**图像处理实验报告**

**姓名：吴阳诚**

**学号：210810507**

**院系：理学院**

**专业：数据科学与大数据技术**

**班级：5 班**

目录

[实验一、CV模型实验 4](#_Toc167982966)

[目的和意义 4](#_Toc167982967)

[模型介绍与说明 4](#_Toc167982968)

[模型的数学原理 4](#_Toc167982969)

[实验数据说明 6](#_Toc167982970)

[程序设计流程 7](#_Toc167982971)

[实验结果 8](#_Toc167982972)

[Size4: 29\*40 9](#_Toc167982973)

[Size4 29\*40 10](#_Toc167982974)

[Delta\_t: 0.1 10](#_Toc167982975)

[Delta\_t: 0.01 10](#_Toc167982976)

[Mar:1 10](#_Toc167982977)

[Mar:5 10](#_Toc167982978)

[Mar: 10 10](#_Toc167982979)

[Size3: 59\*80 11](#_Toc167982980)

[Size3 59\*80 12](#_Toc167982981)

[Delta\_t: 0.1 12](#_Toc167982982)

[Delta\_t: 0.01 12](#_Toc167982983)

[Mar:1 12](#_Toc167982984)

[Mar:5 12](#_Toc167982985)

[Mar: 10 12](#_Toc167982986)

[Mar: 20 12](#_Toc167982987)

[Size2: 118\*160 13](#_Toc167982988)

[Size2: 118\*160 14](#_Toc167982989)

[Delta\_t: 0.1 14](#_Toc167982990)

[Delta\_t: 0.01 14](#_Toc167982991)

[Mar:1 14](#_Toc167982992)

[Mar:10 14](#_Toc167982993)

[Mar: 20 14](#_Toc167982994)

[Mar: 30 14](#_Toc167982995)

[Mar: 40 15](#_Toc167982996)

[Size1: 96\*534 15](#_Toc167982997)

[Size1: 96\*534 16](#_Toc167982998)

[Delta\_t: 0.1 16](#_Toc167982999)

[Delta\_t: 0.01 16](#_Toc167983000)

[Mar:1 16](#_Toc167983001)

[Mar:10 17](#_Toc167983002)

[Mar: 30 17](#_Toc167983003)

[Mar: 50 18](#_Toc167983004)

[Mar: 100 18](#_Toc167983005)

[结论与讨论 19](#_Toc167983006)

[实验二、RSF模型实验 20](#_Toc167983007)

[目的和意义 20](#_Toc167983008)

[模型介绍与说明 20](#_Toc167983009)

[模型的数学原理 20](#_Toc167983010)

[程序设计流程 21](#_Toc167983011)

[实验结果 21](#_Toc167983012)

[结论与讨论 22](#_Toc167983013)

[附录 代码 22](#_Toc167983014)

[实验一 CV C++ opencv 22](#_Toc167983015)

["MatAt.hpp" 23](#_Toc167983016)

["progressbar.hpp" 24](#_Toc167983017)

[class MyCV 24](#_Toc167983018)

[// initPhi() 26](#_Toc167983019)

[// updataC1() 27](#_Toc167983020)

[// updataC2() 28](#_Toc167983021)

[// Heaviside() 28](#_Toc167983022)

[// NormalizedGradient() 29](#_Toc167983023)

[// divNormalizedPhi() 29](#_Toc167983024)

[// updatePhi() 30](#_Toc167983025)

[// fit() 31](#_Toc167983026)

[// LastPhiAdjust() 32](#_Toc167983027)

[// LastPhiOutput() 33](#_Toc167983028)

[testMyCVTotal() 34](#_Toc167983029)

[Main() 36](#_Toc167983030)

[实验二 RSF 36](#_Toc167983031)

# 实验一、CV模型实验

## 目的和意义

本实验采用C++语言，通过基础代码完成对CV模型的复现。

通过对CV模型的复现，可以体会能量泛函在图像分割上的应用，同时理解能量泛函的求解算法。

本实验探究了初始Phi的margin设置，以及训练过程的Max\_Iter, delta\_t设置对于CV模型的影响，并得到一定的基础结论。

## 模型介绍与说明

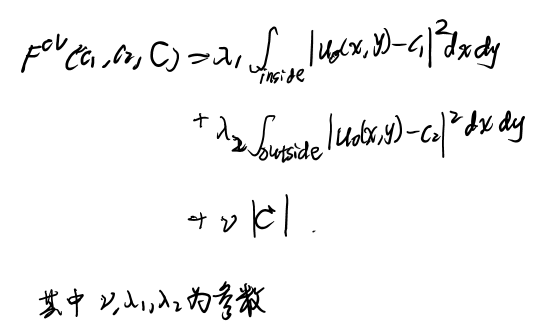
CV 模型是一个基于区域的活动轮廓模型，相比其他活动轮廓模型，CV 模型具有如下几个优点。

