
104                     int tx, ty;
105                     double tmpd;
106
107                     int index = i + CENTER_X + ( j + CENTER_Y ) * wnd.width;
108
109                     // spline energy
110                     tx = ( pt[p].x + i ) - pt[pm].x;
111                     ty = ( pt[p].y + j ) - pt[pm].y;
112                     energy[index] += params.wsp1 * ( tx * tx + ty * ty );
113
114                     tx = pt[pp].x - 2 * ( pt[p].x + i ) + pt[pm].x;
115                     ty = pt[pp].y - 2 * ( pt[p].y + j ) + pt[pm].y;
116                     energy[index] += params.wsp2 * ( tx * tx + ty * ty );
117
118                     energy[index] /= 2.0;
119
120                     // area energyx
121                     tmpd = ( pt[p].x + i ) * ( pt[pp].y - pt[p].y ) -
122                         ( pt[p].y + j ) * ( pt[pp].x - pt[p].x );
123                     energy[index] += params.warea * tmpd / 2.0;
124
125                     // dist energy
126                     tx = ( pt[p].x + i ) - pt[pm].x;
127                     ty = ( pt[p].y + j ) - pt[pm].y;
128                     tmpd = avg_d - sqrt( (double)tx * tx + ty * ty );
129                     energy[index] += params.wdist * ( tmpd * tmpd ) / 2.0;
130
131                     // preprocess edge energy and intens energy
132                     if( pt[p].x + i < 0 )
133                         energy[index] = DBL_MAX;
134                     else if( pt[p].x + i >= srcimg.getwidth() )
135                         energy[index] = DBL_MAX;
136                     else
137                         tx = pt[p].x + i;
138
139                     if( pt[p].y + j < 0 )
140                         energy[index] = DBL_MAX;
141                     else if( pt[p].y + j >= srcimg.getheight() )
142                         energy[index] = DBL_MAX;
143                     else
144                         ty = pt[p].y + j;
145
146                     if( energy[index] != DBL_MAX ) {
147                         // edge energy
148                         energy[index] += params.wedge *
149                             ( imgdx.data( tx, ty ) * imgdx.data( tx, ty ) +
150                               imgdy.data( tx, ty ) * imgdy.data( tx, ty ) ) / (-2.0);
151
152                         // intens energy
153                         energy[index] += params.wintens * srcimg.data( tx, ty );
154                     }
155                 }
156             }
157
158             double e_min = DBL_MAX;
159             int offset_x = 0, offset_y = 0;
160             for( int j = -top; j <= bottom; ++j ) {
161                 for( int i = -left; i <= right; ++i ) {
162                     int index = i + CENTER_X + ( j + CENTER_Y ) * wnd.width;
163                     if( energy[index] < e_min ) {
164                         offset_x = i;
165                         offset_y = j;
166                         e_min = energy[index];
167                     }
168                 }
169             }
170
171             if( offset_x || offset_y ) {
172                 pt[p].x += offset_x;
173                 pt[p].y += offset_y;
174                 ++moved;
175             }
```

```
176         }
177
178         return moved;
179     }
180
181     bool ioep_impl(point_vector &pt,
182                    int i,
183                    const CvPoint &p,
184                    const CvPoint &pp,
185                    const CvPoint &pm,
186                    const csnake_params &params)
187     {
188         CvPoint a, b;
189
190         a.x = pp.x - p.x;
191         a.y = pp.y - p.y;
192         if( sqrt( (double)a.x * a.x + a.y * a.y ) > params.dist ) {
193             pt.insert( pt.begin() + i + 1,
194                        cvPoint( ( pp.x + p.x ) / 2,
195                                 ( pp.y + p.y ) / 2 ) );
196             return true;
197         }
198
199         b.x = pm.x - p.x;
200         b.y = pm.y - p.y;
201         double td1 = sqrt( (double)a.x * a.x + a.y * a.y );
202         double td2 = sqrt( (double)b.x * b.x + b.y * b.y );
203         if( ( a.x * b.x + a.y * b.y ) / ( td1 * td2 ) > params.kurtosis ) {
204             pt.erase( pt.begin() + i );
205             return true;
206         }
207
208         return false;
209     }
210
211     bool check_cross(std::vector<point_vector> &snakes)
212     {
213         bool flg = false;
214         std::vector<point_vector> tmpsnakes;
215
216         for( std::size_t s = 0; s < snakes.size(); ++s ) {
217             point_vector &pt = snakes[s];
218
219             for( std::size_t i = 0; i < pt.size() - 2; ++i ) {
220                 CvPoint v1 = pt[i + 1] - pt[i];
221
222                 std::size_t limit_j = pt.size() - 1;
223                 if( i == 0 )
224                     limit_j = pt.size() - 2;
225
226                 for( std::size_t j = i + 2; j < limit_j; ++j ) {
227                     CvPoint v2 = pt[j + 1] - pt[j];
228                     CvPoint v = pt[j] - pt[i];
229
230                     double c12 = cross( v1, v2 );
231                     double p = cross( v, v1 ) / c12;
232                     double q = cross( v, v2 ) / c12;
233
234                     if( 0 <= p && p <= 1 && 0 <= q && q <= 1 )
235                     {
236                         point_vector buf;
237                         point_vector::iterator itr = pt.begin() + i + 1;
238                         for( int k = i + 1; k <= j; ++k ) {
239                             buf.push_back( *itr );
240                             itr = pt.erase( itr );
241                         }
242
243                         limit_j = pt.size() - 1;
244                         if( i == 0 )
245                             limit_j = pt.size() - 2;
246
247                         if( buf.size() >= 5 )
248                             tmpsnakes.push_back( buf );
249
```

