Duke University
Department of Mathematics

Platypus - Cradle Removal Matlab code Manual

Developer:
Rujie (Rachel) Yin

June 2015

Preface

The digital cradle removal project is one of the art-related projects, originally proposed by my advisor Prof. Ingrid Daubechies, in which I am involved. These projects aim at interdisciplinary collaboration between STEM (science, technology, engineering, mathematics) and humanities, in particular, Mathematics, Electrical Engineering and Art History.

I am glad to receive encouragement not only from my advisor and colleagues, but also from many art conservators whom I collaborate with. I hope my work can assist them to discover more and learn more in art conservation.

The Matlab code I wrote within the past two years is a prototype of my proposed algorithms in the following two papers, *Digital Cradle Removal in X-ray Images of Art Paintings* (presented at ICIP 2014) and *Removing cradle artifacts in X-ray images of paintings* (SIAM preprint). The first conference paper provides a nice overview of the basic algorithm and the second journal paper includes more details and an extension of the previous algorithm.

The potential reader of this manual is expected to be from STEM and wishes to further develop my algorithm in Matlab as a prototype. For those who are looking for deployed C code, please refer to the C code written by Gabor Fodor, who spent tremendous effort translating my Matlab code and making improvement for practical application. For art conservators who are end users looking for the cradle removal tool, please download the Photoshop plugin *Platypus* and try it out on your own X-rays!

The Matlab code is free to download and is meant for academic research ONLY. However, users are responsible for any consequential damages from using the code. No commercial usage is permitted.

Rujie (Rachel) Yin
Durham
June 2015

Contents

List of Figures

Chapter 1

Overview

This chapter gives an overview of the cradle removal algorithm, introducing key functions and variables that are used in the process.

1.1 Background

In this section, we introduce briefly the background of the *digital* cradle removal problem. Fig.1.1 shows a typical cradle structure attached to the back of a panel painting. There are two types of cradling members: fixed versus sliding. The difference between these two types of cradling members is significant in an X-ray image. The fixed member shows up as an entire piece without any slots in between, whereas the sliding member shows up as an array of segments aligned perpendicular to the fixed member, often with dark slots at the intersections. This is because tunnels are cut in the fixed members allowing sliding ones go through, see Fig.1.1. Moreover, the wood grain of fixed members is always aligned in the same direction as that of the wood panel. In general, sliding members show up in horizontal direction and fixed members show up in vertical direction, which is also the convention used in this manual.

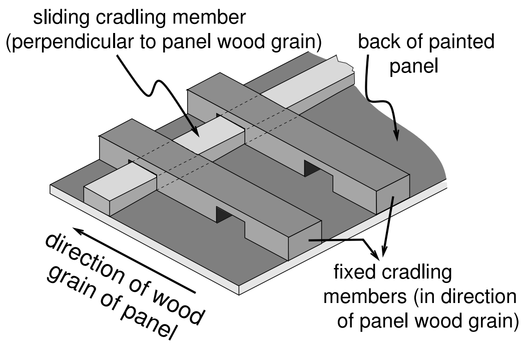
sliding cradling member
(perpendicular to panel wood grain)
back of painted panel
direction of wood grain of panel
fixed cradling members (in direction of panel wood grain)

Figure 1.1: A typical cradle structure on the back of a panel painting

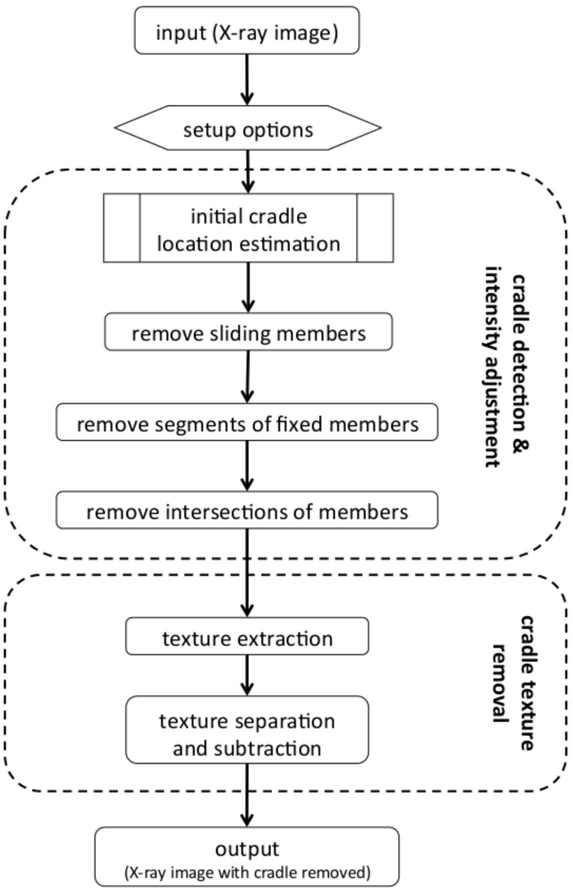
input (X-ray image)
setup options
initial cradle location estimation
remove sliding members
remove segments of fixed members
remove intersections of members
cradle detection & intensity adjustment
texture extraction
texture separation and subtraction
cradle texture removal
output
(X-ray image with cradle removed)

Figure 1.2: Pipeline of cradle removal algorithm

1.2 Algorithm Pipeline

The cradle artifact in an X-ray image (input) consists of two parts: a smooth intensity difference with rectangular shape and the wood grain (texture) coming from the cradle. These two parts are removed successively in two stages: the cradle detection and intensity adjustment and the cradle texture removal, as shown in the pipeline in Fig.1.2. Within each stage, the process is further broken down into automatic/semi-automatic steps.

