Image Stitching

Furkan IŞIKDOĞAN
Boğaziçi University Computer Engineering Department
www.isikdogan.com

I. COMPUTATION OF COMMON POINTS

There are various ways to find the common points on corresponding images such as using SIFT or SURF algorithm as a keypoint detector. However, in this work the common points are marked manually by using the graphical user interface. The points are converted to homogeneous coordinate representation, thus the function returns two 3xn matrices where n is the number of points. Then, the points are stored as a separate file.

readPoints.m

```
%uncomment to read points from gui
number_of_correspondences = 5;
[im1points im2points_l] = getPoints(I_left, I_middle, number_of_correspondences);
[im3points im2points_r] = getPoints(I_right, I_middle,
number_of_correspondences);
save points im1points im2points_l im3points im2points_r;

%uncomment to load points from file
% load points;
```

getPoints.m

```
function [ im1Points im2Points ] = getPoints( im1, im2, numOfPoints )

subplot(1,2,1);
imshow(im1/255);
[x1 y1] = ginput(numOfPoints);

subplot(1,2,2);
imshow(im2/255);
[x2 y2] = ginput(numOfPoints);

%points in homogeneous coordinates
im1Points = [x1 y1 repmat(1,numOfPoints,1)]';
im2Points = [x2 y2 repmat(1,numOfPoints,1)]';

close;

end
```

II. COMPUTATION OF HOMOGRAPHIES

After obtaining points we need to compute homographies that maps each X to x in order to perform perspective alignment.

$$X = Hx$$

We can solve H by the following equation:

$$Ah = 0$$

Where h is a vector with 9 elements and where A is a 3n x 9 matrix in the following form:

$x_i^T = [x_i, y_i, w_i]^T$

$\mathbf{0} = [0\ 0\ 0]$

$$A_i = \begin{bmatrix} 0 & -w_iX & y_iX \\ w_iX & 0 & -x_iX \\ -y_iX & x_iX & 0 \end{bmatrix}$$

Singular Value Decomposition (SVD) is used to solve the homogeneous equation $Ah = 0$. Then the vector h is shaped as a 3x3 matrix. The algorithm is implemented in MATLAB as follows:

```
function H = computeH(im1points,im2points)

      N = length(im1points);
      A = zeros(3*N,9);

      for n = 1:N
          X = im1points(:,n)';
          x = im2points(1,n);
          y = im2points(2,n);
          w = im2points(3,n);
          A(3*n-2,:) = [ zeros(1,3)  -w*X  y*X];
          A(3*n-1,:) = [ w*X   zeros(1,3)  -x*X];
          A(3*n  ,:) = [-y*X  x*X   zeros(1,3) ];
      end

      [U,S,V] = svd(A,0);

      H = reshape(V(:,9),3,3)';

end
```

III. TRANSFORMATION OF IMAGES (WARPING)

In order to generate a grid for the warped image we need to find the transformed coordinates of the corners. Corners can have negative values after transformation, thus we can not use just the rounded values of transformed coordinates. The grid is generated between the minimum and the maximum values of transformed corner coordinates with 1 pixel intervals.

If we perform mapping as *im2points = H x im1points*, we may end up with an image with empty pixels. Hence, it is more reasonable to do inverse mapping as $im1points = H^{-1} \times im2points$, and we can use bilinear interpolation to estimate the pixel values at floating point coordinates.

We need to keep the offsets which are the minimum values of transformed coordinates in order to shift images correctly while stitching images.

```
function [imWarped offset] = warpImg(inputImg, H)

[h w c] = size(inputImg);

%in order to generate a grid for the warped image we need to apply
%the transform to the corner points
cornerPts = H*[ 1 1 w w ; 1 h 1 h ; 1 1 1 1 ];

%normalize homogeneous coordinates
cornerPts = cornerPts./repmat(cornerPts(3,:),3,1);

%since the minimum distance that we have is 1 pixel, find the minimum
%and the maximum points for both axes and fit a grid with 1 pixel intervals
[X_grid,Y_grid] = ndgrid( min( cornerPts(1,:) ) : 1 : max( cornerPts(1,:) ),...
                  min( cornerPts(2,:) ) : 1 : max( cornerPts(2,:) ));
[numRow numCol] = size(X_grid);

% inverse mapping im1points = H^(-1) * im2points
im1points = H \ [ X_grid(:) Y_grid(:) ones(numRow*numCol,1) ]';
im1points = im1points./repmat(im1points(3,:),3,1); %normalize homogeneous
coordinates

xI = reshape( im1points(1,:),numRow,numCol)';
yI = reshape( im1points(2,:),numRow,numCol)';

%interpolate the point values for each color channel
imWarped(:,:,1) = interp2(inputImg(:,:,1), xI, yI);
imWarped(:,:,2) = interp2(inputImg(:,:,2), xI, yI);
imWarped(:,:,3) = interp2(inputImg(:,:,3), xI, yI);

%assign minimum corners as the offset of the image
cornerPts = round(cornerPts);
offset = [ min( cornerPts(1,:)) min( cornerPts(2,:) ) ];

end
```

