ECE 5780 Project Final Report
Vertebrae Localization in Pathological Spine CT using Random Forest

Hang Chu

May 12, 2015

1 Problem Overview

In this project, we investigate the problem of automatic vertebrae localization in pathological spine CT images. Input of our algorithm is a 3D spine CT image, outputs of our algorithm are number of detected vertebrae and their 3D locations. We are particularly interested in investigating the effectiveness of random forest, which has recently become popular in the computer vision and medical image research communities.

2 Clinical Significance

Accurate localization of vertebrae in spinal imaging is crucial for the clinical tasks of diagnosis, surgical planning, and post-operative assessment.

Reliable vertebrae identification could greatly reduce the risk of wrong-level surgery. According to [5], wrong-level surgery has become a unique problem in spine surgery. The consequence of wrong-level spine surgery not only generates another surgery of the intended level, it is usually also associated with lawsuit ranging from \$62,000 to \$1,500,000. This problem can be effectively avoided with the aid of an automatic vertebrae localization system.

3 Related Work

Automatic vertebrae localization has become a upraising research topic in the medical image analysis community. Automatic vertebrae localization provides useful, fundamental analysis for many advanced applications, such as longitudinal spine CT registration [6], which relies on vertebrae localization as the first step. [1] presents an effective, state-of-the-art vertebrae localization method, it is shown to outperform previous methods such as [7]. Previous methods normally rely on statistical models of shape and appearance, which is less robust compared to the random-forest-based method in [1]. The technique described in the paper has provided the major inspirations of our method in this project. Many datasets has been published to accelerate the development of effective vertebrae localization methods. In [2], a database of 224 annotated 3D spine CT images is available. In [4], 6 datasets along with challenges on spine CT images are provided. Random forest has become a popular learning method in recent years, it is originally proposed in [3] and has been shown to be able to deal with large amount of complex structured data.

4 Approach & Execution Results

Our method is inspired by the paper of Glocker et al. [1]. [1] presents the state-of-the-art method in automatic vertebrae localization, where the structures of vertebrae are learned by a random forest using huge number of densely-labeled

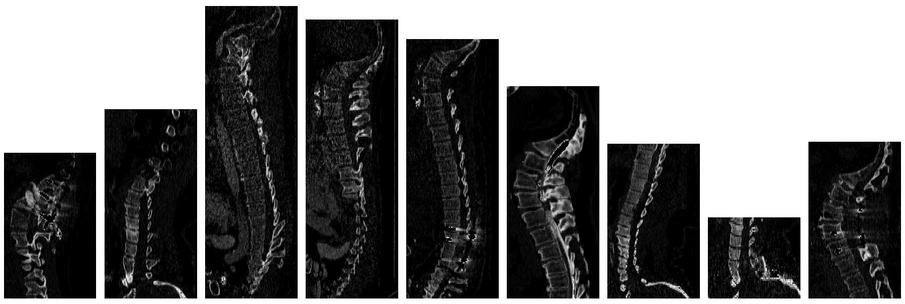

Figure 1: Examples of the training dataset. All samples are 3D, this figure only shows one vertical slice of the image.

3D image patches. Our method in this project follows the idea of [1], despite not being exactly the same in all implementation details.

4.1 Collecting the training data

We use the dataset provided by [1], which includes 224 3D spine CT images. Along with 3D spine CT images, the dataset also provides manually labeled 3D positions of vertebrae. In our actual experiments, we decide to use a small subset of the dataset due to the limit of our hardware resource. We use 9 3D images as the training dataset, which are shown in Figure 1.

For each 3D image in the training set, we randomly select 100 3D patch centers. For each 3D patch, we record three types of data:

- 1. Image intensities of the neighboring region. We divide the a $40mm \times 40mm \times 40mm$ cubic 3D region near the patch center into $40 \times 40 \times 20$ voxels, and compute the intensities of voxels by nearest-neighbor interpolation. We concatenate the intensities to form the data vector, which has dimension of 32000. In total, the size of training data is $9 \times 100 \times 32000$.
- 2. The label of the nearest vertebra center to the generated 3D patch center. The set of labels contains the regular 7 cervical, 12 thoracic, 5 lumbar and 2 additional centroids on the sacrum.
- 3. The 3D distance to the nearest vertebra center.