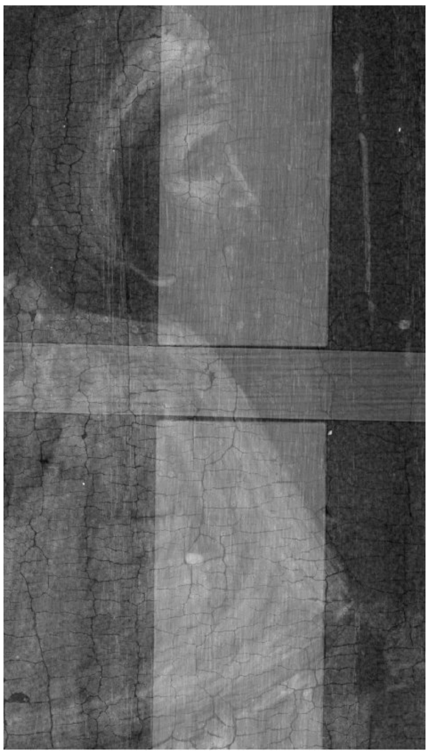

Figure 1.3: Cropped patch from X-ray of Ghissi altarpiece, *Acteus and Eugenius Implore St. John the Evangelist to Restore Their Wealth*, North Carolina Museum of Art

1.3 Input Demo

Throughout this manual, we will use the demo image shown in Fig.1.3, which is a cropped patch from the X-ray of *Acteus and Eugenius Implore St. John the Evangelist to Restore Their Wealth*, currently in the North Carolina Museum of Art. The dimensions of the image are 474×839 pixels. If you choose to remove cradle artifacts in your own X-ray image, please make sure that it satisfies the following properties:

- There is no background on the boundary of the image.
- There are no cradle members on the boundary of the image, all cradle members appear in full width.
- There are minimum defects in the X-ray (the impact of defects depends on the size and the location of the defects).
- All cradle members should have a rectangular shape, curved non-polygon cradle shapes can not be handled by the algorithm.

Chapter 2

Cradle Detection and Intensity Adjustment

The first stage of the cradle detection workflow is crucial for the final result as the detected cradle location will be used in the second stage. In some cases, where the wood grain of cradle doesn't appear in X-ray images, it suffices to only apply the first stage of the process.

The pipeline of the first stage is depicted in Fig.1.2 and can be further broken down into the workflow shown in Fig.2.1.

The rest of this chapter follows the matlab script `CradleRemovalMain.m`, and goes through the major functions and parameters used in the first stage.

2.1 Load image

```
%% load demo image
img = double(imread('../figure/rawX-ray.jpg'));
img = mean(img,3);
```

The image must be converted into a `double` grayscale image.

2.2 Initial cradle location estimation

This step is an initial automatic estimation of the location of cradle members relying on the differences of pixel values across the edges of cradle members.

```
% step 1: estimate roughly the location of horizontal and vertical cradles
% basic parameters setup for cradle detection
vn = 1; % number of vertical cradles
vrange = [1 , size(img,2)]; % the range of vertical cradles, get rid of zero ...
    background
hn = 1; % number of horizontal cradles
hrange = [ 1 , size(img,1)]; % the range of horizontal cradles, get rid of ...
    zero background
```

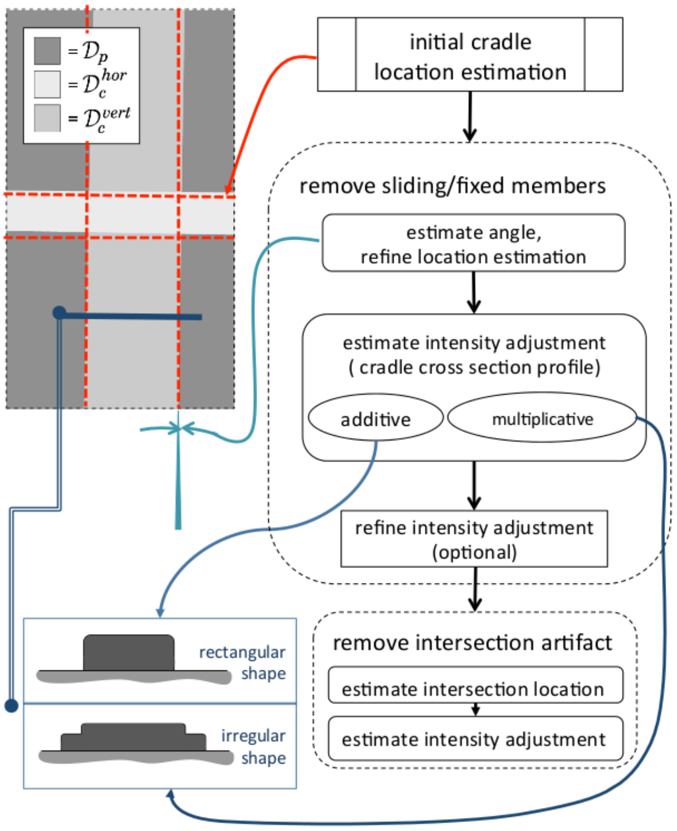
= $\mathcal{D}_p$
= $\mathcal{D}_c^{hor}$
= $\mathcal{D}_c^{vert}$
initial cradle location estimation
remove sliding/fixed members
estimate angle, refine location estimation
estimate intensity adjustment (cradle cross section profile)
additive
multiplicative
refine intensity adjustment (optional)
remove intersection artifact
estimate intersection location
estimate intensity adjustment
rectangular shape
irregular shape

Figure 2.1: Cradle detection and intensity adjustment work flow.

```
opt = struct();% options
opt.L = 20;% length of Haar filter
opt.s = 5;% smoothing parameter, input of built-in function smooth
opt.display = 0;% do not display intermediate estimation result
[verest,horest] = cradledetect(img,vn,vrange,hn,hrange,opt);
```

Input

To estimate the location, you need to provide the number of horizontal cradle members (`hn`) and the number of vertical cradle members (`vn`).

You may also help the algorithm by excluding unnecessary background along the boundary by setting the parameters `hrange, vrange`.