```
250                             flg = true;
251                         }
252                     }
253                 }
254             }
255
256             for( std::size_t i = 0; i < tmpsnakes.size(); ++i )
257                 snakes.push_back( tmpsnakes[i] );
258
259             return flg;
260         }
261
262         bool insert_or_erase_points(point_vector &pt,
263                                     const csnake_params &params)
264         {
265             bool flg = false;
266
267             for( std::size_t i = 1; i < pt.size() - 1; ++i )
268                 flg |= ioep_impl( pt, i, pt[i], pt[i + 1], pt[i - 1], params );
269             flg |= ioep_impl( pt, 0, pt[0], pt[1], pt[pt.size() - 1], params );
270             flg |= ioep_impl( pt, pt.size() - 1, pt[pt.size() - 1], pt[0], pt[pt.size() - 2], params );
271
272             return flg;
273         }
274
275         int csnake(std::vector<point_vector> &snakes,
276                    const Image<unsigned char, 1> &srcimg,
277                    const CvSize &wnd,
278                    const csnake_params &params,
279                    int maxcount,
280                    int th_move)
281         {
282             //image_window snakewnd( "snake" );
283             //Image<unsigned char, 3> img( srcimg.getwidth(), srcimg.getheight() );
284
285             int cnt;
286             for( cnt = 0; cnt < maxcount; ++cnt ) {
287                 bool flg = false;
288
289                 int moved = 0;
290                 for( std::size_t i = 0; i < snakes.size(); ++i )
291                     moved += move_points( snakes[i], srcimg, wnd, params );
292                 if( moved >= th_move )
293                     flg = true;
294
295                 flg |= check_cross( snakes );
296
297                 for( std::vector<point_vector>::iterator itr = snakes.begin(); itr != snakes.end(); ) {
298                     if( itr->size() < 10 )
299                         itr = snakes.erase( itr );
300                     else
301                         ++itr;
302                 }
303
304                 if( !flg ) {
305                     for( std::size_t i = 0; i < snakes.size(); ++i ) {
306                         bool f = insert_or_erase_points( snakes[i], params );
307                         flg = f || flg;
308                     }
309                 }
310
311                 /*
312                 if( IsWindow( snakewnd ) ) {
313                     cvCvtColor( srcimg, img, CV_GRAY2BGR );
314                     draw_snakes( img, snakes );
315                     snakewnd.show( img );
316                     image_window::waitkey( 100 );
317                 }
318                 */
319
320                 if( flg == false )
321                     break;
322             }
323
```

