遠隔探査画像から建物面積の推定を行う基礎的手法の比較

S043027

加藤　広朗

計算機科学講座

田中研究室

平成 20 年 3 月 22 日

目次

第1章　　序論

1.1　はじめに

近年ではコンピュータの発達により画像処理技術が向上し，画像は益々鮮明なものになってきている．その中でも衛星画像や航空画像といった，遠隔探査によって写された画像の進歩は目を見張るものがある．そこで遠隔探査画像から建物を判別し，その建物面積の推定を行う事を本研究の目的とする．

1.2　実験概要

本研究では，画像中における Edge 強調を行う方法として Canny 法を使用し，建物面積を推定する手法として，動的輪郭モデル(Active Contour Model：ACM，Snake)と Freeman Code を用いた．実験に使用するデータは，建物に見立てたサンプル画像を作成し，そのサンプル画像を用いてシミュレートを行いそれぞれの手法の動作を確認した．

そして，実画像を用いて実験を行った．

第２章　面積測定領域抽出

建物面積を抽出するために，まず画像中の輪郭線を強調する手法として，Canny 法を用いた．そして，輪郭線を強調した画像から建物面積を推定する領域を抽出するために，今回は以下の２つの手法を用いた．

- 動的輪郭モデル（Active Contour Model：ACM，Snakes）
- Freeman Code

本章では，それぞれの手法の説明と，建物面積を推定する方法について説明する．

2.1　Canny 法

Canny 法は，Edge 強調を行う手法である[1]．Canny 法の特徴は，Edge 強調を行う為に閾値を２つ使用する．閾値を２つ使用することにより，一般的な Edge 強調では見落とされるような弱い Edge も，強い Edge に関連している場合は出力する．主な処理としては以下である．

（１）　ガウシアンフィルタを用いた画像の平滑化
（２）　微分処理
（３）　非最大値の除去
（４）　閾値による Edge 抽出

今回，Canny 法による画像の輪郭強調は，数値解析などに用いられる MATLAB ver5.3 を用いて行った．

図 2.1.1　Canny 法使用前

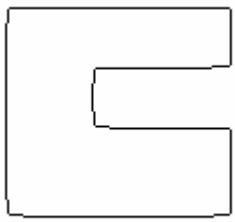


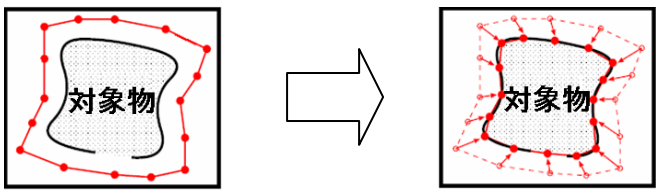
図 2.1.2　Canny 法使用後

2.2　動的輪郭モデル（Active Contour Model：ACM，Snakes）

動的輪郭モデルは M.Kass らによって提案された手法[2]で，最初に対象物に対して，境界線として閉曲線$V(s)=(x(s),y(s))\quad(0\le s\le 1)$を与える．そして，境界線が対象物の輪郭線と一致するように近づけていく．境界線を近づけるために，境界線に働く力として，内部エネルギーE_{int}と外部エネルギーE_{image}を計算し，その和であるE_{ACM}のエネルギーが最小になるように境界線を修正し，輪郭線の抽出を行う．

$$E_{ACM}=\int_0^1\{E_{\text{int}}(V(s))+E_{image}(V(s))\}ds$$

内部エネルギーE_{int}は，境界線の周囲長と複雑さを表しており，それぞれ境界線の一次微分と二次微分に関するエネルギーである．外部エネルギーE_{image}は，境界線に沿った画像の濃淡勾配を表しており，画像濃度の一次微分に関するエネルギーである．これらの計算を行う．各輪郭点に対して計算前の点$(x(i),y(i))$と計算後の点$(x(i+1),y(i+1))$の距離を求め，初期に与えた距離比較用パラメータ$d\max,d\min$を元に，x,yそれぞれ$d(i,i+1)>d\max$ならばiと$i+1$の中間を新しい境界点とする．$d(i,i+1)<d\max$ならばiと$i+1$のうち，輪郭線に近い方の値を次の輪郭点とする．このような計算を輪郭線と境界線が一致するように何回も繰返し行う．



今回，動的輪郭モデルによる実験は，IMAGE ANALYSIS AND COMMUNICATIONS LAB[3]よりデモを用いて行った．尚，本研究で使用したソースコードを付録に示す．

2.3　Freeman Code

Freeman Code は H.Freeman らによって提案された手法[4]で，図の様に隣接する８つの画素に対してコードを与え，輪郭線が連結している方向をそのコードで示し，輪郭線をコードで表すというものである．本研究では，輪郭線によって囲まれた領域の面積抽出を目的としているため，コード化は行わなかった．