The structure `opt` contains options of model settings as well as feedback from functions when it is an output. *Note: For images with finer resolution, increase `opt.L`, `opt.s` to get better estimation.*

Output

This initial guess will treat cradle members as if they are in perfect horizontal or vertical direction. Hence, for a horizontal cradle member, one only need to record the vertical coordinates of two edges `horest`, and similarly, for a vertical cradle member, one records the horizontal coordinates of two edges `verest`. See the red dashed lines in Fig.2.1.

Functions

`cradledetect.m`

Walkaround

If the output estimations `horest, verest` are completely off, you can manually set them using the coordinates of the middle points on the edges.

2.3 Remove horizontal (sliding) cradle artifact

The horizontal (sliding) cradle members are removed first, because they are not cut and show up as a whole piece in X-ray image. Instead of working on the full size image, the sub patch containing each horizontal member with decent amount of surrounding region is cropped out and used in the proces.

```
% step 2: estimate the accuracy of the estimation of verest & horest
opt.hs = 10;
opt.vs = 20;
opt.smoothbd = 0; % indicate if the boundary is smooth (noisy)

% step 3: remove all horizontal cradles
opt.model = 'multiplicative'; % 'additive'
opt.profile_estimation = 'edge'; % 'edge' for edge profile estimation only, ...
    'section' for full cross section
[imgnew,hinfo] = RmHorizontalCradle(img,horest,verest,opt);
```

Input

We should first let the algorithm know the accuracy of the initial guesses for `horest` and `verest`. For example, if the vertical coordinates of the edges of a horizontal member have differences to the corresponding estimation `horest` within 10 pixels, then set `opt.hs = 10`.

Indicate if the edge of the cradle artifact is smooth or not. Different methods will be used to refine the estimation of edge locations. Setting `opt.smoothbd` to 1 for sharp edges in the image will leads to less accurate result.

The model used for intensity adjustment is specified by setting `opt.model`. Follow the rule of thumb in Fig.2.2 to choose a proper model from `'additive'`

and `'multiplicative'`. In general, you can first try out the `'multiplicative'` model, and if it doesn't work well, then switch back to the `'additive'` model.

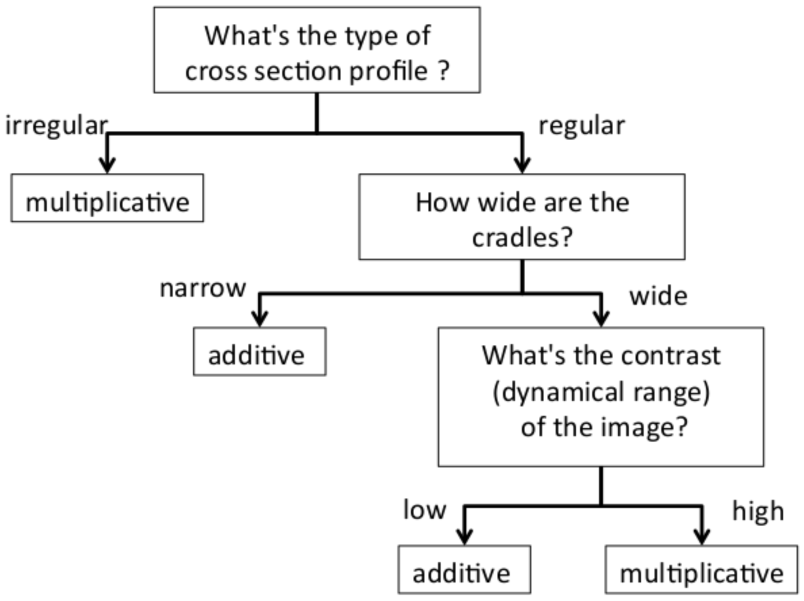
What's the type of cross section profile ?
irregular
regular
multiplicative
How wide are the cradles?
narrow
wide
additive
What's the contrast (dynamical range) of the image?
low
high
additive
multiplicative

Figure 2.2: Decision tree for model selection

Specify `opt.profile_estimation` only if you use the `'multiplicative'` model. If the cross section of your cradle is irregular (i.e. non-rectangular), use `'section'`. Otherwise, use `'edge'`.

Output

`imgnew`	the result image with horizontal cradle artifact removed, see Fig.2.3.
`hinfo`	a `cell` array containing information of horizontal cradle member, including
`rowind, colind`	indices of cropped patch containing the cradle
`lp,rp,ld,rd`	location of the four vertices (top-left, top-right, bottom-left, bottom-right) of the polygon detected as cradled region
`cradleimg`	cradle artifact subtracted from the image patch
`angle`	angle of the cradle member detected using the Radon transform
`dI`	maximum pixel value difference of cradled and non-cradled region
`width`	width of edge neighbourhood used `cradleimg`, parameter of additive model
`p1profile, C`	parameters of multiplicative model

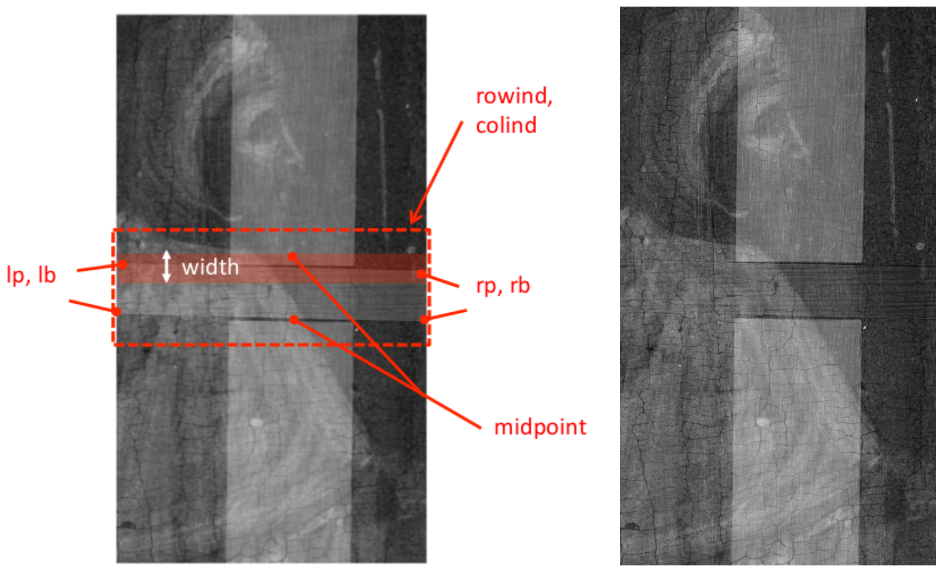

Figure 2.3: Left: `hinfo` output diagram; Right: `imgnew` using multiplicative model