We collect two datasets using the procedure described above. For the first dataset, the 3D patch centers are generated from the entire space of the 3D CT image. We denote this dataset by T_1, it is used for learning where the vertebrae are. For the second dataset, the 3D patch centers can be only within $20mm$ distance to a vertebra center. We denote this dataset by T_2, it is used for learning detailed structures of vertebrae.

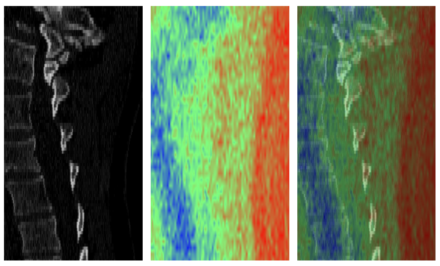

Figure 2: From left to right: the testing image used; the result after applying the regression forest F_1 where the color blue indicates small distance and red indicates large distance; the regression result overlaid with the testing image.

4.2 Training using random forest

We use the MATLAB implementation of random forest provided by [8]. It has two functionalities: random forest for classification, and random forest for regression. When training the forest, we use 20 trees as recommended in [1]. At each node in the tree, we split the data using $\sqrt{32000} = 179$ randomly selected dimensions in the data vector, as recommended in [3].

We ran training on a normal PC with 4G RAM. We observed that the average training time of one tree is around 9 seconds, using 900 samples. We also tried using more samples, but the training time increases significantly (by at least one magnitude, using 4500 samples, training one tree does not complete after 10 minutes of running). This is because each sample has 32000 dimensions, using too many samples exceeds the memory limits, the operating system automatically activates the virtual memory, which is by several magnitudes slower.

4.3 Step 1: rough vertebrae localization

The first step of our system is to roughly find where the vertebrae are, i.e. find the spinal region from the entire CT image. To do this, we train a regression forest F_1 using T_1, where the objective is to regress the 32000 dimensional data vector to the 1 dimensional scalar of the distance to the nearest vertebra center. To test the performance, we used a new 3D CT image and densely sampled 3D patches on a vertical slice, i.e. all patch centers of the testing set have the same x-coordinate. It should be noted that we choose to test on one slice of the 3D image because even for one slice, the testing data size is 3822×32000, which is near to the memory limit that a normal PC can handle. Testing on the entire 3D image would possibly demonstrate more convincing results, but the computational resource needed would be beyond what we have in this project.

Figure 2 shows the testing result. It can be seen that our regression forest method works quite effectively, most of the the spinal region is found successfully. This result met our expectation: as CT images are calibrated, intensities correspond to the actual substance, and the spinal region has a unique range of

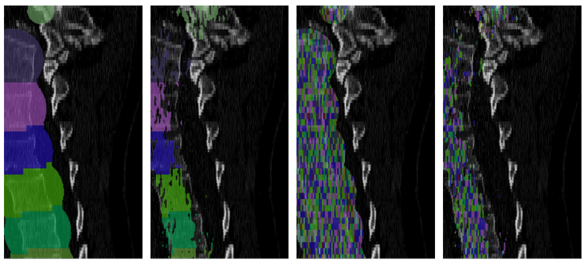

Figure 3: From left to right: the ground-truth with colors indicate different vertebrae; the ground-truth label with our rough vertebrae localization in step 1; the classification result within ground-truth distance-to-nearest-vertebra; the classification result with our rough vertebrae localization in step 1.

image intensity, so the regression forest should be able to roughly identify the spinal region easily.