CV 模型是基于 Mumford-Shah 分割技巧和水平集方法，而不是基于边界函数使得演变曲线停在想要的边缘上。而且即使初始图像含有噪声，也不需要光滑初始图像，边缘的位置仍可以很好地被检测和保留。CV 模型可以检测不是由梯度定义的边缘或者是非常光滑的边缘，而对于这些边缘，经典的活动轮廓模型是不适用的。最后 CV 模型可以仅从一条初始曲线自动地检测内部轮廓，而且初始曲线的位置不必环绕待检测的物体，可以在图像的任意位置。这是CV 模型一个非常重要的优点。

但是 CV 模型主要适用于具有同质区域的图像，对于不均匀强度背景的图像则效果较差。在 CV 模型里，常量 c1 和 c2 用来近似区域 inside(C) 和 outside(C) 内的图像强度，很显然，这只是一种粗糙的近似，而不包含图像的局部强度信息，因此CV 模型不适用于图像强度不均匀的场景。

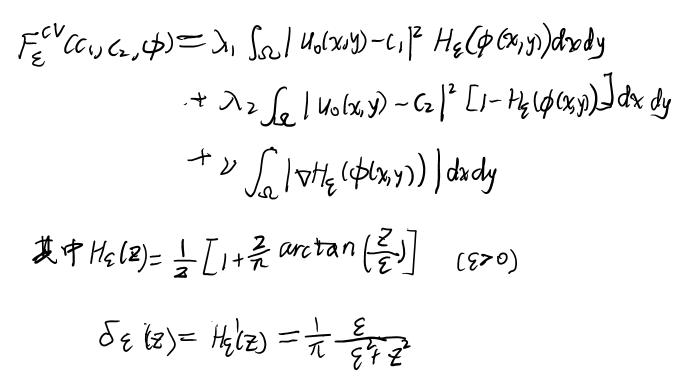
## 模型的数学原理

CV的能量泛函模型如下：

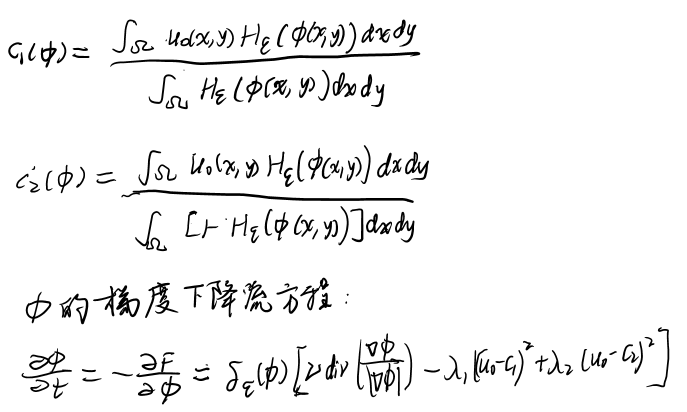


前两项为数据拟合项，最后一项为长度项。

引入水平集函数表示边界C ()，引入Heaviside函数作为边界内外的示性函数。这样，曲线内外就能进行统一表达。另外，进行正规化近似表示，得到正规化的能量函数表达：



之后采用标准的梯度下降法，采用交替极小化格式，求解：



## 实验数据说明

本实验采用的图片是自选的人物图片（单通道）。共有4张，分别是同样的内容放缩得到的不同尺寸（像素）图片。尺寸分别为：（从大到小）396\*534, 118\*160; 59\*80; 29\*40。

## 程序设计流程

(程序语言为C++)依照数学原理进行，进行离散化处理，得到程序。代码见附录。

对于单一图片：

设置myCV的class，成员包含了Max\_Iter（最大迭代次数）, delta\_t（每次Phi更新的步长）， Ph, C1, C2i等 其，并中包含了对单一图片进行CV泛函计算的全部方法。

图像读取为Img（单通道），格式为CV\_8U。

Phi初始化为与Img大小一致的Mat，格式为CV\_32F，且设置margin参数，表示初始的Phi中矩形边界与图像边界的距离。于是初始矩形边界外部为-1，内部为+1，矩形本身为0.

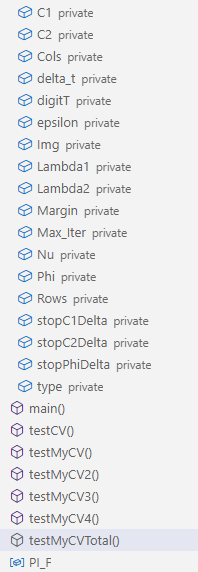
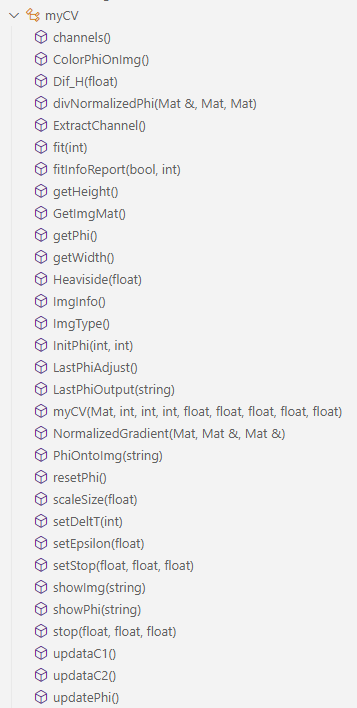
对于单个图像，使用fit()函数可以完成对Phi，C1,C2的极小化。具体方法就是CV能量泛函的极小化过程的代码实现。其中与Phi计算相关的 div（散度矩阵）等均采用CV\_32F格式，用于精确计算。