Functions

`RmHorizontalCradle`

- `rm_single_cradle` (nested)
  - `BackProjection` for additive model
  - `cradle_attenuation_fitting` for multiplicative model
    - `RemoveAttenuationProfile`

Walkaround

When using the additive model, if the edge profile is not correctly detected and the non-cradled region near the cradle member is darkened, try changing `opt.s` and redo the process. It could be that `opt.s` is set too big.

If the edge profile is detected correctly, but the intensity difference `hinfo.dI` is not estimated correctly, then you can prefix it to any value you want by setting `opt.HdI`. If there are more than one horizontal member, then either a scalar `opt.HdI` is used for all members, or a vector `opt.HdI` is used for treating each member separately.

```
% step 3: remove all horizontal cradles using additive model
% with prefixed intensity difference
opt.model = 'additive';
opt.HdI = 50 ; % the intensity difference induced by horizontal cradle
[imgnew,hinfo] = RmHorizontalCradle(img,horest,verest,opt);
```

2.4 Remove vertical (fixed) cradle artifact

After the horizontal cradle members are removed, the vertical cradle artifact shows up in segments in `imgnew`. Each vertical segment is cropped out locally and rotated into the horizontal direction, so that it can be treated as a horizontal cradle member.

It is preferable to remove the vertical cradle members by segments, because some segments are easier to remove than others that are part of the same vertical cradle member. Thus we can refine the intensity adjustment on segments that were not well removed by using the profile learned from segments that are part of the same cradle member and were removed well.

```
% step 4: remove all vertical cradles
Iflag = [ 0, 0]; % indicate whether the very left or right cradle is adjacent ...
    to zero background
[I ,vinfo] = RmRegularVerticalCradleSegmentation(imgnew,hinfo,verest,opt);
```

Input

`Iflag` indicates if there is a background on the border, so that the background pixel value won't be erroneously used in the intensity adjustment estimation.

Output

`I`	the result image with vertical cradle artifact removed, see Fig.2.4.
`vinfo`	same as `hinfo`
`upcutind, downcutind`	midpoint location of edges on the intersection with horizontal members

Functions

- `RmRegularVerticalCradleSegmentation`
  - `cradledetect` re-estimate cradle location for each segment
  - `RmHorizontalCradle`

Walkaround

In 'multiplicative' model:

```
% step 4: remove all vertical cradles
Iflag = [ 0, 0]; % indicate whether the very left or right cradle is adjacent ...
    to zero background
[I ,vinfo] = RmRegularVerticalCradleSegmentation(imgnew,hinfo,verest,opt);
```

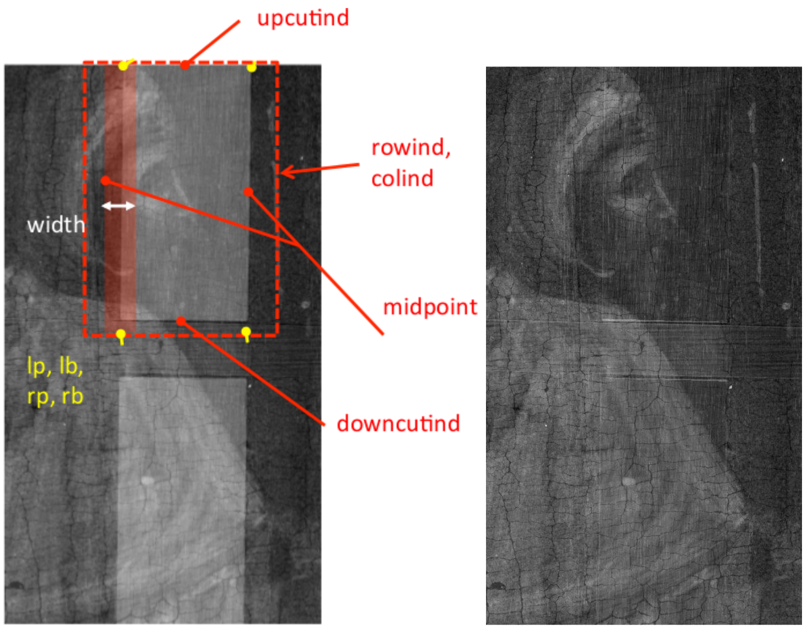

Figure 2.4: Left: `vinfo` output diagram; Right: `I` using multiplicative model

```
% ============optional set edge location using vinfo =====================%
% ******* using multiplicative attenuation model **********%
 i1 = 1; j1 = 1; % cradle to be reprocessed
 i2 = 2; j2 = 1; % cradle whose profile to be used
 [~,vinfo,I] = PostRmCradleProfile(imgnew,vinfo,i1,j1,i2,j2,I);
```

If the profile (model parameters) of the segment at (`i1`,`j1`) is not estimated correctly, you can use the profile of the segment at (`i2`,`j2`) to remove the segment at (`i1`,`j1`) by calling `PostRmCradleProfile`.