```
324             return cnt;
325         }
326     };
327
328     int __declspec(dllexport) __stdcall area_detect(
329         const Nclr::OpenCV::Image<unsigned char, 3> *srcimg,
330         Nclr::OpenCV::Image<unsigned char, 1> *destimg,
331         int *tcnt)
332     {
333         try
334         {
335             /*
336             char inipath[512];
337             char ininame[11] = "csnake.ini";
338             //strcpy( ininame, "csnake.ini" );
339             GetModuleFileName( NULL, inipath, sizeof( inipath ) );
340             for( int i = 511; i > 0; --i ) {
341                 if( inipath[i] == '\\' ) {
342                     for( int j = 1; j <= strlen( ininame ); ++j )
343                         inipath[i + j] = ininame[j - 1];
344                     inipath[i + strlen( ininame ) + 1] = '\0';
345                     break;
346                 }
347             }
348             printf( "%s\n", inipath );
349             */
350             char inipath[512];
351             std::string ininame = "csnake.ini";
352             GetModuleFileName( NULL, inipath, sizeof( inipath ) );
353             std::string path = inipath;
354             path.erase( path.begin() + path.rfind( '\\' ) + 1, path.end() );
355             path += ininame;
356             strcpy( inipath, path.c_str() );
357
358             printf( "%s\n", inipath );
359
360             int length = GetPrivateProfileInt( "snakes", "length", 0, inipath );
361             int maxcount = GetPrivateProfileInt( "snakes", "maxcount", 0, inipath );
362             int th_move = GetPrivateProfileInt( "snakes", "th_move", 0, inipath );
363
364             csnake_params params;
365             char buf[256];
366             GetPrivateProfileString( "weights", "dist", "", buf, sizeof( buf ), inipath );
367             params.dist = atof( buf );
368             GetPrivateProfileString( "weights", "kurtosis", "", buf, sizeof( buf ), inipath );
369             params.kurtosis = atof( buf );
370             GetPrivateProfileString( "weights", "wsp1", "", buf, sizeof( buf ), inipath );
371             params.wsp1 = atof( buf );
372             GetPrivateProfileString( "weights", "wsp2", "", buf, sizeof( buf ), inipath );
373             params.wsp2 = atof( buf );
374             GetPrivateProfileString( "weights", "warea", "", buf, sizeof( buf ), inipath );
375             params.warea = atof( buf );
376             GetPrivateProfileString( "weights", "wdist", "", buf, sizeof( buf ), inipath );
377             params.wdist = atof( buf );
378             GetPrivateProfileString( "weights", "wedge", "", buf, sizeof( buf ), inipath );
379             params.wedge = atof( buf );
380             GetPrivateProfileString( "weights", "wintens", "", buf, sizeof( buf ), inipath );
381             params.wintens = atof( buf );
382             printf( "length = %d\nmaxcount = %d\nth_move = %d\n", length, maxcount, th_move );
383             printf( "dist = %f\nkurtosis=%f\n", params.dist, params.kurtosis );
384             printf( "wsp1 = %f, wsp2 = %f\n", params.wsp1, params.wsp2 );
385             printf( "warea = %f\nwdist=%f\nwedge=%f\nwintens=%f\n",
386                     params.warea, params.wdist, params.wedge, params.wintens );
387             printf( "\n" );
388
389             Image<unsigned char, 1> grayimg( srcimg->getwidth(), srcimg->getheight() );
390             cvCvtColor( *srcimg, grayimg, CV_BGR2GRAY );
391
392             std::vector<point_vector> snakes;
393             snakes.push_back( point_vector() );
394             int ptlen = length / 4;
395             int w = srcimg->getwidth() / ptlen;
396             int h = srcimg->getheight() / ptlen;
```

```
397             for( int i = 0; i < srcimg->getwidth(); i += w )
398                 snakes.back().push_back( cvPoint( i, 0 ) );
399             for( int i = 0; i < srcimg->getheight() - 1; i += h )
400                 snakes.back().push_back( cvPoint( srcimg->getwidth() - 1, i ) );
401             for( int i = srcimg->getwidth() - 1; i >= 0; i -= w )
402                 snakes.back().push_back( cvPoint( i, srcimg->getheight() - 1 ) );
403             for( int i = srcimg->getheight() - 1; i > 0; i -= h )
404                 snakes.back().push_back( cvPoint( 0, i ) );
405
406             int cnt = csnake( snakes, grayimg, cvSize( 15, 15 ), params, maxcount, th_move );
407             if( tcnt != NULL )
408                 *tcnt = cnt;
409
410             std::vector<CvPoint> center;
411             for( std::size_t i = 0; i < snakes.size(); ++i ) {
412                 CvPoint p = cvPoint( 0, 0 );
413                 for( std::size_t j = 0; j < snakes[i].size() - 1; ++j ) {
414                     cvLine( *destimg, snakes[i][j], snakes[i][j + 1], cvScalarAll( 255 ) );
415                     p.x += snakes[i][j].x;
416                     p.y += snakes[i][j].y;
417                 }
418                 cvLine( *destimg, snakes[i][snakes[i].size() - 1], snakes[i][0], cvScalarAll( 255 ) );
419                 p.x += snakes[i][snakes[i].size() - 1].x;
420                 p.y += snakes[i][snakes[i].size() - 1].y;
421
422                 p.x /= snakes[i].size();
423                 p.y /= snakes[i].size();
424                 center.push_back( p );
425             }
426
427             for( std::size_t i = 0; i < center.size(); ++i ) {
428                 cvFloodFill( *destimg, center[i], cvScalarAll( 255 ) );
429             }
430
431             return 0;
432         }
433         catch( std::runtime_error &e ) {
434             fprintf( stderr, "%s\n", e.what() );
435             return 1;
436         }
437         catch( std::bad_alloc &e ) {
438             fprintf( stderr, "%s\n", e.what() );
439             return 1;
440         }
441     }
```

ソースコード B.11: vb_ad.cpp