IV. BLENDING IMAGES

Since we know the offsets of the warped images according to the referance image which is in the middle, we can align the offsets according to the origin point of the canvas. Then, we can add each image with its offset to a layer of result image. Then we can blend the layers by using the maximum intensity values in order to obtain a flatten image.

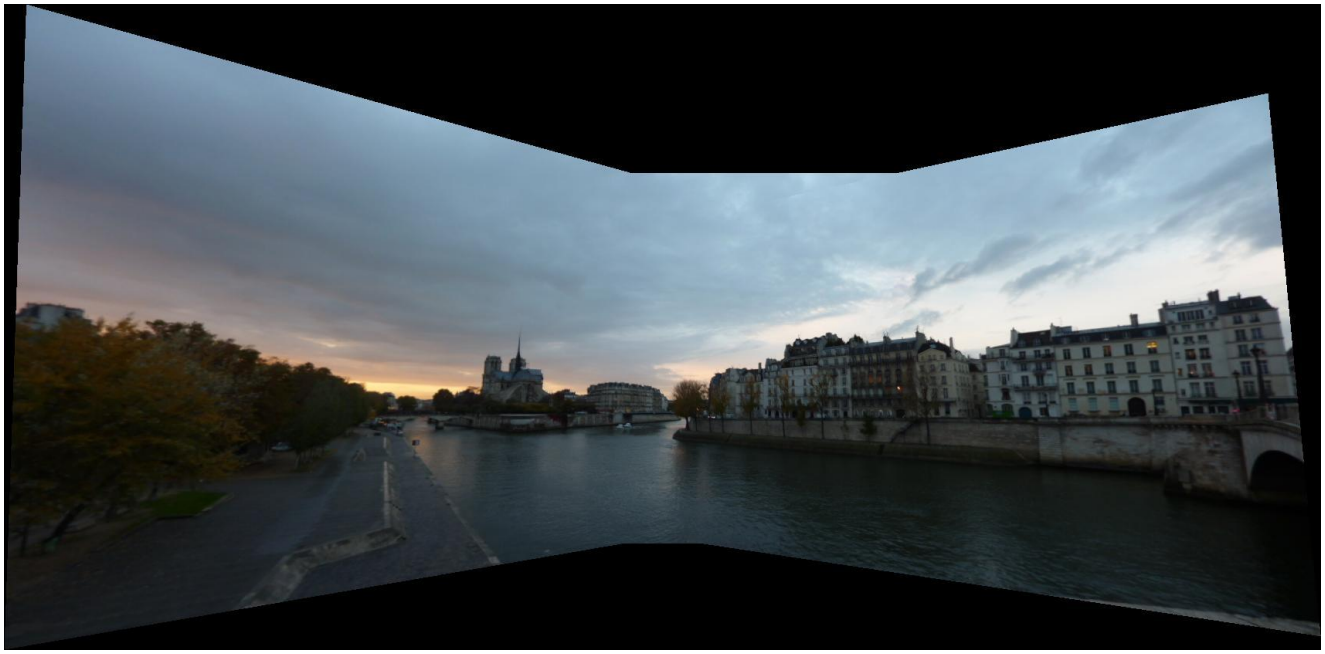
```
function [ panoramaImg ] = blendImages( I_left_warped, I_middle, I_right_warped,
offset_l, offset_r )

%alignment of offsets
offset_m = -offset_l;
min_Y = min(offset_l(2),offset_r(2));
offset_l(1) = 0;
offset_l(2) = offset_l(2) - min_Y;
offset_m(2) = offset_m(2) + offset_l(2);
offset_r(1) = offset_r(1) + offset_m(1);
offset_r(2) = offset_r(2) - min_Y;

%add left image
[hl wl c] = size(I_left_warped);
panoramaImg(1+offset_l(2):hl+offset_l(2),1+offset_l(1):wl+offset_l(1),:,1) =
I_left_warped(1:hl,1:wl,:);

%panoramaImg(1+offset_l(2):hl+offset_l(2), 1+offset_l(1):offset_m(1),:) =
I_left_warped(1:hl,1:offset_m(1),:);

%add middle image
[hm wm c] = size(I_middle);
panoramaImg(1+offset_m(2):hm+offset_m(2),1+offset_m(1):wm+offset_m(1),:,2) =
I_middle(1:hm,1:wm,:);

%add right image
[hr wr c] = size(I_right_warped);
panoramaImg(1+offset_r(2):hr+offset_r(2),1+offset_r(1):wr+offset_r(1),:,3) =
I_right_warped(1:hr,1:wr,:);

%merge the max intensity values of each channel
panoramaImg = max(panoramaImg,[],4);

end
```