Freeman Code を用いた輪郭線抽出の主なフローチャートは次の様である．尚，付録に本研究で使用したソースコードを示す．

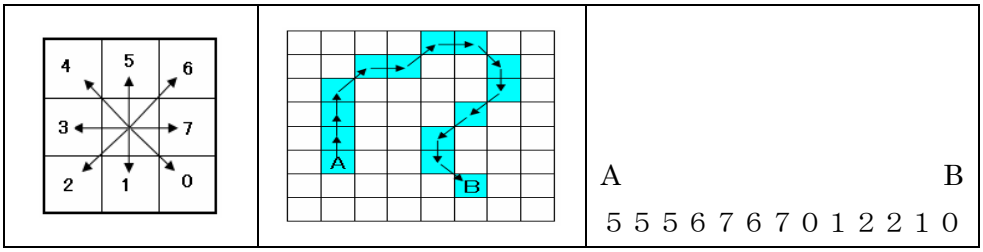


図 2.3.1 Freeman Code におけるコード配置と例

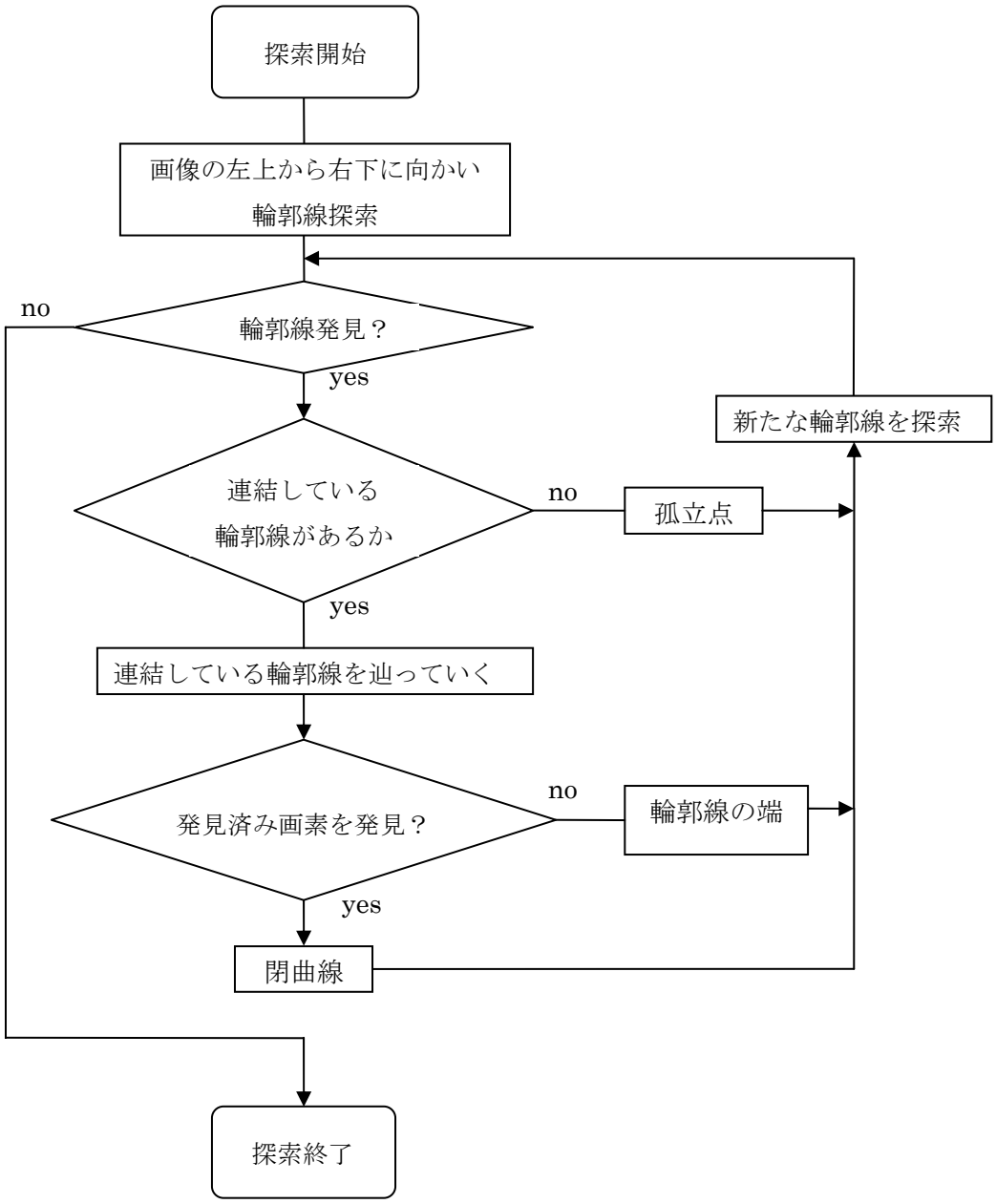


図 2.3. Freeman Code フローチャート

2.3　面積測定方法

建物面積を推定する為に，それぞれの手法によって抽出された面積測定領域の中にある画素数を数える．画素数を数える方法は，画像に対して上下左右，四方向からそれぞれ輪郭線の本数を数え，輪郭線が奇数本目と偶数本目の間にある画素に印を付けていく．四方向全てから印をつけ終わったあと，閉ざされた領域内の画素は印が重なっているので，その重なっている画素数を数えるという方法を考案した．主なアルゴリズムは以下のようである．

（１）画像の左上から右下へ向かって横方向に輪郭線の本数を数える
（２）輪郭線が奇数本目と偶数本目の間にある画素に印を付ける
（３）画像の左上から右下へ向かって縦方向に輪郭線の本数を数える
（４）（２）と同処理
（５）画像の右下から左上へ向かって横方向に輪郭線の本数を数える
（６）（２）と同処理
（７）画像の右下から左上へ向かって縦方向に輪郭線の本数を数える
（８）（２）と同処理
（９）目印が重なっている画素数を数える

この方法で行うと，数えた輪郭線が奇数本だった場合，どの２本の間の画素数を数えればよいのかわからない．そのための不要な印を除去する方法として，上記アルゴリズムで印が付くためにはその１マス前の画素に輪郭線がなければならない．そこで，