In 'additive' model:

```
% step 4: remove all vertical cradles
Iflag = [ 0, 0]; % indicate whether the very left or right cradle is adjacent ...
    to zero background
% ============optional set edge location using vinfo =====================%
% ****** using additive model ********%
opt.edgeloc_v = cell(size(vinfo)); % adjust/prefix vinfo.midpoint
opt.edgeloc_v{1} = [176; 367];
opt.edgeloc_v{2} = [169; 359];
opt.VdI = [100, 120];% the intensity difference induced by vertical cradle
[I, vinfo] = RmRegularVerticalCradleSegmentation(imgnew,hinfo,verest,opt);
```

To prefix edge location of each segment, you can provide `opt.edgeloc_v`, which is a cell array of (# of segments per cradle member) $\times$ (# of vertical members). Each cell contains the prefix `midpoint` value of the corresponding cradle segment.

Like the previous case, you can provide `opt.VdI`, which contains the prefix `dI` value of each `vinfo` cell. You can first call `RmRegularVerticalCradleSegmentation`, and investigate `vinfo` then deside how to set `opt.VdI`.

2.5 Remove intersection of cradle members

Finally, the intersections of horizontal and vertical cradle members are removed. This can be hard, because the intersection region is usually small and has much smaller intensity difference after the removal of horizontal cradle artifact. Therefore, the estimation is less robust.

```
% step 5: remove intersection of cradles
opt.crossmodel = 'linearfit';% 'linearfit' if the crosssection is free of ...
    panel content
[Inew,crossinfo] = Rm_cross_section(I,hinfo,vinfo,opt);
```

Input

For the 'multiplicative' model, use `linearfit` as primary option, which means fitting the model in the intersection region of `I` directly.

For the 'additive' model, there is no input to specify.

Output

`Inew`	the result image of stage 1, see Fig.2.5.
`crossinfo`	same as `hinfo`, `vinfo`
`cutwidth`	the width of the slot between the horizontal member and the vertical member

Functions

- `Rm_cross_section`
  - `connect_vinfo` pull segment information `vinfo` into full member information
  - `cradle_attenuation_fitting` for multiplicative model, 'linearfit'
  - `RemoveAttenuationProfile` for multiplicative model, 'profilefit'

Walkaround

In the 'multiplicative' model:

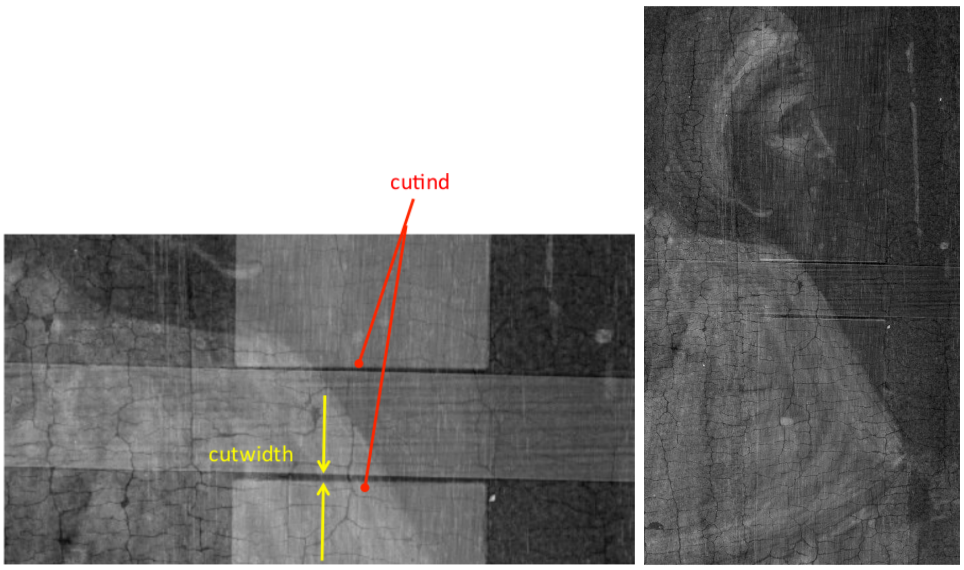

Figure 2.5: Left: `crossinfo` output diagram; Right: `Inew` using multiplicative model

```
% step 5: remove intersection of cradles
 opt.crossmodel = 'profilefit';% 'profilefit' if the linearfit fails
 [Inew,crossinfo] = Rm_cross_section(I,hinfo,vinfo,opt);
 scale = .5;
 Inew = (Inew-I)*scale + I;
```

If `linearfit` fails, use `profilefit`, which means using the profile of two adjacent vertical segements to remove the intersection artifact. Since the intensity difference at an intersection is smaller than that in a vertical cradled region, the difference `Inew-I` has to be rescaled by `scale`.

2.6 Post-processing (optional): remove boundary artifacts

The final result `Inew` obtained may suffer from boundary artifacts along the edges of cradle members. There are two main causes for such artifacts: a defect of the X-ray imaging and an estimation error in stage 1.

Such boundary artifacts show up as long thin extra bright or dark slots around the edges of cradle members, which can be accurately detected and removed with a Shearlet/Curvelet transform.

This post-processing is relatively slow because of the usage of the Shearlet transform.

```
% step 6: remove boundary artifact along edges (optional)
img = Inew;
RmBoundary;
```

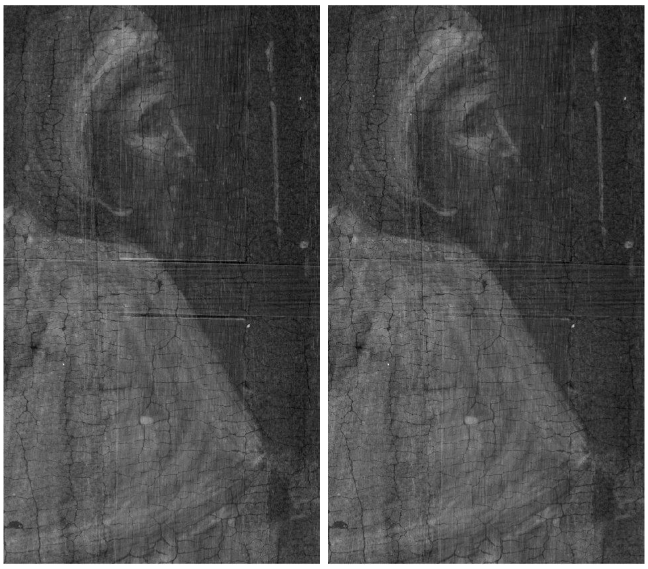

Figure 2.6: Left: `Inew`, stage 1 result with boundary artifact along the cutting edges of the vertical cradle member; Right: post-processing result.