Ph，C1, C2i的迭代过程通过Max\_Iter控制终止。实际上,也尝试设置基于delta变化的终止条件，但是效果不好。

而最终的Phi则进行“示性处理”：负数为outside部分，取为0，正数为inside部分，取为255，再把Phi转化为CV\_8U格式，即inside为白色，outside为黑色。

保存最终Phi的结果到图片对应的输出文件夹中。

批量图片处理与参数网格搜索：

本实验处理了同一内容不同大小的一系列图片，并且希望考察初始Phi的margin设置，以及训练过程的Max\_Iter, delta\_t设置对于CV模型的影响。所以设计了testMyCVTotal()函数，其中使用vector及迭代器等，完成了批量的图像处理，以及对于相依参数的gridSearch（网格搜索）（很粗糙的测试）。其中Max\_Iter固定为10000.

## 实验结果

Mar表示初始的Phi中矩形边界与图像边界的距离，delta\_t表示每次Phi更新的步长。所有结果都是Max\_Iter=10000次循环得到。

结果图片为Phi“示性化”处理的展示，内部为白色，外部为黑色。

本实验结果强调不同Margin设置以及不同delta\_t设置对于CV分割效果的影响。

### Size4: 29\*40

Orginal:



|  |  |  |
| --- | --- | --- |
| Size4 29\*40 | Delta\_t: 0.1 | Delta\_t: 0.01 |
| Mar:1 |  |  |
| Mar:5 |  |  |
| Mar: 10 |  |  |

### Size3: 59\*80

Original:



|  |  |  |
| --- | --- | --- |
| Size3 59\*80 | Delta\_t: 0.1 | Delta\_t: 0.01 |
| Mar:1 |  |  |
| Mar:5 |  |  |
| Mar: 10 |  |  |
| Mar: 20 |  |  |

### Size2: 118\*160

Original:



|  |  |  |
| --- | --- | --- |
| Size2: 118\*160 | Delta\_t: 0.1 | Delta\_t: 0.01 |
| Mar:1 |  |  |
| Mar:10 |  |  |
| Mar: 20 |  |  |
| Mar: 30 |  |  |
| Mar: 40 |  |  |

### Size1: 96\*534

Original:



|  |  |  |
| --- | --- | --- |
| Size1: 96\*534 | Delta\_t: 0.1 | Delta\_t: 0.01 |
| Mar:1 |  |  |
| Mar:10 |  |  |
| Mar: 30 |  |  |
| Mar: 50 |  |  |
| Mar: 100 |  |  |

## 结论与讨论

1. 可以看到，对于图像强度相对均匀的图片，CV模型也有较好的分割效果。本次实验自选的一系列图片，其实强度并不是很均匀，不过CV的效果依然不错。

2. CV模型初始Phi的margin（矩形Phi的边缘到图像的边缘的距离）的设置对于图像的分割效果有一定影响。CV模型中margin很小时（如1）就能自动完成内部轮廓的搜索，这与课本知识一致。同时，本实验中可以注意到，它似乎更容易搜素基于Phi矩形内部信息的轮廓，初始Phi的矩形边界外部的轮廓似乎不容易体现。

如此看来，初始化Phi可以直接取很小的margin，从而自动完成内部所有轮廓的搜索。不过，如果想强调内部对象的轮廓，就可以取较大的margin，从而识别出更单纯的内部对象的轮廓。（参见Size1结果中Mar的设置带来的结果变化）

3. 本实验在CV模型时，尝试设置基于每次变化的stop条件，也就是如果变化很小，就直接终止。但是这样的stop条件似乎很难设置，应为Phi的更新其实每次都是很小的，如果想直接设置stop条件，很容易很快地提前终止，而完全没有达到分割效果。所以，之后直接设置为Max\_Iter。

4. 本实验设置Max\_Iter为10000次，而Phi梯度流中的delta\_t（每次Phi变化的步长）设置要与之匹配，才能确保Phi充分演变完毕。从实验结果的对比很容易看到，10000次对应的delta\_t取为0.1较合适，0.01就过小了，Phi离演变完全还很远。也就是说，如果取delta\_t为0.01， 需要设置更大Max\_Iter。