```
1   #include "area_detect.hpp"
2
3   #include <stdlib.h>
4   #include <string.h>
5   #include <windows.h>
6   #include <vector>
7   #include <algorithm>
8   #include "nclr_cv_mat.hpp"
9   #include "variate_bayes.hpp"
10  #include "timecounter.hpp"
11
12  using namespace Nclr::OpenCV;
13
14  namespace
15  {
16      int binary_otsu(const Image<unsigned char, 1> &src, Image<unsigned char, 1> &dest, int
            maxval = 255)
17      {
18          const int N = src.getwidth() * src.getheight();
19          const int L = 256;
20
21          int hist[L];
22          double p[L];
23          for( int i = 0; i < N; ++i )
24              ++hist[src.data( i, 0 )];
```

```
25          for( int i = 0; i < L; ++i )
26              p[i] = (double)hist[i] / N;
27
28          int max_th = 0;
29          double max_sigma = DBL_MIN;
30          for( int k = 1; k < L; ++k ) {
31              double ur = 0;
32              double u[2] = { 0, 0 };
33              double w[2] = { 0, 0 };
34
35              for( int i = 0; i < k; ++i )
36                  w[0] += p[i];
37              for( int i = k; i < L; ++i )
38                  w[1] += p[i];
39              for( int i = 0; i < k; ++i ) {
40                  u[0] += i * p[i] / w[0];
41                  ur += i * p[i];
42              }
43              for( int i = k; i < L; ++i ) {
44                  u[1] += i * p[i] / w[1];
45                  ur += i * p[i];
46              }
47
48              double sigma = w[0] * ( u[0] - ur ) * ( u[0] - ur ) + w[1] * ( u[1] - ur ) * ( u[1] - ur );
49              if( sigma > max_sigma ) {
50                  max_th = k;
51                  max_sigma = sigma;
52              }
53          }
54
55          cvThreshold( src, dest, max_th, maxval, CV_THRESH_BINARY );
56
57          return max_th;
58      }
59
60      void createdata(const Image<unsigned char, 3> &src,
61                      std::vector<Matrix<double> > &data,
62                      const char *inipath)
63      {
64          time_counter tc;
65
66          char buf[512];
67          int median = GetPrivateProfileInt( "filter", "median", 0, inipath );
68          int gaussian = GetPrivateProfileInt( "filter", "gaussian", 0, inipath );
69          int reverse = GetPrivateProfileInt( "filter", "reverse", 0, inipath );
70          int corner_count = GetPrivateProfileInt( "corner", "maxcount", 0, inipath );
71          GetPrivateProfileString( "corner", "quality", "", buf, sizeof( buf ), inipath );
72          double quality = atof( buf );
73          GetPrivateProfileString( "corner", "mindist", "", buf, sizeof( buf ), inipath );
74          double mindist = atof( buf );
75          int block_size = GetPrivateProfileInt( "corner", "block_size", 0, inipath );
76          int use_harris = GetPrivateProfileInt( "corner", "use_harris", 0, inipath );
77          GetPrivateProfileString( "corner", "harris_param", "", buf, sizeof( buf ), inipath );
78          double harris_param = atof( buf );
79
80          printf( "maxcount = %d, quality = %f, mindist = %f\n", corner_count, quality, mindist );
81          printf( "use_harris = %d, harris_param = %f\n", use_harris, harris_param );
82          CvPoint2D32f *corners = (CvPoint2D32f *)cvAlloc( corner_count * sizeof( CvPoint2D32f ) );
83
84          tc.begin();
85
86          Image<unsigned char, 1> img( src.getwidth(), src.getheight() );
87
88          Image<unsigned char, 3> hsv( src.getwidth(), src.getheight() );
89          cvCvtColor( src, hsv, CV_BGR2HSV );
90          for( int y = 0; y < hsv.getheight(); ++y ) {
91              for( int x = 0; x < hsv.getwidth(); ++x ) {
92                  img.data( x, y ) = hsv.data( x, y, 2 );
93              }
94          }
95
96          /*
97          float kernel[] = { 1, 1, 1, 1, -8, 1, 1, 1, 1 };
```