4.4 Step 2: precise vertebrae localization

We apply the second vertebrae localization process only for regions that have result distance larger than $28mm$, when feeding the 32000 dimensional patches into F_1. We have tried two different approaches. The first approach follows the style of the method in [1]: train a classification forest F_2 using T_2, where the class label corresponds to vertebra type, which solves a 22-class classification problem. Figure 3 shows the execution results.

Apparently from Figure 3, this method does not work well. This is mainly due to two reasons:

- 1. Insufficient training data. Our training dataset T_2 only has 100 patches for each 3D image. In each 3D image, there are normally 5-20 vertebrae. This means in our training dataset, we only have 5-20 samples for each vertebra, that are used to represent the $40mm \times 40mm \times 40mm$ region within the vertebra center. With such insufficient amount of training data, it is not surprising that random forest does not learn much useful information.

- 2. Immature feature extraction. In [1], image intensities are not directly used as features. Instead, the paper mentioned very vaguely that 2000 random features efficiently implemented via integral images are used. This random feature generation is addressed by [9], without publicly available source code. In our method, we directly used the 32000 dimensional image intensity as input to the classifier.

It should be noticed that the two causes are connected. Because of the second cause, our feature vector is much longer than the feature vector in [1]. Thus, using our method only a limited number of sample patches can be used to accommodate the memory size of a normal PC, which leads to the first cause.

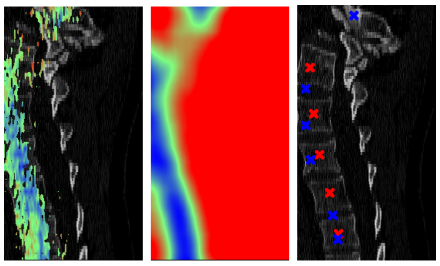

Figure 4: From left to right: the estimated distance image where the color blue to red indicates small to large distance; the template comparison error map where the color blue to red indicates small to large error; our final result (blue) compared with ground-truth (red).

There are two ways to improve the above method. The first is obvious: implementing the whole random feature generation algorithm described in [9], and combine it with our method. This is possibly the best way in order to achieve most improvement. However, doing this would not only be repeating an already-existing work, but also would demand hundreds hours of coding time. Thus, it is beyond the scope of a class project (although we think it is a worth investigating direction for researchers who are dedicated to the area related to vertebrae localization). We take the second way: modify the method such that it works in our situation.

We train a regression forest F_3 using T_2, where the objective is to regress the 32000 dimensional data vector to the 1 dimensional scalar of the distance to the nearest vertebra center. Then we apply F_3 to the testing 3D patch to estimate the distance from the patch center to the nearest vertebra center, which produces an estimation of distance-transform image. Next, for each pixel in the estimated distance image, we compute the sum of absolute difference between the pixels neighborhood and a square template, where the template records distance in millimeter to the center. This produces a map of template comparison error. Finally, we search for local minimums in the template comparison error map. The minimums are our estimated vertebra locations.

Figure 4 shows results of our method. There are 5 vertebrae in the testing image. Our method successfully identifies 6 vertebra centers, with 1 false positive. The average error in vertebra center localization is $12.01 \pm 7.55mm$, minimum error is $1.33mm$, maximum error is $19.50mm$. Although our method is less accurate compared to the state-of-the-art method [1] ($7.0mm \pm 4.7mm$ for cervical), it is able to produce somewhat useful results with a relatively simple algorithm.

5 Summary

5.1 Dataset

We use 9 3D spine CT images with labeled vertebra centers for training. 2 datasets are collected from the images:

- T_1 that has 900 3D patches of size $40 \times 40 \times 20$, each covering a $40mm \times 40mm \times 40mm$ region, randomly sampled from whole 3D image.
- T_2 that has 900 3D patches of size $40 \times 40 \times 20$, each covering a $40mm \times 40mm \times 40mm$ region, randomly sampled from regions that are within $28mm$ to a labeled vertebra center.

We use 1 3D spine CT image with 5 labeled vertebrae for testing. We densely collected 3822 3D patches on a vertical slice of the 3D image. Each patch is of size $40 \times 40 \times 20$ and covers a $40mm \times 40mm \times 40mm$ region.