对于一般化的其他图片，Max\_Iter和delta\_t的合理设置还有待进一步探究。

# 实验二、RSF模型实验

## 目的和意义

本实验采用C++语言，对经典的RSF模型通过基础代码进行复现。本实验能够加深对于RSF这一基于局部信息的图像分割模型的理解，体会该算法与CV模型的差异。

## 模型介绍与说明

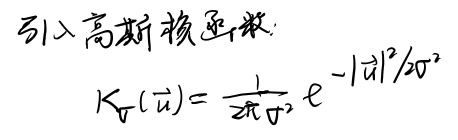
不同于 CV 模型用两个常量 c1 和 c2 来近似轮廓线 C 两侧的区域 inside(C) 和 outside(C)内的图像强度。RSF 模型用两个拟合函数和来拟合C 两侧的区域内的图像强度。注意到 和 的值是随着中心点 x 的变化而变化的。和 的这种空间变化性质使得 RSF 模型从本质上区别于 CV 模型，这种空间变化性质来源于空间变化的核函数 的局部化性质。RSF模型的区域可伸缩性也来源于核函数 的尺度参数 σ， 可以控制局部区域的大小，从小的邻域到整个定义域，这样就可以在一个可控制尺度的区域内充分利用图像的强度信息用于引导活动轮廓线的移动。

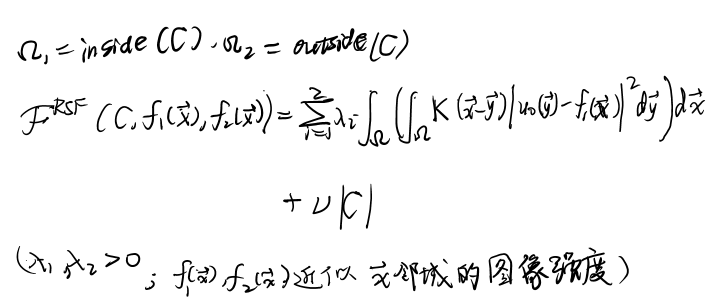
RSF 模型通过使用核函数充分利用图像的局部强度信息，因此该模型可以分割具有图像强度不均匀性质的图像，而且对于一些具有弱边界的物体如血管等的分割有很好的效果。

但是 RSF 模型仅仅利用图像的局部信息可能会导致能量泛函的局部极小，因此 RSF 模型的分割结果会更加依赖于轮廓线的初始化。此外，因为 RSF 模型是非凸的，这也是导致局部极小解存在的一个原因。

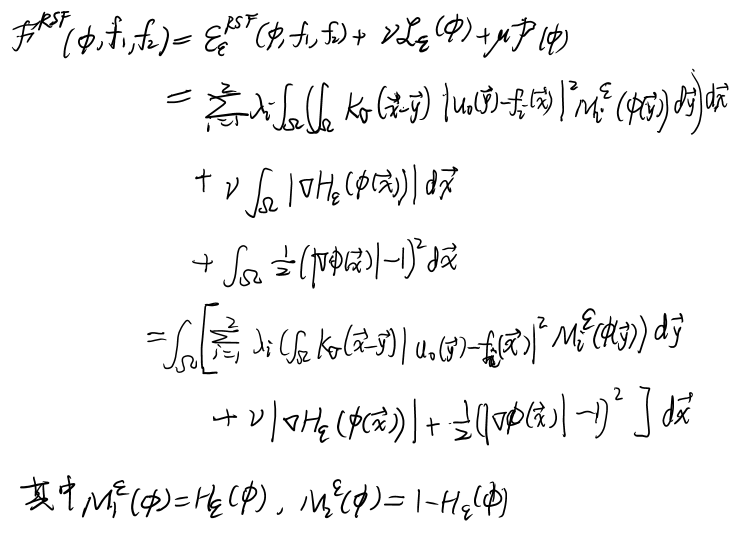
## 模型的数学原理

RSF的能量泛函如下：

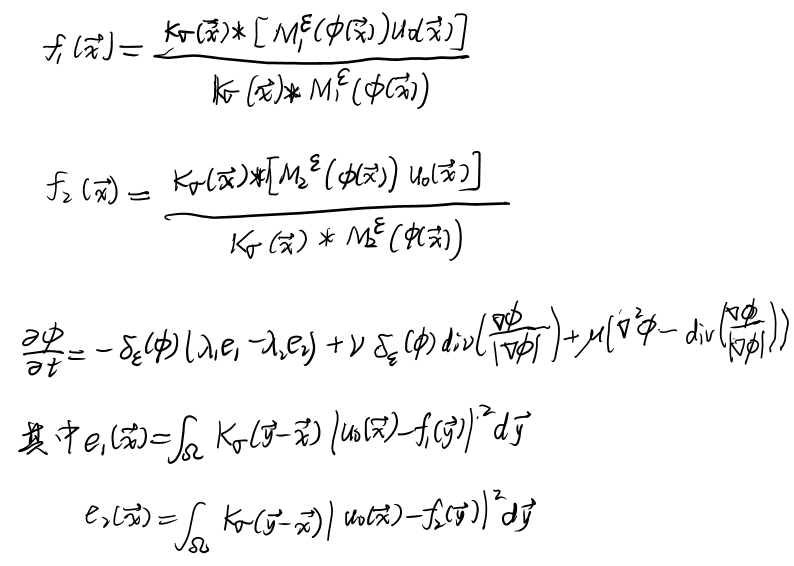




对该泛函模型，类似地，引入水平集函数进行统一表示，同时添加一项水平集正则项（用来保持水平集函数的正规性，保证计算精确性，同时避免重新初始化水平集函数的过程），最后进行光滑化，得到如下光滑的RSF水平集格式的能量泛函：