```
98          CvMat filter = cvMat( 3, 3, CV_32F, kernel );
99          cvSmooth( img, img, CV_MEDIAN, 3, 0, 0, 0 );
100         cvFilter2D( img, img, &filter );
101         binary_otsu( img, img );
102
103         data.clear();
104         for( int y = 0; y < img.getheight(); ++y ) {
105             for( int x = 0; x < img.getwidth(); ++x ) {
106                 if( img.data( x, y ) == 255 ) {
107                     data.push_back( Matrix<double>( 2, 1 ) );
108                     data.back().set( 0, 0, x / (double)src.getwidth() );
109                     data.back().set( 1, 0, y / (double)src.getheight() );
110                 }
111             }
112         }
113
114         if( data.size() > 200 ) {
115             std::random_shuffle( data.begin(), data.end() );
116             data.erase( data.begin() + 200, data.end() );
117         }
118         */
119
120         printf( "data_size_=_%d\n", data.size() );
121
122         cvCvtColor( src, img, CV_BGR2GRAY );
123         if( !reverse ) {
124             if( median != 0 )
125                 cvSmooth( img, img, CV_MEDIAN, median );
126             if( gaussian != 0 )
127                 cvSmooth( img, img, CV_GAUSSIAN, gaussian );
128         }
129         else {
130             if( gaussian != 0 )
131                 cvSmooth( img, img, CV_GAUSSIAN, gaussian );
132             if( median != 0 )
133                 cvSmooth( img, img, CV_MEDIAN, median );
134         }
135
136         Image<float, 1> eigimg( src.getwidth(), src.getheight() );
137         Image<float, 1> tmpimg( src.getwidth(), src.getheight() );
138
139         cvGoodFeaturesToTrack( img, eigimg, tmpimg, corners, &corner_count, quality, mindist,
140                                NULL, 3, use_harris, harris_param );
141         //data.clear();
142         for( int i = 0; i < corner_count; ++i ) {
143             data.push_back( Matrix<double>( 2, 1 ) );
144             data.back().set( 0, 0, corners[i].x / (double)src.getwidth() );
145             data.back().set( 1, 0, corners[i].y / (double)src.getheight() );
146         }
147
148         printf( "data_size_=_%d\n%f[ms]\n", data.size(), tc.end() );
149
150         cvFree( &corners );
151
152         /*
153         Image<unsigned char, 3> dataimg( src.getwidth(), src.getheight() );
154         for( int y = 0; y < dataimg.getheight(); ++y ) {
155             for( int x = 0; x < dataimg.getwidth(); ++x ) {
156                 for( int c = 0; c < 3; ++c )
157                     dataimg.data( x, y, c ) = img.data( x, y, c );
158             }
159         }
160         for( std::size_t i = 0; i < data.size(); ++i ) {
161             cvCircle( dataimg,
162                       cvPointFrom32f( cvPoint2D32f(
163                                           data[i].get( 0, 0 ) * src.getwidth(),
164                                           data[i].get( 1, 0 ) * src.getheight() ) ),
165                       3, CV_RGB( 0, 0, 255 ), 3 );
166         }
167
168         cvNamedWindow( "data" );
169         cvShowImage( "data", dataimg );
170         cvWaitKey( 0 );
```

```
171             cvDestroyWindow( "data" );
172             */
173
174             printf( "vb_ad_:_createdata_completed\n" );
175         }
176 };
177
178 int __declspec(dllexport) __stdcall area_detect(
179         const Nclr::OpenCV::Image<unsigned char, 3> *srcimg,
180         Nclr::OpenCV::Image<unsigned char, 1> *destimg,
181         int *tcnt)
182 {
183         time_counter tc;
184
185         char inipath[512];
186         GetCurrentDirectory( sizeof( inipath ), inipath );
187         strcat( inipath, "\\vb_ad.ini" );
188
189         std::vector<Matrix<double> > data;
190         createdata( *srcimg, data, inipath );
191
192         char buf[512];
193         int cluster = GetPrivateProfileInt( "vb", "cluster", 0, inipath );
194         int maxcnt = GetPrivateProfileInt( "vb", "maxcount", 0, inipath );
195         GetPrivateProfileString( "vb", "param", "", buf, sizeof( buf ), inipath );
196         double param = atof( buf );
197         GetPrivateProfileString( "vb", "threshold", "", buf, sizeof( buf ), inipath );
198         double threshold = atof( buf );
199
200         tc.begin();
201
202         VariateBayes vb( 2, cluster );
203         int t = vb.Learn( data, param, maxcnt );
204         if( tcnt != NULL )
205             *tcnt = t;
206
207         int width = srcimg->getwidth();
208         int height = srcimg->getheight();
209
210         std::vector<double> pd( width * height, 0 );
211         double nrml = 0;
212         for( int y = 0; y < height; ++y ) {
213             for( int x = 0; x < width; ++x ) {
214                 Matrix<double> p( 2, 1 );
215                 p.set( 0, 0, x / (double)width );
216                 p.set( 1, 0, y / (double)height );
217                 pd[x + width * y] = vb.Predict( p );
218                 if( pd[x + width * y] > nrml )
219                     nrml = pd[x + width * y];
220             }
221         }
222
223         for( int y = 0; y < height; ++y ) {
224             for( int x = 0; x < width; ++x ) {
225                 if( ( pd[x + width * y] / nrml ) > threshold )
226                     destimg->data( x, y ) = 255;
227             }
228         }
229
230         printf( "%f[ms]\nvb_ad_completed\n", tc.end() );
231
232         return 0;
233 }
```

ソースコード B.12: image_divide.hpp