（＊）輪郭線が存在していないのに，印が付いている箇所の印を除去する

という処理を上記アルゴリズムの（２），（４），（６），（８）の後に行った．

以下の図中の１を輪郭線，２を面積測定用の目印とする．

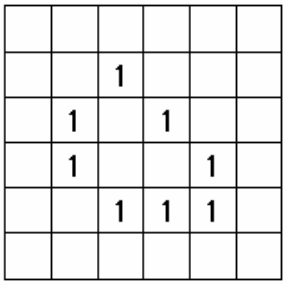


図 2.3.1　処理前

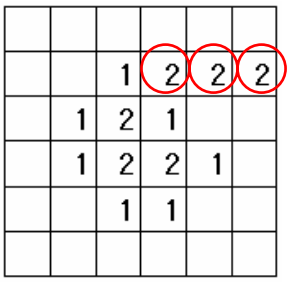


図 2.3.2 目印処理後

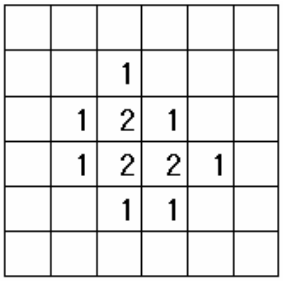
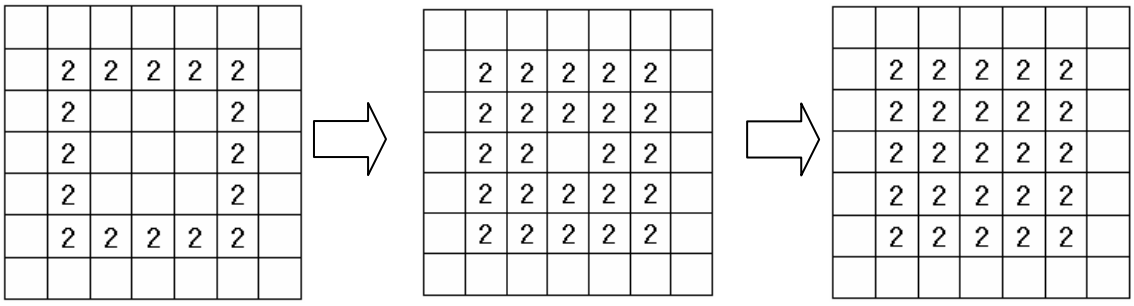


図 2.3.3 除去作業後

図 2.3.1 に対して，上記アルゴリズム（２）を行った例が図 2.3.2 で，図 3.2.3 中の丸印がある箇所が不要な印部分で,不要な印の除去作業を行った結果の例が図2.3.3である.尚，本研究で使用したソースコードを付録に示す.

この他に，画像処理サブルーチン・パッケージ（SPIDER）に画素数測定の方法が記載されているが，この方法は予め一つの閉曲線で区切られている領域に印が付いている状態であり，その印の数を数えるという方法である．本研究で行った実験結果では，そのような状態になっていないため直接この方法を使用することはできない．そこで，Freeman Code を用いて印を付けることができないのかと考えた．印を付ける方法としては，

（１） まず閉曲線である輪郭線に印をつける
（２） 印を付けた画素より内側の画素に印をつける
（３） （２）を繰り返していき，閉曲線で囲まれた領域内の画素全てに印をつける



この方法では（１）を行うときには目印となる輪郭線があるため容易に印をつけることができるが，（２），（３）を行うときには目印がないので，どのようにして輪郭線の内側の画素であるのかという判断がつかないため，この方法を実装することは出来なかった.

第３章　実験結果

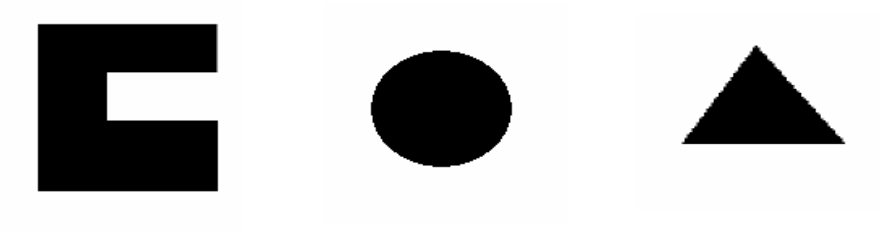
本研究では，用いた手法の動作を確認する為に，建物に見立てたサンプル画像を作成した．そして，動作確認後，実画像を用いて実験を行った．