5.2 Method

Our method has the following steps:

- Train two random forests F_1 and F_2 using T_1 and T_2 respectively. The trained forests regresses the $40 \times 40 \times 20$ to a 1D distance to the nearest vertebra center.
- For each 3D patch in the testing set, apply F_1 and F_2 to get d_1 and d_2.
- For center positions whose $d_1 \leq 28mm$, compute the sum of absolute errors of the 2D neighborhood around the center position and a square template of distance to template center.
- Find local minimums in the comparison result, they are the final result of estimated vertebra center locations.

5.3 Results

Each training set took 15 minutes for data collection. Testing set took 1 hour for data collection. Training each random forest took 3 minutes. The running time of our vertebrae localization algorithm took 3 minutes.

Our method detected all 5 vertebrae in the testing image, 1 false positive also occurred. The location error is $12.01 \pm 7.55mm$.

6 Conclusions

Vertebrae localization in spine CT images is a clinically significant problem that can be solved by automatic computer image analysis. We demonstrated a relatively simple, but effective method that is based on random forest. We conducted a small-scale experiment to show the effectiveness of our method.

7 References

[1] B. Glocker, D. Zikic, E. Konukoglu, D. R. Haynor, and A. Criminisi, Vertebrae Localizaton in Pathological Spine CT via Dense Classification from Sparse Annotations, *MICCAI*, 2013.
[2] Annotated Spine CT Database for Benchmarking of Vertebrae Localization and Identificaton, http://research.microsoft.com/en-us/projects/spine/
[3] L. Breiman, Random Forest, *Machine Learning* 45(1), 5-32, 2001.
[4] 2nd MICCAI Workshop & Challenge on Computational Methods and Clinical Applications for Spine Imaging, http://csi-workshop.weebly.com/
[5] J. Hsiang, Wrong-Level Surgery: A Unique Problem in Spine Surgery, *Surg. Neurol. Int.* 2(47), 2011.
[6] B. Glocker, D. Zikic, and D. R. Haynor, Robust Registration of Longitudinal Spine CT, *MICCAI*, 2014.
[7] B. Glocker, J. Fuelner, A. Criminisi, D. R. Haynor, and E. Konukoglu, Automatic Localization and Identification of Vertebrae in Arbitrary Field-of-View CT Scans, *MICCAI*, 2012.
[8] A. Jaiantilal, Random Forest for MATLAB, https://code.google.com/p/randomforest-matlab/
[9] A. Criminisi, J. Shotton, and S. Bucciarelli, Decision Forest with Long-Range Spatial Context for Organ Localization in CT Volumes, *MICCAI Workshops*, 2009.