Input

`hinfo, vinfo` and `crossinfo` are required to compute the exact location of edges.

Output

See Fig.2.6.

Functions

The shearlet transform uses the implementation of Fast Finite Shearlet transform (FFST) toolbox, `http://www.mathematik.uni-kl.de/imagepro/software/ffst/`

- `RmBoundary`
  - `info2mask` compute mask of cradle region from `info`'s
  - `BoundaryImg` extract boundary artifact
    - `shearletTransformSpect, inverseShearletTransformSpect` Shearlet and inverse Shearlet transform from toolbox FFST

Chapter 3

Cradle Texture Removal

The second stage of the cradle removal algorithm separates the wood grain of cradle members as much as possible in order to maximise the enhancement of the X-ray image.

All the processing in this stage is done on blocks of the resulting image from stage one. Therefore, it can be parallelised and run on a cluster to speed up. The division of a full image into blocks and the merging of blocks back to a full image is implemented in a naive way, hence artifacts around the boundaries of blocks may arise in the result.

The pipeline of the first stage in Fig.1.2 can be further broken down into the workflow shown in Fig.3.1.

The rest of this chapter follows the matlab script `TextureRemovalMain.m`, and goes through the major functions and utilities used in the second stage.

3.1 Texture extraction

The texture extraction from an image uses Morphological Component Analysis (MCA), which decomposes an image into its 'cartoon' component and its 'texture' component. The two components are encoded using two competing transforms (or dictionaries) that are optimized for different types of signals. The MCA decomposition in this algorithm uses the Dual-tree complex wavelet transform (DTCWT) for the 'cartoon' component and a curvelet transform for the 'texture' component. Further, the DC component of the curvelet transform is set to zero, such that the low frequency part of the image is always contained in the 'cartoon' part.

```
opt.transform = 'shearlet';
opt.direction = 'horizontal';

% decompose image to blocks
[subimg,orow,ocol,rowind,colind] = img2subpatch(img);

%% step 1: MCA decomposition into cartoon and texture
cartoon = cell(size(subimg));
texture = cell(size(subimg));
for i = 1:size(subimg,1)
    for j = 1:size(subimg,2)
        img = subimg{i,j};
```

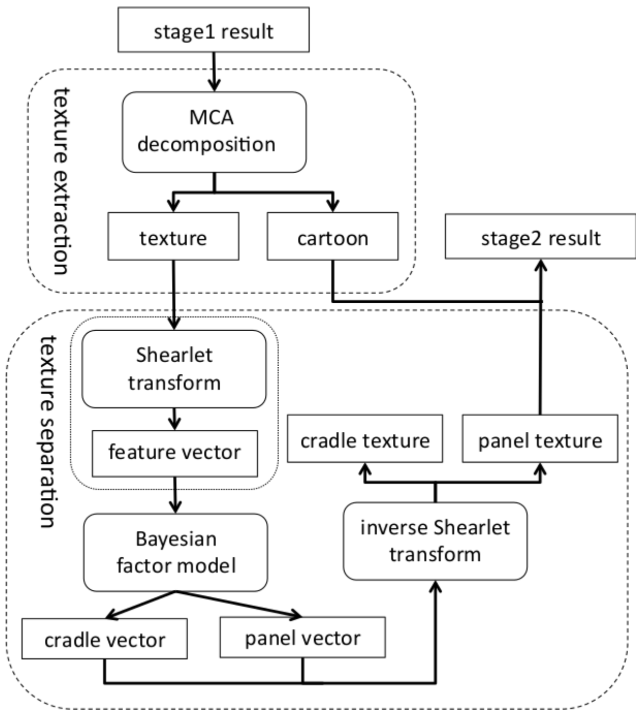

Figure 3.1: Texture extraction and separation work flow.

```
        MCAdecomp;
        cartoon{i,j} = parts(:,:,1);
        texture{i,j} = parts(:,:,2);
    end
end
```

Input

`subimg`, see Fig.3.2.

Output

See Fig. 3.3

Functions

- `MCAdecomp`
  - `FastCDWT2Analysis`, `FastCDWT2Synthesis` function wrappers for DTCWT

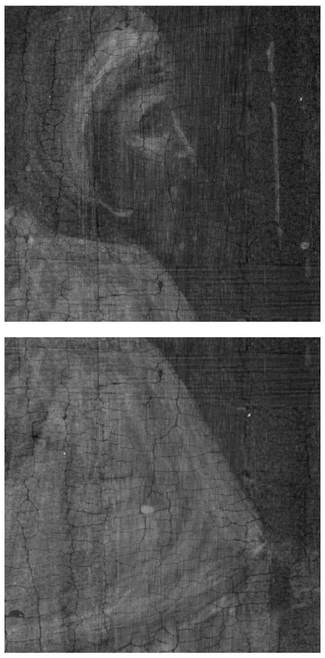
Figure 3.2: `subimg` image blocks of stage 1 result.