```
1  #ifndef IMAGE_DIVIDE_HPP_
2  #define IMAGE_DIVIDE_HPP_
3
4  #include <vector>
5  #include "nclr_cv_image.hpp"
6
7  struct imgdiv_info
```

```
 8 {
 9     std::vector<CvScalar> color;
10     std::vector<int> area_cnt;
11     Nclr::OpenCV::Image<unsigned short, 1> *data;
12 };
13
14 int __declspec(dllexport) __stdcall image_divide(
15     const Nclr::OpenCV::Image<unsigned char, 3> *srcimg,
16     imgdiv_info *dest);
17
18 typedef int (__stdcall *imgdiv_ptr)(
19     const Nclr::OpenCV::Image<unsigned char, 3> *srcimg,
20     imgdiv_info *dest);
21
22 #endif // IMAGE_DIVIDE_HPP_
```

ソースコード B.13: empty_imgdiv.cpp

```
1  #include "image_divide.hpp"
2
3  int __declspec(dllexport) __stdcall image_divide(
4      const Nclr::OpenCV::Image<unsigned char, 3> *srcimg,
5      imgdiv_info *dest)
6  {
7      dest->color.push_back( cvScalar( 0, 0, 0, 0 ) );
8      dest->area_cnt.push_back( srcimg->getwidth() * srcimg->getheight() );
9      cvZero( *(dest->data) );
10
11     return 0;
12 }
```

ソースコード B.14: hcseg_imgdiv.cpp

```
 1 #include "image_divide.hpp"
 2
 3 #include <string.h>
 4 #include <windows.h>
 5 #include <vector>
 6 #include <stack>
 7 #include <assert.h>
 8 #include "hcseg.hpp"
 9
10 using namespace Nclr::OpenCV;
11
12 namespace
13 {
14     /*
15     struct scalar3
16     {
17         int val[3];
18     };
19
20     int hc_next(const std::vector<unsigned int> &hist, int p, int &itr, bool flg)
21     {
22         int prev = p - itr;
23         int next = p + itr;
24
25         if( prev < 0 ) {
26             if( flg ) {
27                 prev = hist.size() - 1;
28
29                 if( hist[prev] == hist[p] && hist[p] == hist[next] )
30                     ++itr;
31                 else if( hist[prev] < hist[p] && hist[p] <= hist[next] )
32                     p = next;
33                 else if( !( hist[p] > hist[prev] && hist[p] > hist[next] ) )
34                     p = prev;
35             }
36             else {
37                 if( hist[p] <= hist[next] )
38                     p = next;
39             }
40         }
```

```
41          else if( next >= hist.size() ) {
42              if( flg ) {
43                  next = 0;
44
45                  if( hist[prev] == hist[p] && hist[p] == hist[next] )
46                      ++itr;
47                  else if( hist[prev] < hist[p] && hist[p] <= hist[next] )
48                      p = next;
49                  else if( !( hist[p] > hist[prev] && hist[p] > hist[next] ) )
50                      p = prev;
51              }
52              else {
53                  if( hist[p] < hist[prev] )
54                      p = prev;
55              }
56          }
57          else {
58              if( hist[prev] == hist[p] && hist[p] == hist[next] )
59                  ++itr;
60              else if( hist[prev] < hist[p] && hist[p] <= hist[next] )
61                  p = next;
62              else if( !( hist[p] > hist[prev] && hist[p] > hist[next] ) )
63                  p = prev;
64          }
65
66          return p;
67      }
68
69      void hc_peak(std::vector<CvScalar> &peaks,
70                   std::vector<int> &histpeak,
71                   const std::vector<unsigned int> hist[3],
72                   int h, int s, int v)
73      {
74          int tp[3] = { h, s, v };
75          int ptp[3] = { h, s, v };
76          std::size_t quant[3];
77          for( int i = 0; i < 3; ++i )
78              quant[i] = hist[i].size();
79
80          std::stack<scalar3> moved;
81
82          for( int i = 0; i < 3; ++i ) {
83              int itr = 1, pitr = itr;
84              for(;;) {
85                  tp[i] = hc_next( hist[i], tp[i], itr, i == 0 ? true : false );
86
87                  scalar3 t;
88                  for( int j = 0; j < 3; ++j )
89                      t.val[j] = tp[j];
90                  moved.push( t );
91
92                  if( itr == pitr ) {
93                      itr = 1;
94
95                      if( tp[i] == ptp[i] )
96                          break;
97                  }
98
99                  ptp[i] = tp[i];
100                 pitr = itr;
101             }
102         }
103
104         std::vector<scalar3> tmp_peaks;
105         std::size_t i;
106         for( i = 0; i < tmp_peaks.size(); ++i ) {
107             if( tmp_peaks[i].val[0] == tp[0] && tmp_peaks[i].val[1] == tp[1] && tmp_peaks[i].val[2]
                    == tp[2] )
108                 break;
109         }
110         if( i == tmp_peaks.size() ) {
111             peaks.push_back( cvScalar( tp[0], tp[1], tp[2], 0 ) );
112             scalar3 t;
113             for( int i = 0; i < 3; ++i )
```