8 Appendix

This appendix lists the source code that has been written by Hang Chu and used in this project. Source code for random forest is not listed.

```
function [ set_raw_info ] = read_set( set_file_path )
% MATLAB function for reading and formating the 3D image and vertebra labels.
% Author: Hang Chu
% Inputs:
% set_file_path: a .txt file that has format
% sample#   xsize   ysize   zsize   xunit       yunit       zunit
% for example:
% 2805012   512     512     115     0.347656    0.347656    2.5
% Outputs:
% set_raw_info: the 3D images with formats and labels
A=textread(set_file_path);
load('info-template.mat');
for ii=1:size(A,1)
    ii
    info.Filename=['../../spine-1/',int2str(A(ii,1)),'.raw'];
    info.Dimensions=[A(ii,2),A(ii,3),A(ii,4)];
    info.PixelDimensions=[A(ii,5),A(ii,6),A(ii,7)];
    V=raw_read_volume(info);
    L=read_lml(['../../spine-1/',int2str(A(ii,1)),'.lml']);
    set_raw_info(ii).V=V;
    set_raw_info(ii).info=info;
    set_raw_info(ii).L=L;
end
end

function [ Distance,Position,Features ] = extract_vectors ...
( set_raw_info_label,maxalloweddis )
% MATLAB function for extracting 3D patches and recording
% its distance to the nearest vertebra center
% Author: Hang Chu
% Inputs:
% set_raw_info_label: the 3D images with formats and labels
% maxalloweddis: the maximum allowed distance-to-vertebra-center
% Outputs:
% Distance: 900 x 1 distances
% Position: 900 x 3 center positions
% Features: 900 x 32000 image intensities
cube_size=40;
Distance=[];
Position=[];
Features=[];
offset=[];
for z=-20:2:20
    for y=-20:1:20
        for x=-20:1:20
            offset=[offset;x,y,z;];
        end
    end
end
for ii=1:length(set_raw_info_label)
    nowinfo=set_raw_info_label(ii).info;
    xrange=nowinfo.PixelDimensions(1)*nowinfo.Dimensions(1);
    yrange=nowinfo.PixelDimensions(2)*nowinfo.Dimensions(2);
    zrange=nowinfo.PixelDimensions(3)*nowinfo.Dimensions(3);
    range=[xrange,yrange,zrange];
    nowL=set_raw_info_label(ii).L;
    nowV=set_raw_info_label(ii).V;
    for jj=1:100
        display(['Image:␣',int2str(ii),',␣Sample:␣',int2str(jj)]);
        while 1
            center=rand(1,3).*(range-cube_size-4)+cube_size/2+2;
            mindis=10000;
            for kk=1:length(nowL)
                dis=sqrt(sum((center-(nowL(kk).pos)').^2));
                if dis<mindis
                    mindis=dis;
                end
            end
            if mindis<maxalloweddis
                break;
            end
        end
        Position=[Position;center];
        Distance=[Distance;mindis];
        nowdata=zeros(1,size(offset,1));
        for kk=1:size(offset,1)
            nowpos=center+offset(kk,1:3);
            nowx=round(nowpos(1)/nowinfo.PixelDimensions(1));
            nowy=round(nowpos(2)/nowinfo.PixelDimensions(2));
            nowz=round(nowpos(3)/nowinfo.PixelDimensions(3));
            nowdata(1,kk)=double(nowV(nowx,nowy,nowz));
        end
        Features=[Features;nowdata];
end
end

function [ Distance,Position,Features,cor2d ] = collect1slice ...
( testdata,xreal,zratio )
% MATLAB function for collecting all patches in one vertical
% slice for testing
% Author: Hang Chu
% Inputs:
% testdata: one sample with the 3D image, format, and labels
% xreal: x-coordinate of the vertical slice in mm
% zratio: the ratio of z-unit to x-unit
% Outputs:
% Distance: 3822 x 1 distances
% Position: 3822 x 3 center positions
% Features: 3822 x 32000 image intensities
% cor2d: 3822 x 2 2D coordinate on the slice