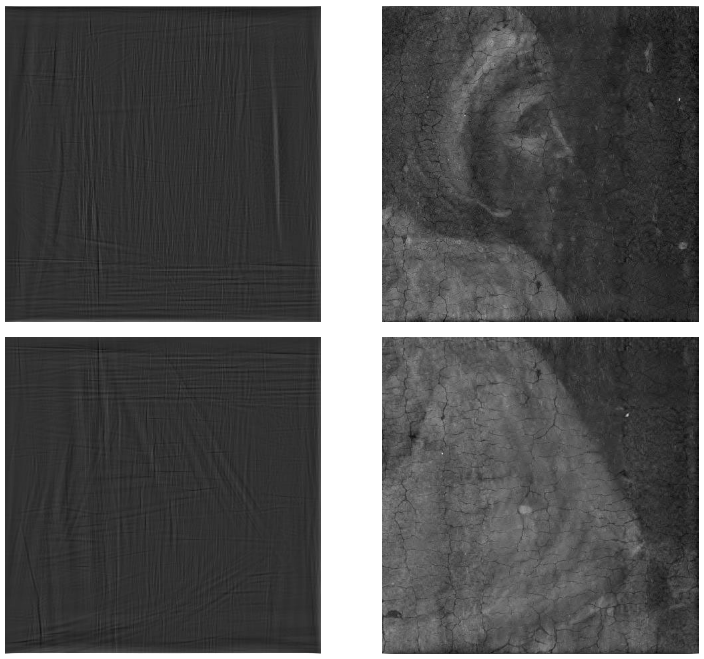
Figure 3.3: MCA decomposition of `subimg`. Left: `texture`; Right: `cartoon`.

  - `FastCURVWRAPAnalysis`, `FastCURVWRAPSynthesis` function wrappers for Curvelet transform

The MCA decomposition uses the MCALab toolbox, `https://fadili.users.greyc.fr/demos/WaveRestore/downloads/mcalab/Download.html`.

The Curvelet transform uses the Curvelab toolbox, `http://www.curvelet.org/`.

The dual-tree complex wavelet transform uses the dtcwt toolbox by Nick Kingsbury and Cian Shaffrey.

All these toolboxes and their dependencies are provided in this package. For correct installation, please follow the instruction on the websites, whose links are provided above.

`MCAdecomp` is an iterative algorithm solving an optimization problem. All the parameters used are the same as the default setting in MCALab. You may use a smaller maximum number of iterations, `itermax`, according to the convergence of the algorithm, or change the stopping criterion to decrease the computation time.

3.2 Texture separation I: compute feature vector

After the texture image is extracted, it is separated using a Bayesian factor model, which separates each input signal into a cradle signal and a panel signal. Instead of treating the pixels in the texture image `texture` directly as signals, the algorithm first takes the Shearlet transform of `texture`, such that a feature vector (i.e. a vector of Shearlet coefficients) is generated at the location of each pixel. These feature vectors are separated by using the Bayesian model, and the cradle feature vectors and the panel feature vectors are then converted back to a cradle texture image and a panel texture image respectively by taking the inverse Shearlet transform. See the workflow in Fig.3.1.

```
%% set path to folder which saves all intermediate results
result_path = '../result/';

%% step 2: Bayesian factor model for texture separation

% prepare feature vectors as input of factor model
computeShearletDescriptor;
```

Input

`texture` in Fig.3.3 and `hinfo, vinfonew` for labelling feature vectors in cradle and non-cradle region.

Output

`hfeatureST_*.mat,rfeatureST_*.mat` saved in `result_path`.

- `featureST.mat` Matlab data file containing the horizontal/vertical feature vectors obtained form `texture`
  - `featureST.cradle` feature vectors from the cradled region

`featureST.free` feature vectors from the non-cradle region

– `cradle_sample_size` number of feature vectors in `featureST.cradle`

`noncradle_sample_size` number of feature vectors in `featureST.free`

Functions

- `computeShearletDescriptor`
  - `getSTfeature` extract feature vectors from Shearlet coefficients
  - `shearlet_angleselection` select Shearlet coefficients associated to near horizontal or near vertical directions

Since the Shearlet transform breaks down `texture` into textures of different directions and we are only interested in textures in near horizontal and vertical directions that align with the directions of cradle members, Shearlet coefficients associated to other directions are discarded. This angle selection decreases the length of feature vectors and benefits the Bayesian factor model computation that follows.

3.3 Texture separation II: Bayesian factor model

On each image block and for texture in each direction (horizontal or vertical), a Bayesian factor model is learned using a relatively small number of samples from the labeled (cradle vs. non-cradle) feature vectors in `featureST.mat`. The learned model is used for posterior inference on all the feature vectors with label 'cradle' that needs to be separated. Finally, the separation result on the feature vectors are reassembled and converted back to the image domain using the inverse Shearlet transform.

```
% run MCMC of Bayesian model
dir = 'h';
computeBayesianSeparation;

dir = 'v';
computeBayesianSeparation;
```

Input

`featureST.mat`

Output

`FactorModel_*.mat`	learned Bayesian factor model
`Bayesian_result_*.mat`	posterior inference on `FactorModel_*.mat`
`originImg`	input texture image for separation
`result`	separated cradle texture image
`residual`	residual texture image in the cradled region that cannot be identified as cradle or non-cradle texture

Fig.3.4 shows the result of texture separation running one trial of the Bayesian factor model. Since the Bayesian model is a probabilistic model, the separation result is not deterministic, but close to a steady state.

Functions

- `computeBayesianSeparation`
  - `PrepareData4spFactorModel` preprocessing feature vector samples for factor model input
  - `spfactcovest_mgploadings` compute factor model
  - `spfactor_post_inference` posterior inference on all feature vectors
  - `invfeatureST` convert feature vectors back to image domain

Sampling in large image

When processing a large image, such as Fig.3.5, the whole image is divided into many blocks, such that each block may not contain enough cradle or non-cradle feature vectors to sample from. Further, we need to keep the consistency between the Bayesian factor models learned in adjacent blocks, therefore, it's necessary to take samples of labeled feature vectors in a block neighbourhood. As shown in Fig.3.5, for non-cradle feature vectors, samples are taken in a 8-neighbourhood of the current block, whereas for cradle feature vectors, samples are taken in a 2-neighbourhood of the current block, which contains the same cradle member.