为了极小化该能量泛函，采用标准的梯度下降法，交替极小化格式，求解如下：



## 程序设计流程

依照数学过程，离散化处理，转化为程序。代码见附录。

## 实验结果

## 结论与讨论

可以注意到，对于图像强度不均匀的场景，RSF的分割效果较CV有较大的提升。不过，RSF也暴露出对初始形状相当敏感的问题，调节初始参数是一件困难的事情。

# 附录 代码

## 实验一 CV C++ opencv

// to reproduct CV algorithm

// the original picture may be CV\_8U, what if I do not change it, directly use a CV\_32F Phi on it? the calculation of Phi should can "include" the CV\_8U type.

// PHi needs negative pixel, but there is no need to change the original picture!

// so we can try just same size between Img and Phi, but Phi with different type CV\_32F for calculation, and when it comes to show Phi on Img, then make some convertion.

// so Img CV\_8U, while PHI and NGX, NGY, div :  CV\_32F (for calcuation)

// maybe use CV\_64F (double) for higher accuracy?

// let us try the experiment picture

// stoppage is not suitable! each time just tiny moves, so only use max\_iter!!

// adjust last Phi,to distinguish negative(as 0, outside) and positive(255, inside)

// max\_iter may be 1e4, 1e5; 1e5 too large and slower, 1e4 is preferred

// for 1e4 max\_iter, delt\_t can not be too tiny like 0.01, 0.1 is suitable

// use vector for grid search of parameters!

// make testMyCVTotal() to systematically handle those pictures and use grid search!

// for different pictures, may use Threads !! to parallelly handle them! to try...

// from the result, it seeems that Phi is easier to check edges inside the original Phi?

#include <filesystem>

#include <string>

#include <vector>

#include <math.h>

#include <cmath>

#include <iostream>

#include <opencv2\core\core.hpp>

#include <opencv2\highgui\highgui.hpp>

#include <opencv2\opencv.hpp>

#include "MatAt.hpp" // my hpp to uniformly get Mat\_at

#include "progressbar.hpp" // to show progress

using namespace std;

using namespace cv;

namespace fs = std::filesystem; // rename

const float  PI\_F=3.14159265358979f;

### "MatAt.hpp"

// to give an automatica mat\_at() without typename

// FMat\_at return float of data !

//

#ifndef MATAT\_HPP\_

#define MATAT\_HPP\_

#include <opencv2/core.hpp>

#include <iostream>

using namespace std;

// Function for accessing pixel values of CV\_8U Mat

float FMat\_at(cv::Mat& img, int row, int col) {

    if(img.type()==0){ // 8U

        return img.at<uchar>(row, col);

    }else if(img.type()==5){ // 32F

        return img.at<float>(row, col);

    }else{ // many other situation

        cout << " type error" << endl;

        exit(0);

    }

}

#endif

### "progressbar.hpp"

#ifndef \_PROGRESSBAR\_HPP

#define \_PROGRESSBAR\_HPP

#include <iostream>

using namespace std;

void ProgressShow(int loopNum, int i, int SectionNum=10){ // inside loop, get index i(0~loopNum-1)

            int section = loopNum/SectionNum;

            // at start

            if(i == 0){

                cout <<  "progress: 0%" ;

                return;

            }

            if( (i+1)%section == 0){

                cout << "\r";  // return to the head

                cout << "progress: " << (i+1)/section\*100 /SectionNum<<"%"; // those int!! should \*100 first, or get 0!

            }

}

#endif

### class MyCV

class myCV

{

    private:

        Mat Img;

        int Rows; // height of Img

        int Cols; // width of Img

        int type; // type of Img, 0:CV\_8U, 5:CV\_32F,   should make a type dict!

        int digitT = 2;

        float delta\_t = pow(0.1, digitT); // for gradient decline of Phi

        float epsilon = 1e-6;

        // float sigma = 25.0;

        float C1;

        float C2;

        float Lambda1;

        float Lambda2;

        float Nu;

        Mat Phi;   // 2D

        int Margin;

        int Max\_Iter;

        // stop delta

        float stopC1Delta=1e-6;

        float stopC2Delta=1e-6;

        float stopPhiDelta = 1e-6;

    public:

        myCV(Mat img = Mat(), int margin =10, int max\_iter=100, int thickness = 1, float c1=1, float c2=1, float lambda1=1, float lambda2=1, float nu=1){

            // c1, c2, Phi to update

            Img = img;

            type = Img.type();

            Rows = Img.rows;

            Cols = Img.cols;

            if(Img.channels()>1){

                ExtractChannel();

                cout << "multichannel, have already extracted one channel picture." << endl;

                exit(0);

            }

            // no change on Img, just set Phi!