cube_size=40;
names=[{'C1_center'},'C2_center','C3_center','C4_center','C5_center', ...
'C6_center','C7_center','T1_center','T2_center','T3_center','T4_center', ...
'T5_center','T6_center','T7_center','T8_center','T9_center','T10_center', ...
'T11_center','T12_center','L1_center','L2_center','L3_center', ...
'L4_center','L5_center','S1_center','S2_center'];
Distance=[];
Position=[];
Features=[];
Classes=[];
```

```
offset=[];
for z=-20:2:20
    for y=-20:1:20
        for x=-20:1:20
            offset=[offset;x,y,z;];
        end
    end
end

    nowinfo=testdata.info;
    xrange=nowinfo.PixelDimensions(1)*nowinfo.Dimensions(1);
    yrange=nowinfo.PixelDimensions(2)*nowinfo.Dimensions(2);
    zrange=nowinfo.PixelDimensions(3)*nowinfo.Dimensions(3);
    range=[xrange,yrange,zrange];
    nowL=testdata.L;
    nowV=testdata.V;
    nowLnamenum=[];
    for jj=1:length(nowL)
        for ff=1:length(names)
            if strcmp(nowL(jj).name,names{ff})
                nowLnamenum(jj,1)=ff;
            end
        end
    end
    cor2d=[];
    for ii=1:4:size(testdata.V,2)
        disp([int2str(ii),'/',int2str(size(testdata.V,2))]);
        for jj=1:2:size(testdata.V,3)
            if ((ii*nowinfo.PixelDimensions(2))>(cube_size/2+3)) && ...
            ((ii*nowinfo.PixelDimensions(2))<(yrange-(cube_size/2+3))) ...
            && ((jj*nowinfo.PixelDimensions(3))>(cube_size/2+3)) && ...
            ((jj*nowinfo.PixelDimensions(3))<(zrange-(cube_size/2+3)))
                cor2d=[cor2d;ii,(jj-1)*zratio+zratio/2;];
                center=[xreal,ii*nowinfo.PixelDimensions(2),(jj-1) ...
                *nowinfo.PixelDimensions(3)+nowinfo.PixelDimensions(3)/2];
                mindis=10000;
                for ff=1:length(nowL)
                    dis=sqrt(sum((center-(nowL(ff).pos)').^2));
                    if dis<mindis
                        mindis=dis;
                        nowname=nowLnamenum(ff);
                    end
                end
                Position=[Position;center];
                Distance=[Distance;mindis];
                Classes=[Classes;nowname];
                nowdata=zeros(1,size(offset,1));
                for ff=1:size(offset,1)
                    nowpos=center+offset(ff,1:3);
                    nowx=round(nowpos(1)/nowinfo.PixelDimensions(1));
                    nowy=round(nowpos(2)/nowinfo.PixelDimensions(2));
                    nowz=round(nowpos(3)/nowinfo.PixelDimensions(3));
                    nowdata(1,ff)=double(nowV(nowx,nowy,nowz));
                end
                Features=[Features;nowdata];
            end
        end
    end
end

function [dense_D_hat] = plot1slice ...
( testdata,xcor,zratio,D_hat,D_hat_level2,cor2d )
% MATLAB function for ploting the estimated distance image, giving results
% of the two regression forests
% Author: Hang Chu
% Inputs:
% testdata: one sample with the 3D image, format, and labels
% xreal: x-coordinate of the vertical slice in mm
% zratio: the ratio of z-unit to x-unit
% D_hat: 3822 x 1 regression result of F_1
% D_hat_level2: 3822 x 1 regression result of F_2
% cor2d: 3822 x 2 2D coordinate on the slice
% Outputs:
% 3 figures will be plotted
% dense_D_hat: interpolated 2D distance image
for ii=1:size(testdata.V,2)
    for jj=1:size(testdata.V,3)
        img(((jj-1)*zratio+1):((jj)*zratio),ii)=testdata.V(xcor,ii,jj);
    end
end
img=double(img);
img(img>1500)=0;
figure;
imshow(img/1500);
disimg=zeros(size(img,1),size(img,2));
[X,Y]=meshgrid(1:size(img,2),1:size(img,1));