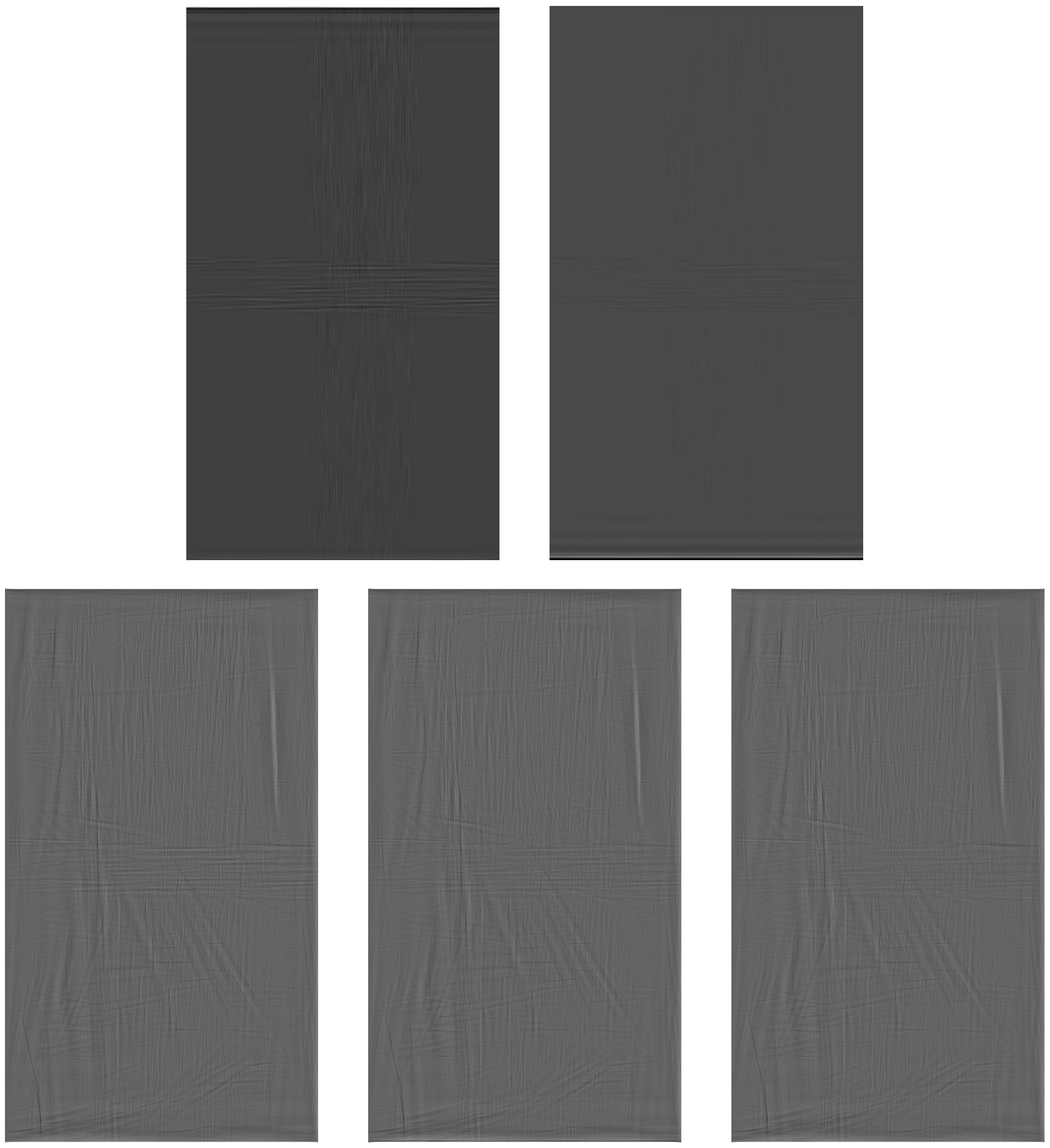
Figure 3.4: Bayesian factor model result. Top left: cradle texture `result`, top right: residual texture `residual`, bottom left: `texture`, bottom middle: `texture - result`, bottom right: `texture - result - residual`.

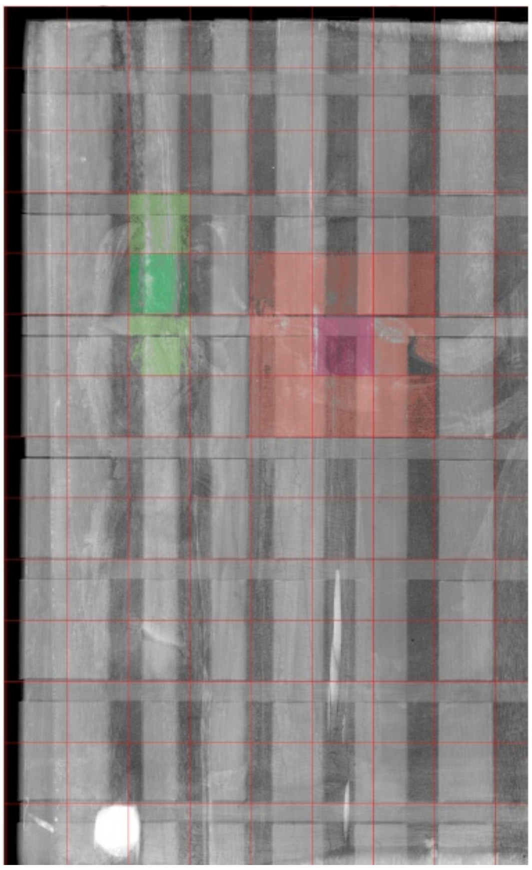

Figure 3.5: Sampling strategy for cradle (green) and non-cradle (red) feature vectors.