本章ではそれぞれの実験結果についてまとめる．

3.1　サンプル画像による実験

3.1.1　作成したサンプル画像

実験を行う為に作成したサンプル画像が図 3.1.1 である．



polygon　　circle　　triangle

図 3.1.1　サンプル画像

図 3.1.1 のサンプル画像に対して Canny 法を使用した結果が図 3.1.2 である．

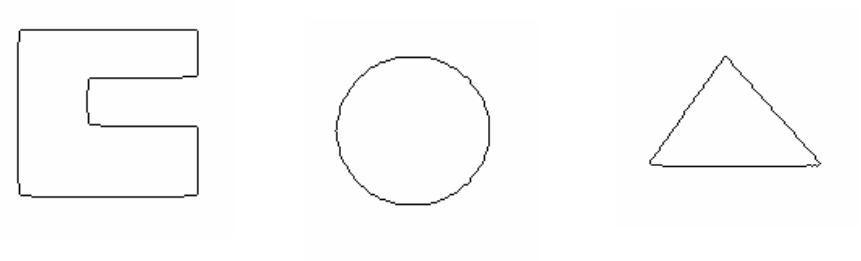


図 3.1.2　Canny 法使用後のサンプル画像

3.1.2　動的輪郭モデルによる結果と考察

図 3.1.2 に対して動的輪郭モデルの実行結果を表 3.1.1 に示す．

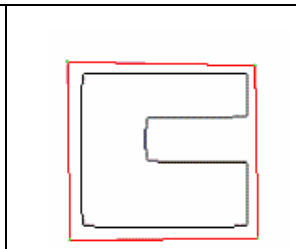
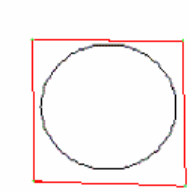
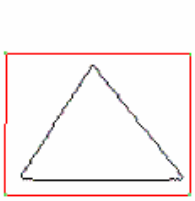
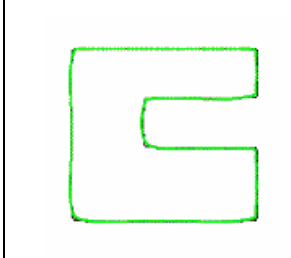
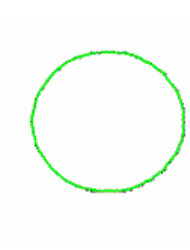
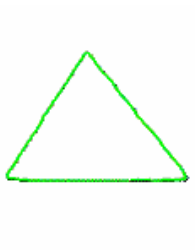
実行前			
実行後			

表 3.1.1 サンプル画像による動的輪郭モデルの実行結果

表 3.1.1 の実行前画像にある赤線が対象のサンプル画像に対して，初期値として与えた境界線である．境界線と輪郭線が一致するまでのループ回数は，対象の輪郭線の複雑さや，境界線と輪郭線との距離によって増加する．今回サンプル画像の circle と triangle はループ回数 100 回程度で一致したのに対し, polygon では 1500～1700 回程度で一致した. 特に, polygon の画像では凹んだ部分を一致させるのに多くのループ回数がかかった.

3.1.3 Freeman Code による結果と考察

図 3.1.2 に対して Freeman Code の実行結果を表 3.1.2 に示す.

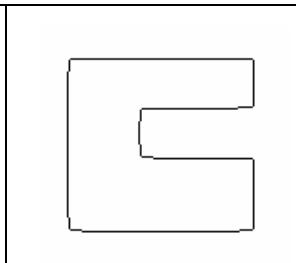
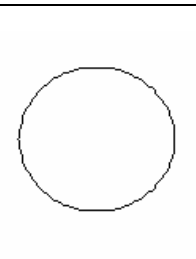
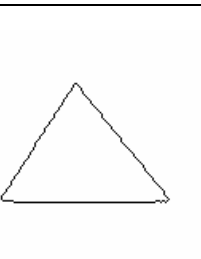
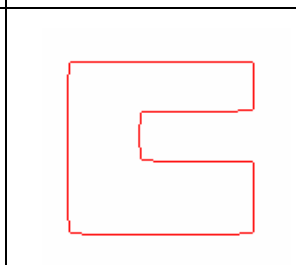
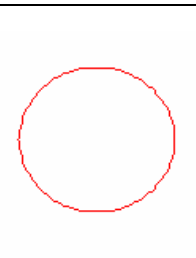
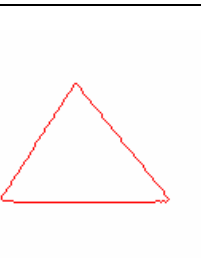
実行前			
実行後			

表 3.1.2 サンプル画像による Freeman Code の実行結果

Freeman Code では，画像中の画素全てに対して探索を行う為，動的輪郭モデルのように実行前に初期値を手動で与える必要がない．また，Freeman Code では計算回数，つまり輪郭線探索を行う画像の画素数に比例して実行時間が遅くなっていく．

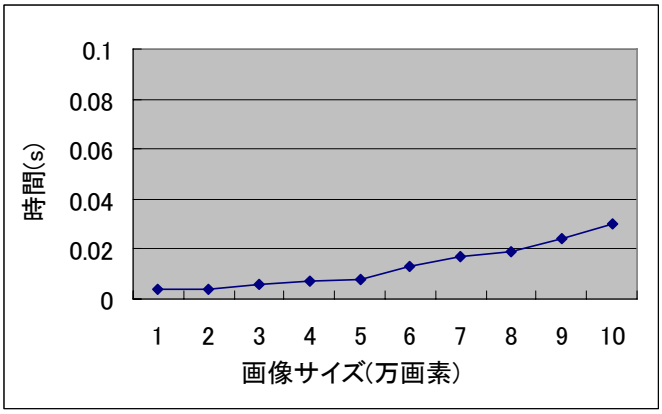


図 3.1.3　サンプル画像（polygon）における実行時間

サンプル画像に対して，動的輪郭モデル，Freeman Code 共に正確な面積測定領域を抽出することが出来た．サンプル画像（画総サイズ 150×150）に対して，２つの手法によって求められた領域内の画素数測定結果を表 3.1.3 に示す．

	Polygon	circle	triangle
動的輪郭モデル	9375	7004	4090
Freeman Code	9949	7838	4214

表 3.1.3 画素数測定結果　　（pixel）

3.2　実画像による実験

3.2.1　使用した実画像

実画像による実験として，Google Maps から２枚の画像を用いた．図 3.2.1 は対象の建物の周りに面積測定領域抽出には不要なものがあまり写っていない画像である．面積測定領域抽出を行う為に Canny 法によって Edge 強調を行った結果が図 3.2.2 である．もう 1 枚の画像を示している図 3.2.3 は，街中の建物の画像で，Edge 強調や測定領域抽出を行うときに，その精度の低下の原因となりうる，様々なものが写っている画像である．図 3.2.3 も図 3.2.1 と同様に Canny 法によって Edge 強調を行った結果が図 3.2.4 である．

図 3.2.1 実画像 1

図 3.2.2 実画像 1 （Canny 法使用後）

図 3.2.3　実画像 2

図 3.2.4　実画像 2 （Canny 法使用後）

3.2.2　実画像 1 における実行結果と考察

図 3.2.2 の画像に対して動的輪郭モデル，Freeman Code を実行した結果を表 3.2.1 に示す.