```
114                 t.val[i] = tp[i];
115             tmp_peaks.push_back( t );
116
117             histpeak[h + s * quant[0] + v * quant[0] * quant[1]] = peaks.size() - 1;
118             while( !moved.empty() ) {
119                 scalar3 &t = moved.top();
120                 histpeak[t.val[0] + t.val[1] * quant[0] + t.val[2] * quant[0] * quant[1]] = peaks.size()
                        - 1;
121                 moved.pop();
122             }
123         }
124         else {
125             histpeak[h + s * quant[0] + v * quant[0] * quant[1]] = i;
126             while( !moved.empty() ) {
127                 scalar3 &t = moved.top();
128                 histpeak[t.val[0] + t.val[1] * quant[0] + t.val[2] * quant[0] * quant[1]] = i;
129                 moved.pop();
130             }
131         }
132     }
133     */
134
135     /*
136     void hc_segment(const Image<unsigned char, 3> &src,
137                     Image<unsigned short, 1> &segment,
138                     std::vector<CvScalar> &peaks,
139                     std::vector<int> &segcnt,
140                     int qh, int qs, int qv)
141     {
142         Image<float, 3> src_nrml( src.getwidth(), src.getheight() );
143         Image<unsigned short, 3> src_s( src.getwidth(), src.getheight() );
144         cvZero( src_s );
145         cvConvert( src, src_nrml );
146         for( int y = 0; y < src.getheight(); ++y ) {
147             for( int x = 0; x < src.getwidth(); ++x ) {
148                 for( int i = 0; i < 3; ++i )
149                     src_nrml.data( x, y, i ) /= 255.0f;
150             }
151         }
152         cvCvtColor( src_nrml, src_nrml, CV_BGR2HSV );
153         for( int y = 0; y < src.getheight(); ++y ) {
154             for( int x = 0; x < src.getwidth(); ++x )
155                 src_nrml.data( x, y, 0 ) /= 360.0f;
156         }
157
158         int quant[3] = { qh, qs, qv };
159         float quant_f[3];
160         std::vector<unsigned int> hist[3];
161         for( int i = 0; i < 3; ++i ) {
162             quant_f[i] = 1.0f / quant[i];
163             hist[i].resize( quant[i], 0 );
164         }
165
166         for( int y = 0; y < src.getheight(); ++y ) {
167             for( int x = 0; x < src.getwidth(); ++x ){
168                 for( int i = 0; i < 3; ++i ) {
169                     float val = src_nrml.data( x, y, i );
170                     float q = 0;
171                     for( int j = 0; j < quant[i]; ++j ) {
172                         if( q <= val && val < q + quant_f[i] ) {
173                             src_s.data( x, y, i ) = j;
174                             if( i != 0 || ( i == 0 && src_nrml.data( x, y, 0 ) != 0 ) ) {
175                                 ++hist[i][j];
176                             }
177                             break;
178                         }
179                         q += quant_f[i];
180                     }
181                 }
182             }
183         }
184
185         std::vector<int> histpeak( quant[0] * quant[1] * quant[2], -1 );
```