            C1 = c1; // inside  first 0?

            C2 = c2; // outside first 255?

            Lambda1 = lambda1;

            Lambda2 = lambda2;

            Nu = nu;

            Margin = margin;

            InitPhi(Margin, thickness); // Init Phi

            Max\_Iter = max\_iter;

        }

#### // initPhi()

        void InitPhi(int margin, int thickness){ // yes!  but actually the 0 in middle is not neccessary

            Phi = cv::Mat::ones(Img.size(), CV\_32F);  // set Phi CV\_32F for calculation

            Phi = -1\*Phi;  // outside edge as -1, just white

            // first Rect, get 0 inside

            cv::Rect PhiInner1(margin,margin,Phi.size().width-2\*margin, Phi.size().height-2\*margin); // choose inner ROI

            Phi(PhiInner1) = 0\*Phi(PhiInner1);  // edge as 0

            // second Rect, get -1 inside the edge, thickness of the edge is 1

            int marginInner = margin + thickness; // margin to the outer

            cv::Rect PhiInner2(marginInner, marginInner, Phi.size().width-2\*marginInner, Phi.size().height-2\*marginInner); // choose inner ROI

            Phi(PhiInner2).convertTo(Phi(PhiInner2), -1, 1.0, 1); // add 1, inner as +1

        }

        void ImgInfo(){  // cout the information

            ImgType();

            channels();

            cout << GetImgMat().size() <<  endl;

        }

        void ImgType(){

            cout << "type: "<< Img.type() << endl;

        }

        void channels(){

            cout << "channels:  " << Img.channels() << endl;;

        }

        void ExtractChannel(){ // get 1 channel

            std::vector<cv::Mat> channels;  // vector of each channel

            cv::split(Img, channels);

            cv::Mat oneChannel = channels[0];

            oneChannel.convertTo(oneChannel, CV\_32F); // convert to 32-bit floating point

            Img = oneChannel;

            cv::imshow("OneChannel",Img);

            cv::waitKey(0);

            cv::imwrite("onechannel.png", Img, {cv::IMWRITE\_PNG\_COMPRESSION, 0});

        }

        void setEpsilon(float x=0.001){

            epsilon = x;

        }

#### // updataC1()

        float updataC1(){ // return the change

            float numerator=0;

            float denominator=0;

            for (int i = 0; i < Img.rows; i++) {

                for (int j = 0; j < Img.cols; j++) {

                    // get local point, use the short name instead of at<>..

                    float phi\_local = Phi.at<float>(i,j);

                    float u\_local;

                    u\_local = FMat\_at(Img, i,j);

                    numerator += u\_local\*Heaviside(phi\_local);

                    denominator += Heaviside(phi\_local);

                }

            }

            float delta = abs(C1-numerator/denominator) ;

            C1 = numerator/denominator ;

            return delta;

        }

#### // updataC2()

        float updataC2(){ // return change

            float numerator=0;

            float denominator=0;

            for (int i = 0; i < Img.rows; i++) {

                for (int j = 0; j < Img.cols; j++) {

                    // get local point, use the short name instead of at<>..

                    float phi\_local = Phi.at<float>(i,j);

                    // float u\_local = Img.at<uchar>(i,j);  // uchar for CV\_8U

                    float u\_local = FMat\_at(Img, i,j);

                    numerator += u\_local\*(1-Heaviside(phi\_local));

                    denominator += (1-Heaviside(phi\_local));

                }

            }

            float delta = abs(C2-numerator/denominator) ;

            C2 = numerator/denominator ;

            return delta;

        }

#### // Heaviside()

        float Heaviside(float x){ // approximately distinguish: x>=0 get 1 and x<0 get 0

            return ( 1+(2/PI\_F)\*(atan(x/epsilon)) )/2.0;

        }

        void resetPhi(){ // to reset the phi into -1 or 1 ??

            for (int i = 0; i < Img.rows; i++) {

                for (int j = 0; j < Img.cols; j++) {

                    float phi\_local = Phi.at<float>(i,j);

                    Phi.at<float>(i,j) = (2/PI\_F)\*(atan(phi\_local/epsilon))  ; // into -1 or 1

                }

            }

        }

        float Dif\_H(float x){ // no use?

            return (1/PI\_F)\*( epsilon/( x\*x+ epsilon\*epsilon) ) ;

        }

#### // NormalizedGradient()

        void NormalizedGradient(Mat Phi, Mat& Gx, Mat& Gy){ // Gx, Gy to store gradient in x/y order