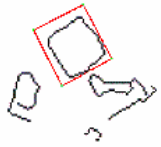
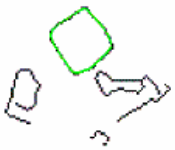
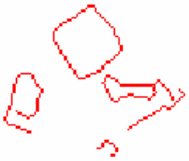
	動的輪郭モデル	Freeman Code
実行前		
実行後		

表 3.2.1　実画像 1 の実行結果

動的輪郭モデルでは，面積推定を行いたい対象物を指定して実行するため，実行後では上手く面積推定領域の抽出が行えている．一方，Freeman Code では，Canny 法によって強調された Edge すべてに対して輪郭線であると判断してしまうため，表 3.2.1 の様になっている.

3.2.3　実画像２における実行結果と考察

図 3.2.4 の画像に対して動的輪郭モデル，Freeman Code を実行した結果を表 3.2.2 に示す.

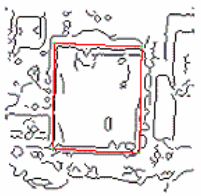
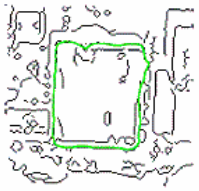
	動的輪郭モデル	Freeman Code
実行前		
実行後		

表 3.2.2　実画像２の実行結果

表 3.2.1 における動的輪郭モデルの実行後画像では，対象物の周りに不要なものがあまりないので，きちんと面積測定領域を抽出することが出来たが，実画像２では，対象物の周りに様々なものが写っているため，画素数測定領域は明確だが，対象物の領域とは誤差が生じた.

Freeman Code の結果では，実画像１と同様にすべての Edge に反応しているため，画素数測定領域が不明確である．画素数を測定するには閉じた領域である必要があるが，そのためには Freeman Code を使用する前の Canny 法での出力結果を改良する必要がある.

3.3　建物面積推定結果

2.3 節の方法によって求めた画素数から，実画像の尺度を元に建物面積の推定を行う．面積推定における精度の例として出雲ドームの画像（尺度:1/1500）を使用した．出雲ドームの画像に対する実行結果を次に示す．

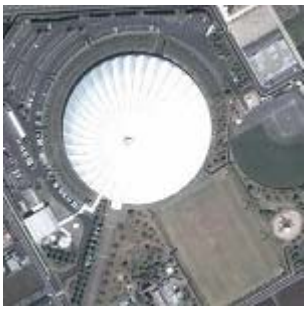


図 3.3.1　出雲ドーム

図 3.3.2　Canny 法使用後

図 3.3.3　動的輪郭モデル使用後

図 3.3.4　Freeman Code 使用後

本研究では，尺度 1/1500 の画像に対して 1pix≒2.5m^2 として面積推定を行った．

図 3.3.3 の動的輪郭モデル使用後における画素数測定領域の画素を測定したところ，6,191pix だったので，建物面積は 6,191×2.5≒15,477.5(m^2)と推測することができる．実際の出雲ドームの建物面積は約 16,277m^2 なので，約 95.1%の精度である．

図 3.3.4 の Freeman Code 使用後における画像では，3.2.3 節と同様に画素数測定領域が不明確であるため，建物面積の推定を行うことは出来なかった．

第４章　終論

今回の研究では，建物の面積測定領域を抽出するのに，動的輪郭モデルと Freeman Code の手法を用いて実験を行った．

動的輪郭モデルでは，様々な画像に対して，初期値として与える境界線や，曲率などの変数の与え方を変える場合があるが，高い精度での面積推定を行うことができた．

Freeman Code では，すべての Edge 抽出を正確に行えるが，抽出した Edge から面積推定を行う領域の抽出を行うことが困難なため，面積推定を行うことは出来なかった．

今後の課題としては，さまざまな画像に対して面積推定を可能にすることとして，動的輪郭モデルは面積推定領域の精度向上が挙げられる．Freeman Code は，動的輪郭モデルと同程度の面積測定領域の抽出を行えるようにする必要がある．また，どちらの手法も実行する前処理として Canny 法による Edge 強調を行ったが，Edge 強調結果によって面積推定領域の精度が変わるため，Canny 法を行うときの閾値の取り方も工夫が必要である．

謝辞

最後に，本研究を進めるにあたり，ゼミを中心に最後まで熱心なご指導，ご助言をして下さった田中章司郎教授に深く感謝の意を示すとともに，心より御礼申し上げます.

また，同じ研究室の木村眞吾さん，的場祥仁さん，清水洋志さん，岩脇正浩さんにも色々なご協力，ご助言を頂いたことに御礼申し上げます.

尚，本研究室で作成したプログラム，発表資料など全ての著作権を田中章司郎教授に譲渡いたします.

文献

[1] J.Canny: “A Computational Approach to Edge Detection”， IEEE Transactions on Pattern Analysis and Machine Intelligence，Vol.8，No.6，pp.679-697，Nov，1986

[2] M.Kass，A.Witkin，andD.Terzopoulos: “SNAKES: Active Contour Models，” Int.J.of Computer Vision，1，pp.321-331，1988.

[3] IMAGE ANALYSIS AND COMMUNICATIONS LAB
http://iacl.ece.jhu.edu/projects/gvf/

[4] H.Freeman，“On the encoding of arbitrary geometric configurations，”IRE Trans.，vol.EC-10，pp.260-268，June 1961.