```
186             for( int v = 0; v < quant[2]; ++v ) {
187                 for( int s = 0; s < quant[1]; ++s ) {
188                     for( int h = 0; h < quant[0]; ++h ) {
189                         if( histpeak[h + s * quant[0] + v * quant[0] * quant[1]] >= 0 )
190                             break;
191 
192                         hc_peak( peaks, histpeak, hist, h, s, v );
193                     }
194                 }
195             }
196 
197             for( std::size_t i = 0; i < peaks.size(); ++i ) {
198                 for( int j = 0; j < 3; ++j )
199                     peaks[i].val[j] /= quant[j];
200             }
201             segcnt.resize( peaks.size(), 0 );
202             for( int y = 0; y < src.getheight(); ++y ) {
203                 for( int x = 0; x < src.getwidth(); ++x ) {
204                     int pos = src_s.data( x, y, 0 ) + src_s.data( x, y, 1 ) * quant[0]
205                         + src_s.data( x, y, 2 ) * quant[0] * quant[1];
206                     segment.data( x, y ) = histpeak[pos];
207                     ++segcnt[histpeak[pos]];
208                 }
209             }
210         }
211         */
212 
213         int labeling(const imgdiv_info &info, Image<unsigned short, 1> &dest)
214         {
215             Image<float, 1> doneseg( info.data->getwidth(), info.data->getheight() );
216             Image<float, 1> tmpdest( info.data->getwidth(), info.data->getheight() );
217             for( int y = 0; y < doneseg.getheight(); ++y ) {
218                 for( int x = 0; x < doneseg.getwidth(); ++x ) {
219                     doneseg.data( x, y ) = info.data->data( x, y );
220                     tmpdest.data( x, y ) = info.data->data( x, y );
221                 }
222             }
223 
224             int num = 0;
225             for( int y = 0; y < doneseg.getheight(); ++y ) {
226                 for( int x = 0; x < doneseg.getwidth(); ++x ) {
227                     if( fabs( doneseg.data( x, y ) - (-1) ) <= 1.0e-8 )
228                         continue;
229 
230                     cvFloodFill( tmpdest, cvPoint( x, y ), cvScalarAll( num ) );
231                     cvFloodFill( doneseg, cvPoint( x, y ), cvScalarAll( -1 ) );
232                     ++num;
233                 }
234             }
235 
236             for( int y = 0; y < doneseg.getheight(); ++y ) {
237                 for( int x = 0; x < doneseg.getwidth(); ++x ) {
238                     dest.data( x, y ) = (unsigned short)(tmpdest.data( x, y ));
239                 }
240             }
241 
242             return num;
243         }
244 };
245 
246 int __declspec(dllexport) __stdcall image_divide(
247         const Nclr::OpenCV::Image<unsigned char, 3> *srcimg,
248         imgdiv_info *dest)
249 {
250         char inipath[512];
251         GetCurrentDirectory( sizeof( inipath ), inipath );
252         strcat( inipath, "\\hcseg_imgdiv.ini" );
253 
254         int qh = GetPrivateProfileInt( "hcseg", "h", 0, inipath );
255         int qs = GetPrivateProfileInt( "hcseg", "s", 0, inipath );
256         int qv = GetPrivateProfileInt( "hcseg", "v", 0, inipath );
257         int median = GetPrivateProfileInt( "param", "median", 0, inipath );
258         int gaussian = GetPrivateProfileInt( "param", "gaussian", 0, inipath );
```

```
259     int reverse = GetPrivateProfileInt( "param", "reverse", 0, inipath );
260     int use_labeling = GetPrivateProfileInt( "param", "use_labeling", 0, inipath );
261
262     Image<unsigned char, 3> img( srcimg->getwidth(), srcimg->getheight() );
263     cvCopy( *srcimg, img );
264     if( !reverse ) {
265         if( median != 0 )
266             cvSmooth( img, img, CV_MEDIAN, median, 0, 0, 0 );
267         if( gaussian != 0 )
268             cvSmooth( img, img, CV_GAUSSIAN, gaussian, 0, 0, 0 );
269     }
270     else {
271         if( gaussian != 0 )
272             cvSmooth( img, img, CV_GAUSSIAN, gaussian, 0, 0, 0 );
273         if( median != 0 )
274             cvSmooth( img, img, CV_MEDIAN, median, 0, 0, 0 );
275     }
276
277     Segment seg( srcimg->getwidth(), srcimg->getheight() );
278     SegPeaks peaks;
279     SegCount segcnt;
280     HCSegment( img, seg, peaks, segcnt, qh, qs, qv );
281
282     dest->color = peaks;
283     dest->area_cnt = segcnt;
284     cvCopy( seg, *(dest->data) );
285
286     /*
287     Image<unsigned short, 1> seg( srcimg->getwidth(), srcimg->getheight() );
288     std::vector<CvScalar> peaks;
289     std::vector<int> segcnt;
290     hc_segment( img, seg, peaks, segcnt, qh, qs, qv );
291
292     dest->color = peaks;
293     dest->area_cnt = segcnt;
294     cvCopy( seg, *(dest->data) );
295     */
296
297     if( use_labeling != 0 ) {
298         Image<unsigned short, 1> limg( srcimg->getwidth(), srcimg->getheight() );
299         int lnum = labeling( *dest, limg );
300
301         imgdiv_info tmpinfo;
302         for( int i = 0; i < lnum; ++i ) {
303             tmpinfo.color.push_back( cvScalarAll( 0 ) );
304             tmpinfo.area_cnt.push_back( 0 );
305         }
306
307         for( int y = 0; y < limg.getheight(); ++y ) {
308             for( int x = 0; x < limg.getwidth(); ++x ) {
309                 for( int c = 0; c < 3; ++c )
310                     tmpinfo.color[limg.data( x, y )].val[c] = dest->color[dest->data->data( x, y )].
                            val[c];
311                 ++tmpinfo.area_cnt[limg.data( x, y )];
312             }
313         }
314
315         dest->color = tmpinfo.color;
316         dest->area_cnt = tmpinfo.area_cnt;
317         cvCopy( limg, *dest->data );
318     }
319
320     printf( "peaks = %d\n", peaks.size() );
321     printf( "hcseg_imgdiv_complete\n" );
322
323     return 0;
324 }
```