            // original Phi, only +1, -1

            // Calculate the gradient in the x and y directions

            cv::Sobel(Phi, Gx, CV\_32F, 1, 0); //X order

            cv::Sobel(Phi, Gy, CV\_32F, 0, 1); // Y order

            // Access the gradient values

            for (int i = 0; i < Img.rows; i++) {

                for (int j = 0; j < Img.cols; j++) {

                    float gx = Gx.at<float>(i, j);

                    float gy = Gy.at<float>(i, j);

                    float len = sqrt(gx\*gx + gy\*gy);  // len of the gradient

                    if(len < 1e-6){ // gx=gy=0

                        // len = 1 ; // for 0 in denominator! the gx,gy get 0 naturally

                        Gx.at<float>(i, j) = 0;

                        Gy.at<float>(i, j) = 0; // set 0

                    }else{

                        Gx.at<float>(i, j) = gx/len;

                        Gy.at<float>(i, j) = gy/len; // normalized

                    }

                }

            }

        }

#### // divNormalizedPhi()

        void divNormalizedPhi(Mat& div, Mat NGx, Mat NGy){ // normalized Gx,Gy

            // get div

            div = cv::Mat::zeros(Img.size(),CV\_32F); // initialize! // 32F for float TYPE

            cv::Mat gradientX, gradientY; // gx, gy in Mat

            cv::Sobel(NGx, gradientX, CV\_32F, 1, 0); //X order for Phi\_x

            cv::Sobel(NGy, gradientY, CV\_32F, 0, 1); // Y order for Phi\_y

            // Access the gradient values

            for (int i = 0; i < Img.rows; i++) {

                for (int j = 0; j < Img.cols; j++) {

                    float gx = gradientX.at<float>(i, j);

                    float gy = gradientY.at<float>(i, j);

                    div.at<float>(i,j) = gx+gy;  // get div

                }

            }

        }

        void setDeltT(int digit\_T=2){ // set delta t for gradient decline

            digitT = digit\_T;

            delta\_t = pow(0.1, digitT);

        }

#### // updatePhi()

        float updatePhi(){

            Mat NGX, NGY;

            NormalizedGradient(Phi, NGX, NGY);

            Mat div;

            divNormalizedPhi(div, NGX, NGY);  // get div

            float delta=0;

            for (int i = 0; i < Img.rows; i++) {

                for (int j = 0; j < Img.cols; j++) {

                    // get local point, use the short name instead of at<>..

                    float phi\_local = Phi.at<float>(i,j);

                    float div\_local =  div.at<float>(i,j);  // div local!

                    float u\_local = FMat\_at(Img, i,j);

                    // updat Phi

                    float delta\_local =  Dif\_H(phi\_local)\*( Nu\*div\_local - Lambda1\*pow(u\_local-C1 ,2) + Lambda2\*pow(u\_local-C2, 2) );

                    Phi.at<float>(i,j) += delta\_t\* delta\_local;

                    delta += delta\_local;

                }

            }

            return delta;

        }

        void setStop(float stopc1=1e-6, float stopc2=1e-6, float stopPhi=1e-6){

            stopC1Delta = stopc1;

            stopC2Delta = stopc2;

            stopPhiDelta = stopPhi;

        }

        bool stop(float deltaC1, float deltaC2, float deltaPhi){

            if( deltaC1 < stopC1Delta && deltaC2 < stopC2Delta && deltaPhi < stopPhiDelta ){ // may more strict?

                return true;

            }else{

                return false;

            }

        }

#### // fit()

        void fit( int max\_iter = 100 ){

            bool finishFlag = false;

            Max\_Iter = max\_iter;

            int maxtimes;

            for(int i=0; i<Max\_Iter; i++){

                ProgressShow(Max\_Iter, i);

                float deltaC1 = updataC1();

                float deltaC2 = updataC2();

                float deltaPhi = updatePhi();

                // ban stoppage  // it seems that, each time just small changes, so stop condition is not suitable!

                // if(stop( deltaC1, deltaC2, deltaPhi) && false) {

                //     finishFlag = true;

                //     maxtimes = i+1;

                //     cout << endl;

                //     cout << "finish at index "<< i+1 << endl;

                //     break;

                // }

            }

            LastPhiAdjust();

            fitInfoReport(finishFlag, maxtimes);

        }

#### // LastPhiAdjust()

        void LastPhiAdjust(){  // make Phi easier to show the edge, change from 32F into 8U?