付録

- 動的輪郭モデル（SnakeIter）ソースコード（MATLAB）

  SnakeIter から次に示す snakeinterp 関数と snakedeform 関数を呼び出している．

```
%%%%%%%%%%% Snake Iter %%%%%%%%%%

function SnakeIter(action,varargin)
if nargin<1,
   action='Initialize';
end;

feval(action,varargin{:});
return;

function Initialize()

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% snake deformation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[x,y] = snakeinterp(XSnake,YSnake,dmax,dmin);

set(HDmainf,'CurrentAxes',HDorigPic);
hold on

XS=[x; x(1)];
YS=[y; y(1)];
HD=line(XS,YS);
for i=1:ceil(NoSnakeIterations/5),
   if i<=floor(NoSnakeIterations/5)
      [x,y] = snakedeform(x,y,alpha,beta,gamma,kappa,px,py,5);
      title(['Iter = ', num2str(i*5)])
   else
      [x,y] = snakedeform(x,y,alpha,beta,gamma,kappa,px,py,NoSnakeIterations-floor(NoSnakeIterations/5)*5);
      title(['Iter = ', num2str(NoSnakeIterations)])
   end;
   [x,y] = snakeinterp(x,y,dmax,dmin);
   XS=[x; x(1)];
   YS=[y; y(1)];

   set(HD,'Color','Red','Marker','None');
   HD=line(XS,YS);
   if (SnakeDotsON==1) % draw dots if it is chosen
        set(HD,'Color','Red','Marker','.','MarkerEdgeColor','Green','MarkerFaceColor','Blue','MarkerSize',DotsSize);
        else
         set(HD,'Color','Red','Marker','None');
        end;

   pause(0.1);
end
XSnake=x; YSnake=y;
title(['Iter = ' num2str(NoSnakeIterations)])
pause(0.1);
hold off;

imdisp(-Image1); title('Original image');
```

・動的輪郭モデル（snakedeform）　ソースコード(MATLAB)

```
function [x,y] = snakedeform(x,y,alpha,beta,gamma,kappa,fx,fy,ITER)
% SNAKEDEFORM  外部力フィールドが与えるsnakeの変形.
%              それぞれのパラメータを用いてsnakeを変形.
%
%     alpha:   弾力 パラメータ
%     beta:    剛性 パラメータ
%     gamma:   粘性 パラメータ
%     kappa:   外部の重み
%     fx,fy:   外部力フィールド
%     ITER:    ループ回数

%      Chenyang Xu and Jerry L. Prince, 4/1/95, 6/17/97
%      Copyright (c) 1995-97 by Chenyang Xu and Jerry L. Prince
%      Image Analysis and Communications Lab, Johns Hopkins University

% generates the parameters for snake
N = length(x);

alpha = alpha* ones(1,N);
beta = beta*ones(1,N);

% produce the five diagonal vectors
alpham1 = [alpha(2:N) alpha(1)];
alphap1 = [alpha(N) alpha(1:N-1)];
betam1 = [beta(2:N) beta(1)];
betap1 = [beta(N) beta(1:N-1)];

a = betam1;
b = -alpha - 2*beta - 2*betam1;
c = alpha + alphap1 +betam1 + 4*beta + betap1;
d = -alphap1 - 2*beta - 2*betap1;
e = betap1;

% generate the parameters matrix
A = diag(a(1:N-2),-2) + diag(a(N-1:N),N-2);
A = A + diag(b(1:N-1),-1) + diag(b(N), N-1);
A = A + diag(c);
A = A + diag(d(1:N-1),1) + diag(d(N),-(N-1));
A = A + diag(e(1:N-2),2) + diag(e(N-1:N),-(N-2));

invAI = inv(A + gamma * diag(ones(1,N)));

for count = 1:ITER,
   vfx = interp2(fx,x,y,'*linear');
   vfy = interp2(fy,x,y,'*linear');

   % deform snake
   x = invAI * (gamma* x + kappa*vfx);
   y = invAI * (gamma* y + kappa*vfy);
end
```

・動的輪郭モデル（snakeinterp）　ソースコード(MATLAB)

```
function [xi,yi] = snakeinterp(x,y,dmax,dmin)
%SNAKEINTERP  snakeに順応して補間する関数
%   [xi,yi] = snakeinterp(x,y,dmax,dmin)
%
%   dmax: ２点の最大距離
%   dmin: ２点の最小距離
%   d(i,i+1)>dmax ならiとi+1の間に新しい点を取る
%   d(i,i+1)<dmin ならiかi+1を新しい点とする
%

%   Chenyang Xu and Jerry L. Prince, 4/1/95, 6/17/97
%   Copyright (c) 1995-97 by Chenyang Xu and Jerry L. Prince
%   Image Analysis and Communications Lab, Johns Hopkins University

% convert to column vector
x = x(:); y = y(:);

N = length(x);

d = abs(x([2:N 1])- x(:)) + abs(y([2:N 1])- y(:));

% remove the points which distance to neighbor points is shorter than dmin
IDX = (d<dmin);

idx = find(IDX==0);
x = x(idx);
y = y(idx);

N = length(x);
d = abs(x([2:N 1])- x(:)) + abs(y([2:N 1])- y(:));

IDX = (d>dmax);

z = snakeindex(IDX);

p = 1:N+1;

xi = interp1(p,[x;x(1)],z');
yi = interp1(p,[y;y(1)],z');

N = length(xi);
d = abs(xi([2:N 1])- xi(:)) + abs(yi([2:N 1])- yi(:));

while (max(d)>dmax),

    IDX = (d>dmax);
    z = snakeindex(IDX);

    p = 1:N+1;

    xi = interp1(p,[xi;xi(1)],z');
    yi = interp1(p,[yi;yi(1)],z');

    N = length(xi);
    d = abs(xi([2:N 1])- xi(:)) + abs(yi([2:N 1])- yi(:));
end
```