smallsize1=(max(cor2d(:,2))-min(cor2d(:,2)))/16+1;
smallsize2=(max(cor2d(:,1))-min(cor2d(:,1)))/4+1;
D_hat_2d=reshape(D_hat,smallsize1,smallsize2);
D_hat_2d_level2=reshape(D_hat_level2,smallsize1,smallsize2);
corx2d=reshape(cor2d(:,1),smallsize1,smallsize2);
cory2d=reshape(cor2d(:,2),smallsize1,smallsize2);
dense_D_hat=interp2(corx2d,cory2d,D_hat_2d,X,Y);
dense_D_hat_level2=interp2(corx2d,cory2d,D_hat_2d_level2,X,Y);
for ii=1:size(dense_D_hat,1)
    for jj=1:size(dense_D_hat,2)
        if dense_D_hat(ii,jj)<28
            dense_D_hat(ii,jj)=dense_D_hat_level2(ii,jj);
        end
    end
end
figure;
ddh_color=uint8(zeros(size(dense_D_hat,1),size(dense_D_hat,2),3));
minval=min(min(dense_D_hat));
maxval=max(max(dense_D_hat));
for ii=1:size(dense_D_hat,1)
    for jj=1:size(dense_D_hat,2)
        val=(dense_D_hat(ii,jj)-minval)/(maxval-minval);
        ddh_color(ii,jj,1)=round(val*255);
        ddh_color(ii,jj,2)=255-round(abs(val-0.5)*2*255);
        ddh_color(ii,jj,3)=255-round(val*255);
    end
end
imshow(ddh_color);
```

```
showimg=uint8(zeros(size(dense_D_hat,1),size(dense_D_hat,2),3));
for ii=1:size(showimg,1)
    for jj=1:size(showimg,2)
        showimg(ii,jj,1)=0.5*ddh_color(ii,jj,1)+0.5*img(ii,jj)/1500*255;
        showimg(ii,jj,2)=0.5*ddh_color(ii,jj,2)+0.5*img(ii,jj)/1500*255;
        showimg(ii,jj,3)=0.5*ddh_color(ii,jj,3)+0.5*img(ii,jj)/1500*255;
    end
end
figure;
imshow(showimg);
end

function [ pmap,vets ] = localize_vet( dense_D_hat,disresolution )
% MATLAB function for final veterbrae localization
% Author: Hang Chu
% Inputs:
% dense_D_hat: interpolated 2D distance image
% disresolution: x-unit in mm
% Outputs:
% pmap: the template comparison result
% vets: final vertebrae locations
tempwidth=64;
template=zeros(tempwidth*2+1,tempwidth*2+1);
for ii=1:tempwidth*2+1
    for jj=1:tempwidth*2+1
        template(ii,jj)=sqrt(((ii-tempwidth-1)*disresolution)^2+ ...
        ((jj-tempwidth-1)*disresolution)^2);
    end
end
pmap=zeros(size(dense_D_hat,1),size(dense_D_hat,2));
for ii=1:size(dense_D_hat,1)
    ii
    for jj=1:size(dense_D_hat,2)
        if isnan(dense_D_hat(ii,jj))
            pmap(ii,jj)=-0.1;
        else
            xmin=max([1,jj-tempwidth]);
            xmax=min([size(pmap,2),jj+tempwidth]);
            ymin=max([1,ii-tempwidth]);
            ymax=min([size(pmap,1),ii+tempwidth]);
            datamat=dense_D_hat(ymin:ymax,xmin:xmax);
            tempmat=template((tempwidth+1-(ii-ymin)):(tempwidth+1+(ymax-ii)), ...
            (tempwidth+1-(jj-xmin)):(tempwidth+1+(xmax-jj)));
            resultmat=abs(datamat-tempmat);
            thenumber=sum(sum(1-isnan(resultmat)));
            resultmat(isnan(resultmat))=0;
            pmap(ii,jj)=sum(sum(resultmat))/thenumber;
        end
    end
end
imshow(pmap/max(max(pmap)));
vets=[];
for ii=1:size(dense_D_hat,1)
    ii
    for jj=1:size(dense_D_hat,2)
        xmin=max([1,jj-30]);
        xmax=min([size(pmap,2),jj+30]);
        ymin=max([1,ii-30]);
```

```
            ymax=min([size(pmap,1),ii+30]);
            datamat=pmap(ymin:ymax,xmin:xmax);
            datamat(datamat==-0.1)=100;
            if pmap(ii,jj)==-0.1
                continue;
            end
            if dense_D_hat(ii,jj)>28
                continue;
            end
            if pmap(ii,jj)==min(min(datamat))
                vets=[vets;jj,ii];
            end
        end
    end
    hold on;
    for ii=1:size(vets,1)
        plot(vets(ii,1),vets(ii,2),'rx');
    end
    hold off;
    end
```