            // edge between negative and positive

            // directly, negative(outside) into 0, positive(inside) into 255

            for(int i=0; i<Phi.rows; i++){

                for(int j=0; j<Phi.cols; j++){

                    float phi\_local = Phi.at<float>(i,j);

                    if(phi\_local >=0){

                        Phi.at<float>(i,j) = 255; // white

                    }else{

                        Phi.at<float>(i,j) = 0;  // black

                    }

                }

            }

            Phi.convertTo(Phi, CV\_8U); // no value change, just change type

        }

        void fitInfoReport(bool finishFlag, int maxtimes){

            // CV reports

            cout << endl;

            cout<< "Img size: "<< Img.size() << endl;

            cout << "Phi original margin: " << Margin << endl;

            cout << "Delta\_t: " << delta\_t << endl;

            // cout << cv::format("stop condition: stopC1Delta: %f, stopC2Delta: %f, stopPhiDelta: %f ", stopC1Delta, stopC2Delta, stopPhiDelta)  << endl;

            // cout << "Max\_Iteration: " << Max\_Iter << endl;

            cout << "C1: " << C1 << " C2: " << C2 << endl;

            if(finishFlag){

                cout << "finish up to stop condition, at times: "<< maxtimes << endl;

            }else{

                cout << "finish up to max\_iter: " << Max\_Iter << endl;

            }

        }

#### // LastPhiOutput()

        void LastPhiOutput(string pathbase){ // save Phi in the designated folder

            // save Phi

            cv::imwrite( pathbase + "/" + cv::format("Phi\_Mar%d\_C1\_%d\_C2\_%d\_Iter%d\_deltT\_1e-%d.png", Margin ,static\_cast<int>(C1), static\_cast<int>(C2),Max\_Iter, digitT ), Phi);

        }

        Mat getPhi(){

            return Phi;

        }

        void showPhi(string windowName = "Phi"){  // can set the windowname

            cv::namedWindow(windowName, cv::WINDOW\_NORMAL); // then drag for changing size

            imshow(windowName, Phi);

            cv::waitKey(0);

        }

        void showImg(string windowName = "Img"){

            cv::namedWindow(windowName, cv::WINDOW\_NORMAL);

            imshow(windowName, Img);

            cv::waitKey(0);

        }

        Mat GetImgMat(){

            return Img;

        }

        void scaleSize(float scale){ // should operate before Phi operation

            int h = Img.cols;

            int w = Img.rows;

            // resize(Img, Img, Size(int(w\*scale), int(h\*scale)),0,0,INTER\_LINEAR);

            resize(Img, Img, cv::Size(Img.size().width\*scale, Img.size().height\*scale) ,0,0,INTER\_LINEAR);

            // cv::namedWindow("Scale", cv::WINDOW\_NORMAL);

            cv::imshow("Scale", Img);

            cv::imwrite("mikasa1\_onechannell\_resize4.png", Img);

            cv::waitKey(0);

        }

};

### testMyCVTotal()

void testMyCVTotal(){ // total test, different size of mikasa picture, automatically store Phi results in the corresponding folder

    // in ./mikasaTest or ./mikasa :  4 pic: resize1,2,3,4 respectively: 396\*534, 118\*160; 59\*80; 29\*40

    // how to set marginList?

    // for max\_iter: 10000, delta\_t: 0.1 is suitable, 0.01 is too small!

    fs::path path1 = "./mikasaTest"; // test // ./ can represent the current path!

    // iteration traverse

    fs::directory\_iterator list(path1); // it works!!

    for(auto& it:list){  // resize 1,2,3,4

        fs::path file = it.path().filename();

        fs::path filenameNoExtension = file.stem();

        // cout<< filename << endl;

        string newFolderStr = "./PhiRes\_" + filenameNoExtension.string(); // folderName

        fs::path newFolder(newFolderStr);

        if( ! fs::exists( newFolder ) ){

            if( fs::create\_directory(newFolder) ){

            cout << "create: "<< newFolder << endl;

            }

        }

        // operate the file. use new folder as output folder

        cv::Mat img = cv::imread(file.string(), cv::IMREAD\_UNCHANGED);

        vector<int>DeltaTdigit({1,2});

        // different MarList for different size?

        vector<int>MarList;

        vector<int>MarList1({1,10,30,50,100});

        vector<int>MarList2( {1,10,20,30,40});

        vector<int>MarList3({1,5,10,20});

        vector<int>MarList4({1,5,10});

        string filenameNoExtensionStr = filenameNoExtension.string();

        int resizeNum =  static\_cast<int>(filenameNoExtensionStr.at(filenameNoExtensionStr.size()-1)) - '0';

        switch (resizeNum){

            case 1:

                MarList = MarList1;

                break;

            case 2:

                MarList = MarList2;

                break;

            case 3:

                MarList = MarList3;

                break;

            case 4:

                MarList = MarList4;

                break;

            default:

                break;

        }

        for(auto digit : DeltaTdigit){  // iterator ! auto!

            for(auto mar : MarList){

                myCV cv = myCV(img, mar); // margin of Phi edge to outer:5, thickness of Phi edge is 1  // a margin bigger than 1 is better for cv! suitable margin is important

                cv.ImgInfo();

                cv.setDeltT(digit);  // 0.1 is enough for 10000 iter, 0.01 too small

                cv.fit(1e4); // max\_iter

                cv.LastPhiOutput(newFolderStr);

            }

        }

    }

}

### Main()

int main(){

    cout << "I am a fool but I can make it" << endl;

    testMyCVTotal();

    cout << "YES!" << endl;

    return 0;

}

## 实验二 RSF