・Freeman Code　ソースコード(C 言語)

```
#include<stdio.h>
#include<string.h>
#include<time.h>

#define isx 10
#define isy 10
#define leng 100

int jp[isx][isy];
int list[2][leng];
FILE *fp;

int bdfl1(int jp[][isy], int jsx, int jsy, int leng1, int ix0, int iy0, int is)
{
        int i, j, r, s;
        int iyp[8]={0,-1,-1,-1,0,1,1,1};
        int ixp[8]={1,1,0,-1,-1,-1,0,1};
        int ix, iy, ixw, iyw, ixn, iyn;
        int is1, is8, isw, nn, in, in1, flag;

        ix = ix0;
        iy = iy0;
        is1 = is;
        isw = 0;
        nn = 1;
        flag = 0;

        jp[ix][iy] = 2;
        /*---1-LINE TRACKING----------------------*/
        /*---SEARCH OF NEXT POINT-----------------*/

        /*--CONTINUE 11---------------------------*/
        while( flag == 0 ){
                list[0][leng1] = ix;
                list[1][leng1] = iy;
                is1 += nn;
                is8 = is1 + 8;
                /*--DO 10-------------------------------*/
                for( in1 = is1; in1 < is8; in1++ ){
                        in = in1;
                        if( 8 <= in ) in -= 8;
                        ixw = ix + ixp[in];
                        iyw = iy + iyp[in];
                        /*---ISOLATED POINT JUDGMENT------------*/
                        if( ((ixw < 0) || (jsx <= ixw) || (iyw < 0) || (jsy <= iyw) ||
                                                (jp[ixw][iyw] == 0)) && (in1 == (is8-1)) ){
                                fprintf( fp, "(%d, %d) is ISOLATED POINT¥n", ixw, iyw );
                                return leng1;
                        }
                        if( ((ixw < 0) || (jsx <= ixw) || (iyw < 0) || (jsy <= iyw) ||
                                                (jp[ixw][iyw] == 0)) && (in1 != (is8-1)) ){
                                continue;
                        }

                        /*---LAST LINE JUDGMENT-----------------*/
                        if( (in1 == (is8-1)) && (jp[ixw][iyw] == 2) ){
                                fprintf( fp, "LAST LINE POINT¥n" );
                                return leng1;
                        }

                        if( jp[ixw][iyw] == 1 ) jp[ixw][iyw] = 2;
                        if( in1 != (is8-1) ) break;
                }
```

```
            /*--CONTINUE 20----------------------*/
            is1 = in + 4;
            if( isw != 1 ){
                    isw = 1;
                    ixn = ixw;
                    iyn = iyw;
            }

            /*--CONTINUE 60----------------------*/
            if( is1 > 8 ) is1 -= 8;
            if( leng <= leng1 ){
                    printf( "leng1 is size limit!¥n" );
                    fclose(fp);
                    return -1;
            }
            leng1++;

            if( leng1 > leng ){
                    fprintf( fp, "leng1 is size over!¥n" );
                    return -1;
            }

            ix = ixw;
            iy = iyw;
            if( (ix != ix0) || (iy != iy0) ) continue;
            /*-----------------------------------*/
            is1 += nn;
            is8 = is1 + 8;

            /*--DO 40----------------------------*/
            for( in1 = is1; in1 <= is8; in1++ ){
                    in = in1;
                    if( in > 8 ) in -= 8;
                    ixw = ix + ixp[in];
                    iyw = iy + iyp[in];
                    if( jp[ixw][iyw] == 0 ) continue;
                    break;
            }

            /*--CONTINUE 50----------------------*/
            is1 -= nn;
            if( (ixw != ixn) || (iyw != iyn) ){
                    continue;
            }else{
                    fprintf( fp, "閉曲線¥n" );
            }

            list[0][leng1] = ix;
            list[1][leng1] = iy;
            flag = 1;
        }
        return leng1;
}
```

```
int main(void)
{
        int i, j;
        int ix0, iy0, ix, iy, jsx, jsy, jp4;
        int leng0, leng1=0, nbd=0;
        int is, kerr;
        int stime, etime;
        int ip[isx][isy];

        fp = fopen( "log.txt", "w" );
        if( fp == NULL ){
                printf( "File Open error! >> log.txt¥n" );
                return -1;
        }

        /*測定開始*/
        stime = clock();

        jsx = isx;
        jsy = isy;


       /*---CLEAR LIST---------------------*/
       for( i = 0; i < leng; i++ ){
              list[0][i] = 0;
              list[1][i] = 0;
       }

       /*---SEARCHE FOR THE STARTING POINT OF THE LINE---*/
       /*---CONTINUE 11---------------------*/

       for( ix = 0; ix < isx; ix++ ){
              for( iy = 0; iy < isy; iy++ ){
                     if( jp[ix][iy] != 1 ) continue;
                     jp4 = jp[ix-1][iy]*jp[ix+1][iy]*jp[ix][iy-1]*jp[ix][iy+1];
                     if( jp4 == 0 )
                     {
                            /*---TRACK THE LINE----------------------------*/

                            /*---CONTINUE 20------------------*/
                            is = 1;
                            if( jp[ix-1][iy] == 0 ) is = 3;
                            if( jp[ix+1][iy] == 0 ) is = 7;
                            if( jp[ix][iy-1] == 0 ) is = 5;
                            ix0 = ix;
                            iy0 = iy;
                            leng1 = bdfl1(jp, jsx, jsy, leng1, ix0, iy0, is);
                            if( leng1 == -1 ){
                                   printf( "bdfl1 is error¥n" );
                                   fclose(fp);
                                   return -1;
                            }
                            leng1 += 2;
                            nbd++;

                            if( leng <= leng1 ){
                                   fprintf( fp, "leng1 is size over!¥n" );
                                   fclose(fp);
                                   return -1;
                            }
                     }else if( (jp4%2) == 0 ){
                            continue;
                     }
              }
       }
       leng1--;

       fclose(fp);
```

```
        /*測定終了*/
        etime = clock();
        printf( "Time : %d(ms)¥n", (etime-stime)/1000 );

        return 0;
]
```