V. EFFECT OF CORRESPONDENCES

Number of correspondences is one of the major factors that effects the robustness to noise of the program. If we only choose few points our system will more likely to be effected by noise. In the following example only five correspondence points are used for each image couples. Even though the points are selected very precisely, use of many points has resulted with a better stitched image.

Panorama Image with 5 correspondence points:

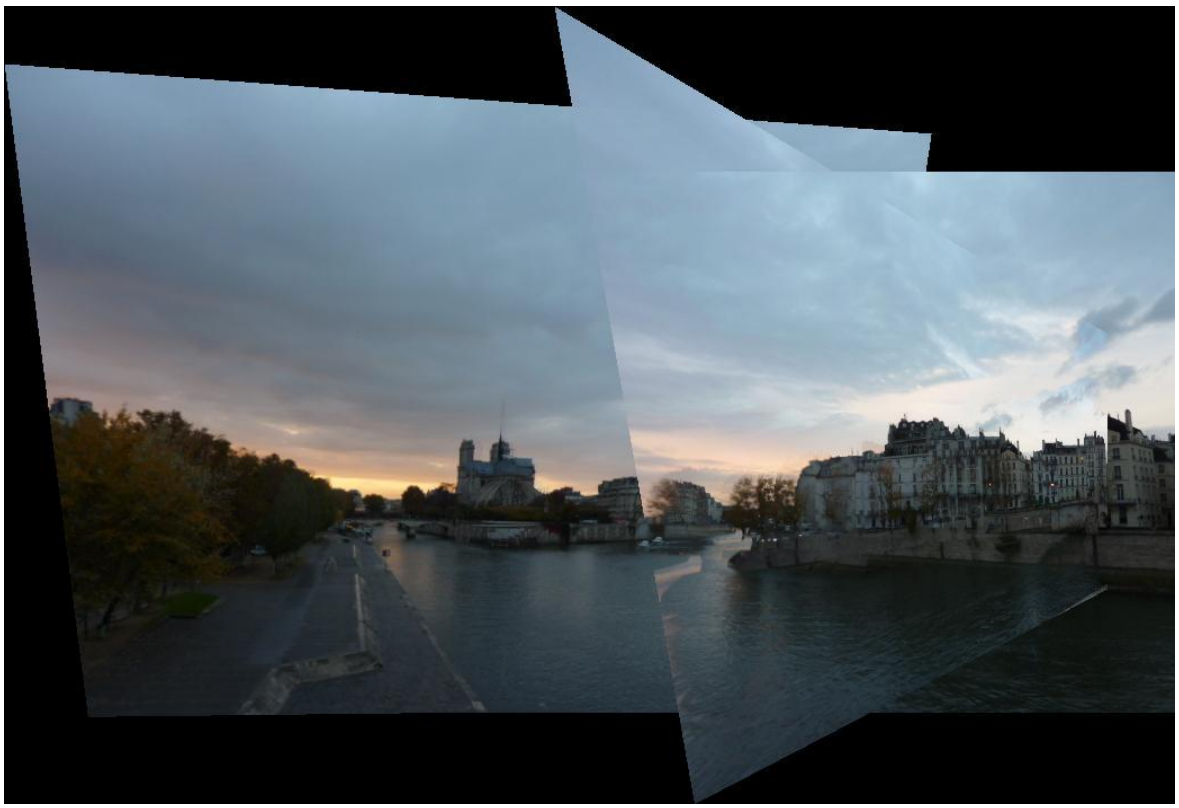
Panorama Image with 14 correspondence points:



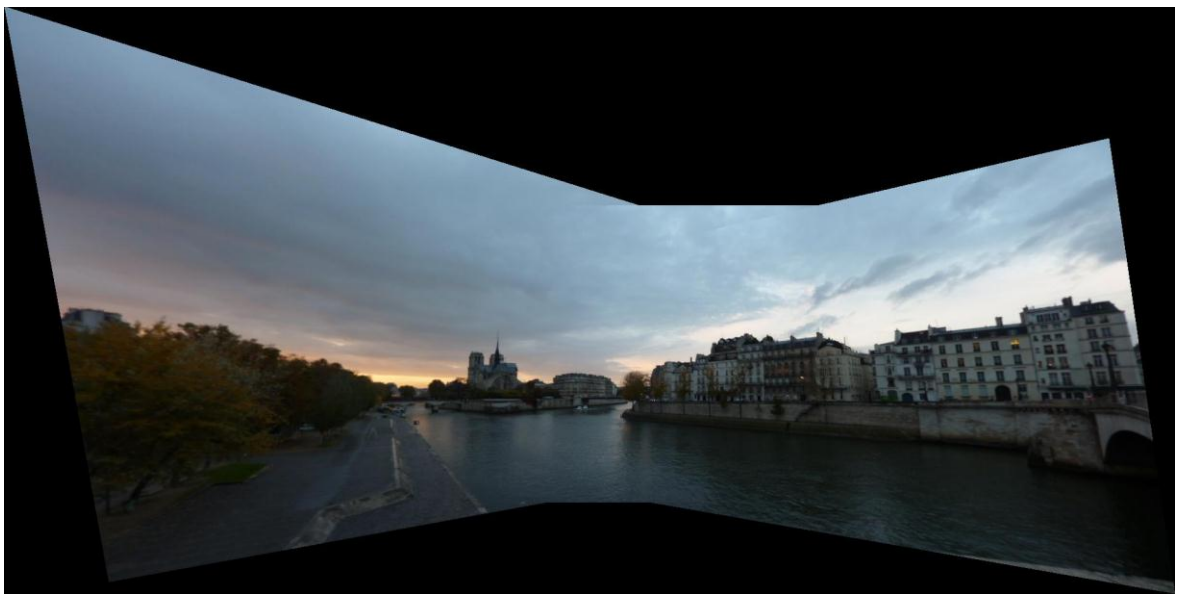
When we add some gaussian noise between [0,5] the effect of correspondences becomes more dramatical.

```
im1points(1:2,:)    = im1points(1:2,:)  + 5*rand(2,length(im1points));
im2points_l(1:2,:)  = im2points_l(1:2,:)+ 5*rand(2,length(im2points_l));
im2points_r(1:2,:)  = im2points_r(1:2,:)+ 5*rand(2,length(im2points_r));
im3points(1:2,:)    = im3points(1:2,:)  + 5*rand(2,length(im3points));
```

Panorama Image with 5 noisy correspondence points:



Panorama Image with 14 noisy correspondence points:



Even if we have many correspondences any wrong couple of points can deform the blended image dramatically. In the following example a couple of random correspondences added within the rectangle between [0,0] and [30,30]. If we have few points the warped images can be much more deformed which can not to be merged due to non-adjustable offsets.

```
im1points   = [im1points [rand(1,2)*30 1]'];
im2points_l = [im2points_l [rand(1,2)*30 1]'];
im2points_r = [im2points_r [rand(1,2)*30 1]'];
im3points   = [im3points [rand(1,2)*30 1]'];
```

Panorama Image with 14 correct and 1 wrong correspondence points:

Warped images with 5 correct and 1 wrong correspondence points (generally can not to be merged due to non-adjustable offsets):

The method that we used to calculate homographies is not independent from the origin and the scale of the coordinates of the correspondence points. In order to make our algorithm more stable and robust to noise we should normalize the points by shifting the mean to zero, and scaling lengths to make the average distance $\sqrt{2}$. We can calculate the transformation matrix that normalizes the points. After normalization our homography matrix will perform mapping between normalized points. We need to denormalize transformed points in order to obtain actual coordinates.