・画素数測定プログラム　ソースコード(C 言語)

```
#include<stdio.h>
#define isx
#define isy

int main(void)
{
        int i, j, cnt=0, area=0;
        int jp[isx][isy];

        //画像の左上から右下へ、横方向の輪郭線カウント
        for( i = 0; i < isx; i++ ){
                for( j = 0; j < isy; j++ ){
                        if( jp[i][j] == 1 ){
                                cnt++;
                                j++;
                        }

                        //輪郭線に囲まれた画素に目印
                        if( cnt%2 == 1 ){
                                if( (jp[i][j-1] == 1)&&(jp[i][j] != 1) ){
                                        jp[i][j] += 2;
                                }
                                if( (jp[i][j-1] != 1)&&(jp[i][j] != 1) ){
                                        jp[i][j] += 2;
                                }
                        }
                }
                //横方向の輪郭線カウントリセット
                cnt = 0;
        }


        //除去作業
        for( i = 0; i < isx; i++ ){
                for( j = 1; j < isy; j++ ){
                        if( (jp[i][j]%2==0)&&(jp[i][j-1]==0) ){
                                jp[i][j] = 0;
                        }
                }
        }
        for( j = isy-1; 0 <= j; j-- ){
                for( i = isx; 0 <= i; i-- ){
                        if( (jp[i][j]%2==0)&&(jp[i][j+1]==0) ){
                                jp[i][j] = 0;
                        }
                }
        }

        //画像の左上から右下へ、縦方向の輪郭線カウント
        for( j = 0; j < isy; j++ ){
                for( i = 0; i < isx; i++ ){
                        if( jp[i][j] == 1 ){
                                cnt++;
                                i++;
                        }
                        //輪郭線に囲まれた画素に目印
                        if( cnt%2 == 1 ){
                                if( (jp[i-1][j] == 1)&&(jp[i][j] != 1) ){
                                        jp[i][j] += 2;
                                }
                                if( (jp[i-1][j] != 1)&&(jp[i][j] != 1) ){
                                        jp[i][j] += 2;
                                }
                        }
                }
                //縦方向の輪郭線カウントリセット
                cnt = 0;
        }
```

```
//除去作業
for( i = 0; i < isx; i++ ){
        for( j = 1; j < isy; j++ ){
                if( (jp[i][j]%2==0)&&(jp[i][j-1]==0) ){
                        jp[i][j] = 0;
                }
        }
}
for( j = isy-1; 0 <= j; j-- ){
        for( i = isx; 0 <= i; i-- ){
                if( (jp[i][j]%2==0)&&(jp[i][j+1]==0) ){
                        jp[i][j] = 0;
                }
        }
}

//画像の右下から左上へ、横方向の輪郭線カウント
for( i = isx; 0 <= i; i-- ){
        for( j = isy; 0<= j; j-- ){
                if( jp[i][j] == 1 ){
                        cnt++;
                        j--;
                }
                //輪郭線に囲まれた画素に目印
                if( cnt%2 == 1 ){
                        if( (jp[i][j+1] == 1)&&(jp[i][j] != 1) ){
                                jp[i][j] += 2;
                        }
                        if( (jp[i][j+1] != 1)&&(jp[i][j] != 1) ){
                                jp[i][j] += 2;
                        }
                }
        }
        cnt = 0;
}


//除去作業
for( i = 0; i < isx; i++ ){
        for( j = 1; j < isy; j++ ){
                if( (jp[i][j]%2==0)&&(jp[i][j-1]==0) ){
                        jp[i][j] = 0;
                }
        }
}
for( j = isy-1; 0 <= j; j-- ){
        for( i = isx; 0 <= i; i-- ){
                if( (jp[i][j]%2==0)&&(jp[i][j+1]==0) ){
                        jp[i][j] = 0;
                }
        }
}

//画像の右下から左上へ、縦方向の輪郭線カウント
for( j = isy; 0 <= j; j-- ){
        for( i = isx; 0 <= i; i-- ){
                if( jp[i][j] == 1 ){
                        cnt++;
                        i--;
                }
                //輪郭線に囲まれた画素に目印
                if( cnt%2 == 1 ){
                        if( (jp[i+1][j] == 1)&&(jp[i][j] != 1) ){
                                jp[i][j] += 2;
                        }
                        if( (jp[i+1][j] != 1)&&(jp[i][j] != 1) ){
                                jp[i][j] += 2;
                        }
                }
        }
        cnt = 0;
}
```

```
//除去作業
for( i = 0; i < isx; i++ ){
        for( j = 1; j < isy; j++ ){
                if( (jp[i][j]%2==0)&&(jp[i][j-1]==0) ){
                        jp[i][j] = 0;
                }
        }
}
for( j = isy-1; 0 <= j; j-- ){
        for( i = isx; 0 <= i; i-- ){
                if( (jp[i][j]%2==0)&&(jp[i][j+1]==0) ){
                        jp[i][j] = 0;
                }
        }
}
```

```
        //目印が重なっている画素数カウント
        for( i = 0; i < isx; i++ ){
                for( j = 0; j < isy; j++ ){
                        if( 4 <= jp[i][j]) area++;
                }
        }

        printf( "%d¥n", area );

        return 0;

}
```