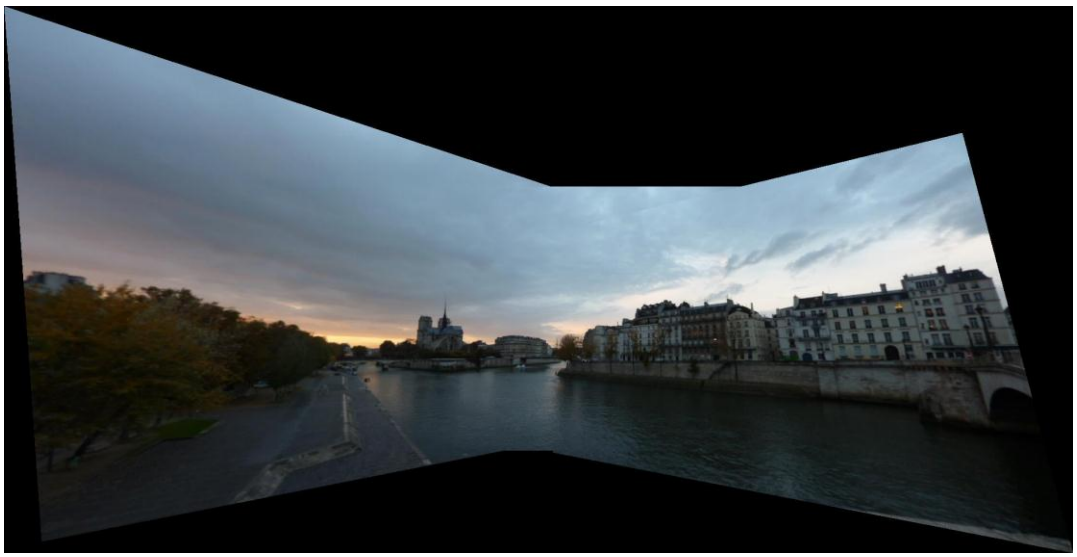
The MATLAB function that returns the transform matrix is imlemented as follows.

```
function [ NT ] = getNormalizationTransform( imPoints )

    imPoints = imPoints./repmat(imPoints(3,:),3,1); %h=1
    meanPts = mean(imPoints(1:2,:),2); %calculate mean
    imPoints = imPoints(1:2,:) - repmat(meanPts,1,length(imPoints)); %mean=0
    lengthPts = sqrt(imPoints(1,:).^2 + imPoints(2,:).^2); %find length
    avglength = mean(lengthPts); %get average length

    %build transformation matrix for normalization(shift and scale)
    NT = [sqrt(2)/avglength,  0,                    -sqrt(2)/avglength*meanPts(1);
                 0,           sqrt(2)/avglength,    -sqrt(2)/avglength*meanPts(2);
                 0,           0 ,                                1              ];
end
```

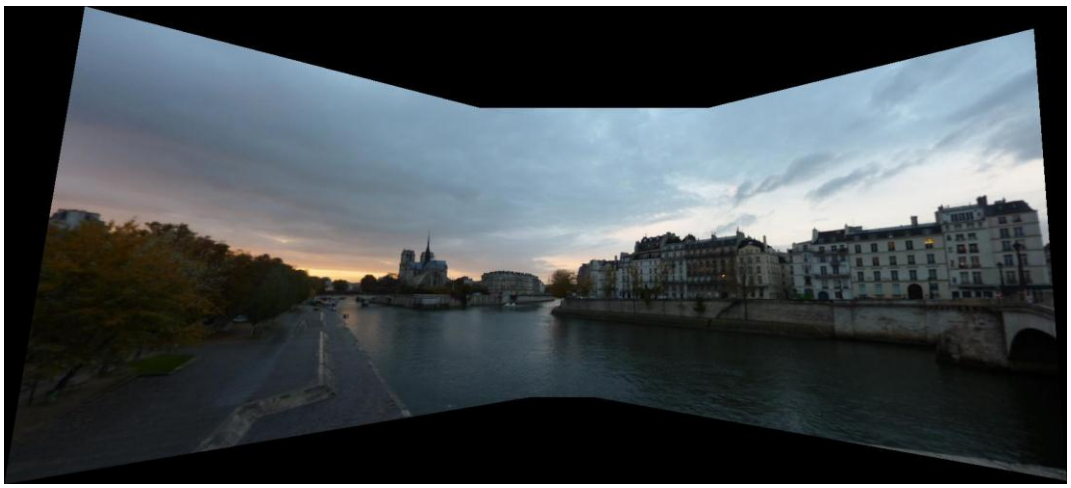
And the *computeH* function is modified as follows:

```
function H = computeH(im1points,im2points)

    %get normalization transform matrices
    NT1 = getNormalizationTransform(im1points);
    NT2 = getNormalizationTransform(im2points);

    %normalize points with 0 mean and sqrt(2) average length
    im1points = NT1*im1points;
    im2points = NT2*im2points;

    ...
    %the code that calculates homography matrix (see page 2)
    ...

    %denormalize homography
    H = NT2\H*NT1;

end
```

Panorama Image with 14 non-normalized noisy correspondence points:



Panorama Image with 14 normalized noisy correspondence points:



Finally, let us try our algorithm with another example. There are three pictures that I have taken at the Old Market Square, Poznan, Poland. The pictures have been taken with a low lens distortion camera by using a tripod.

Stitched image with 8 correspondence points: