Assignment 2 Solutions

CS283, Computer Vision

1. (a) We denote

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i, \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i. \tag{1}$$

It is easy to see that $x' = sx + t_x$ and $y' = sy + t_y$. From the requirement about the centroid, and using (1), we get

$$\sum_{i=1}^{N} x'_i = 0 \Rightarrow \sum_{i=1}^{N}(sx_i + t_x) = 0 \Rightarrow s\sum_{i=1}^{N} x_i + Nt_x = 0 \Rightarrow t_x = -s\bar{x}, \tag{2}$$

and similarly

$$t_y = -s\bar{y}. \tag{3}$$

From the average distance constraint, and using (2) and (3), we get

$$\frac{1}{N}\sum_{x=1}^{N}\sqrt{(sx_i + t_x)^2 + (sy_i + t_y)^2} = \sqrt{2} \Rightarrow$$

$$s = \frac{\sqrt{2}N}{\sum_{x=1}^{N}\sqrt{(x_i - \bar{x})^2 + (y_i - \bar{y})^2}}. \tag{4}$$

Replacing (4) into (2) and (3), we get the complete expressions for t_x and t_y.

(b) A possible implementation for this part is included below.

```
%%
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 2
% Question 1, part b
%
% Center and scale samples
%
function T = getT(X)

if ((¬ismatrix(X)) || (size(X, 2) ≠ 2) || (size(X, 1) == 0))
    error('Input matrix must be a Nx2 matrix.');
end;

X = transpose(X);
%% Calculate tx, ty and s

Xbar = mean(X(1, :));
Ybar = mean(X(2, :));

% We use hypot(x, y) instead of sqrt(x ^ 2 + y ^ 2) or norm([x, y]), as
% this function can be significantly faster and numerically more robust
% than the other two.
stdVal = mean(hypot(X(1, :) - Xbar, X(2, :) - Ybar));
if stdVal > 0,
    s = sqrt(2) / stdVal;
else
    s = 1;
    warning('All points coincide.');
end;

tx = - s * Xbar;
```

```
ty = - s * Ybar;

%% Form and return T
T = [s 0 tx;
     0 s ty;
     0 0 1];
```

(c) The resulting plot is shown in figure 1.

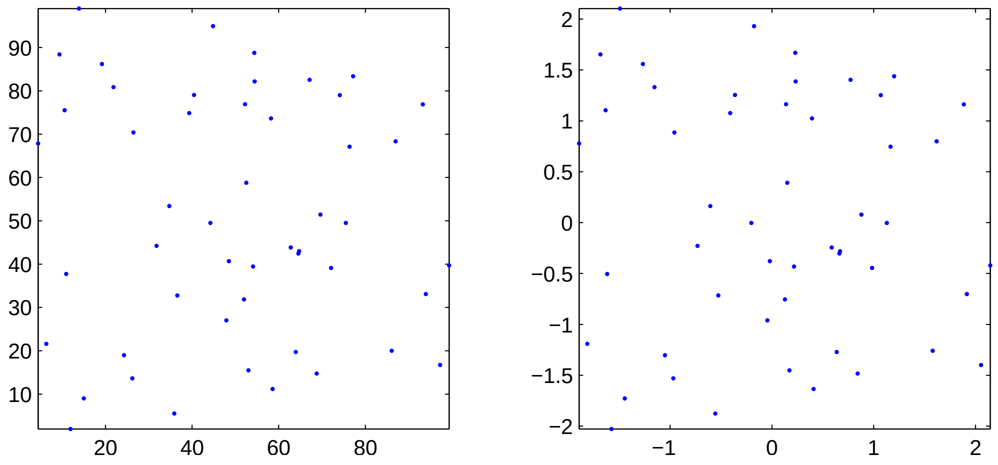

Figure 1: Resulting plots for script of question 3c.

Line 2 of the test script converts the inhomogeneous co-ordinates in the matrix X to their homogeneous versions by concatenating a 1 to each vector, and then applies the similarity matrix T to them. Line 3 then converts the homogeneous co-ordinates back to inhomogeneous ones, by dividing the first two coordinates with the third one.

2. (a) Each corresponding pair of points $x = [x_1, x_2, x_3]^T, y = [y_1, y_2, y_3]^T \in \mathbb{R}^3$, in homogeneous coordinates, is related by the homography H as $Hx \times y = 0$. If we define $H^T = [h_1, h_2, h_3]$ where $h_1, h_2, h_3 \in \mathbb{R}^3$ correspond to the rows of H, we can write

$$Hx = \begin{bmatrix} h_1^T x \\ h_2^T x \\ h_3^T x \end{bmatrix}, \tag{5}$$

and using this we can express the homography relationship as

$$Hx \times y = \begin{bmatrix} (h_2^T x)y_3 - (h_3^T x)y_2 \\ (h_3^T x)y_1 - (h_1^T x)y_3 \\ (h_1^T x)y_2 - (h_2^T x)y_1 \end{bmatrix} = 0. \tag{6}$$

From every corresponding pair of points, using (6) we get three equations for the elements of H, only two of which are linearly independent. If we select the first two equations, then they can equivalently be written in the form

$$\underbrace{\begin{bmatrix} 0\ 0\ 0 & y_3 x^T & -y_2 x^T \\ y_2 x^T & -y_1 x^T & 0\ 0\ 0 \end{bmatrix}}_{A} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = 0. \tag{7}$$

If we have multiple corresponding pairs of points, then for each of those we can form a matrix A

of the form of (7). If we stack up the matrices A for all pairs, we get

$$\underbrace{\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_N \end{bmatrix}}_{\Lambda} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = 0, \tag{8}$$

where Λ is a $2N \times 9$ matrix. Adding the constraint $\sum_{i,j} H_{ij}^2 = 1$ to avoid the trivial solution $H = 0$, we can obtain the least-squares solution of the above overcomplete linear system. To do so, we can use the Singular Value Decomposition (SVD) of Λ, as was discussed in class (see also section 3.1 of [1]).

As suggested in [2], we should firstly normalize the points x and y, and then calculate the homography between their respective images. If T_x and T_y are the similarity matrices for the two sets of points, and H' the homogaphy that sends $T_x x$ to $T_y y$, then overall transformation is given by $H = T_y^{-1} H' T_x$. As usual however, the inverse is used *only for notational purposes*, in order to express the solution in closed-form. It is almost never necessary to compute the inverse of a matrix. In practice, we can solve the above problem by using some appropriate factorization method, such as QR factorization. These methods offer significant advantages both in terms of computational efficiency and numerical stability. In Matlab, these can be easily employed using the backslash operator, as shown in the implementation in the following.

As discussed in the hints, in Matlab it is important to avoid loops whenever possible. In the implementation below, we do provide a version that uses no loops whatsoever. If you need to use a loop, there are several ways in which you can improve performance. Possibly the most important is to make sure that you have *preallocated* all memory you are going to use inside the loop. The difference in performance with and without preallocation can be dramatic. You can read more about this technique in Matlab's Techniques for Improving Performance[1], as well as in the Undocumented Matlab website[2]. In the following implementation, we also provide a loopy version of the code, where we take care to preallocate the matrix A before entering the for loop.

Below is a possible implementation of function `getH.m`, following the discussion above. It uses `getT.m` from question 4.

```
%%
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 2
% Question 2, part a
%
% Estimate a homography from corresponding pairs of points
%
function H = getH(X1, X2)

if ((¬ismatrix(X1)) || (¬ismatrix(X2)) || ¬isempty(find(size(X1) ≠ size(X2), 1))...
        || (size(X1, 2) ≠ 2) || (size(X1, 1) == 0))
    error('Input must be two Nx2 matrices.');
end;

N = size(X1, 1);

%% Normalize points using getT
T1 = getT(X1);
T2 = getT(X2);

X1h = T1 * in2hom(transpose(X1));
X2h = T2 * in2hom(transpose(X2));
```

[1]http://www.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html
[2]http://undocumentedmatlab.com/blog/preallocation-performance/

```

%% Construct matrix A of the homogeneous linear system
% 1) Loop−free version.
% A = [zeros(N, 3), −repmat(X2h(3, :)', [1 3]) .* X1h', repmat(X2h(2, :)', [1 ...
    3]) .* X1h';
%   repmat(X2h(3, :)', [1 3]) .* X1h', zeros(N, 3), −repmat(X2h(1, :)', [1 3]) ...
    .* X1h'];

% 2) Loopy version, two variants.
% Make sure to preallocate A.
A = zeros(2 * N, 9);
% A = zeros(3 * N, 9);  % for redundant version where A is 3*N x 9 matrix.
for iter = 1:N,
    A((iter * 2 − 1): (iter * 2), :) = ...
    [zeros(1, 3) −X2h(3, iter) * transpose(X1h(:, iter)) X2h(2, iter) * ...
        transpose(X1h(:, iter));
    X2h(3, iter) * transpose(X1h(:, iter)) zeros(1, 3) −X2h(1, iter) * ...
        transpose(X1h(:, iter));];
% % Alternatively, for redundant version where A is 3*N x 9 matrix.
%   A((iter * 3 − 2): (iter * 3), :) = ...
%   [zeros(1, 3) −X2h(3, iter) * transpose(X1h(:, iter)) X2h(2, iter) * ...
    transpose(X1h(:, iter));
%   X2h(3, iter) * transpose(X1h(:, iter)) zeros(1, 3) −X2h(1, iter) * ...
    transpose(X1h(:, iter));];
%   −X2h(2, iter) * transpose(X1h(:, iter))  X2h(1, iter) * transpose(X1h(:, ...
    iter)) zeros(1, 3)];
end;

%% Compute SVD and take last vector of V
[¬, ¬, V] = svd(A);
h = V(:, end);
Htild = transpose(reshape(h, [3 3]));

%% "Remove" normalizations
% We use the backslash operator (which employs QR decomposition) instead of
% calling inv or using H ^ (− 1).
H = T2 \ Htild * T1;


function Hcoords = in2hom(Icoords)

Hcoords = [Icoords; ones(1, size(Icoords, 2))];
```

(b) A possible implementation of function `applyH.m` is shown below.

```
%%
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 2
% Question 2, part b
%
% Apply a homography to an input image
%
function Iout = applyH(Iin, H)

% convert to double
Iin = im2double(Iin);

% find borders
w = size(Iin, 2);
h = size(Iin, 1);
corners = [1 1; 1 h; w 1; w h];
cornersH = transpose(H * transpose([corners ones(4, 1)]));
cornersH = cornersH(:, 1:2) ./ repmat(cornersH(:, 3), [1 2]);
xmin = min(cornersH(:, 1));
ymin = min(cornersH(:, 2));
xmax = max(cornersH(:, 1));
```

```
ymax = max(cornersH(:, 2));

% Set up resolution so that both images have approximately
% the same number of pixels.
xres = (xmax - xmin + 1);
yres = (ymax - ymin + 1);
zfactr = sqrt(w * h / xres / yres);
xres = ceil(xres * zfactr);
yres = ceil(yres * zfactr);

% Compute grid and corresponding co-ordinates
[x y] = meshgrid(linspace(xmin, xmax, xres), linspace(ymin, ymax, yres));
X2h = [x(:) y(:) ones(numel(x), 1)];
% We use the backslash operation (which employs QR decomposion) instead of
% calling inv or using H ^ (-1).
X1h = transpose(H \ transpose(X2h));
X1 = X1h(:, 1:2) ./ repmat(X1h(:, 3), [1 2]);

% Form image
numC = size(Iin,3);
Iout = zeros([size(x) numC]);
for i = 1:numC
    Iout(:, :, i) = reshape(interp2(Iin(:, :, i), X1(:, 1), X1(:, 2), 'linear'), ...
        size(x));
end;

Iout(isnan(Iout)) = 0;
```

(c) A rectification of the Maxwell-Dworkin picture using the code above is shown in figure 2.

3. Below is one possible solution for question 3. The function uses the coordinate system of the central image, and projects all other images to that through the use of homographies. When regions overlap, blending is done by averaging intensity estimates from all the images.

```
%%
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 2
% Question 3
%
% Stitch together images given a set of correspondences
%
function Iout = stitch(cimg, imgs, Hs)

numI = length(imgs);
cimg = im2double(cimg);

for i = 1:numI
    imgs{i} = im2double(imgs{i});
end;

% Compute full extents
w = size(cimg, 2);
h = size(cimg, 1);
extns = [1 1; 1 h; w 1; w h];

for i = 1:numI
    w = size(imgs{i}, 2);
    h = size(imgs{i}, 1);
    crns = [1 1; 1 h; w 1; w h];
    crnsH = transpose(Hs{i} * transpose([crns ones(4, 1)]));
    crns = crnsH(:, 1:2) ./ repmat(crnsH(:, 3), [1 2]);
    extns = [extns; crns];
end;

xmin = min(extns(:, 1));
```

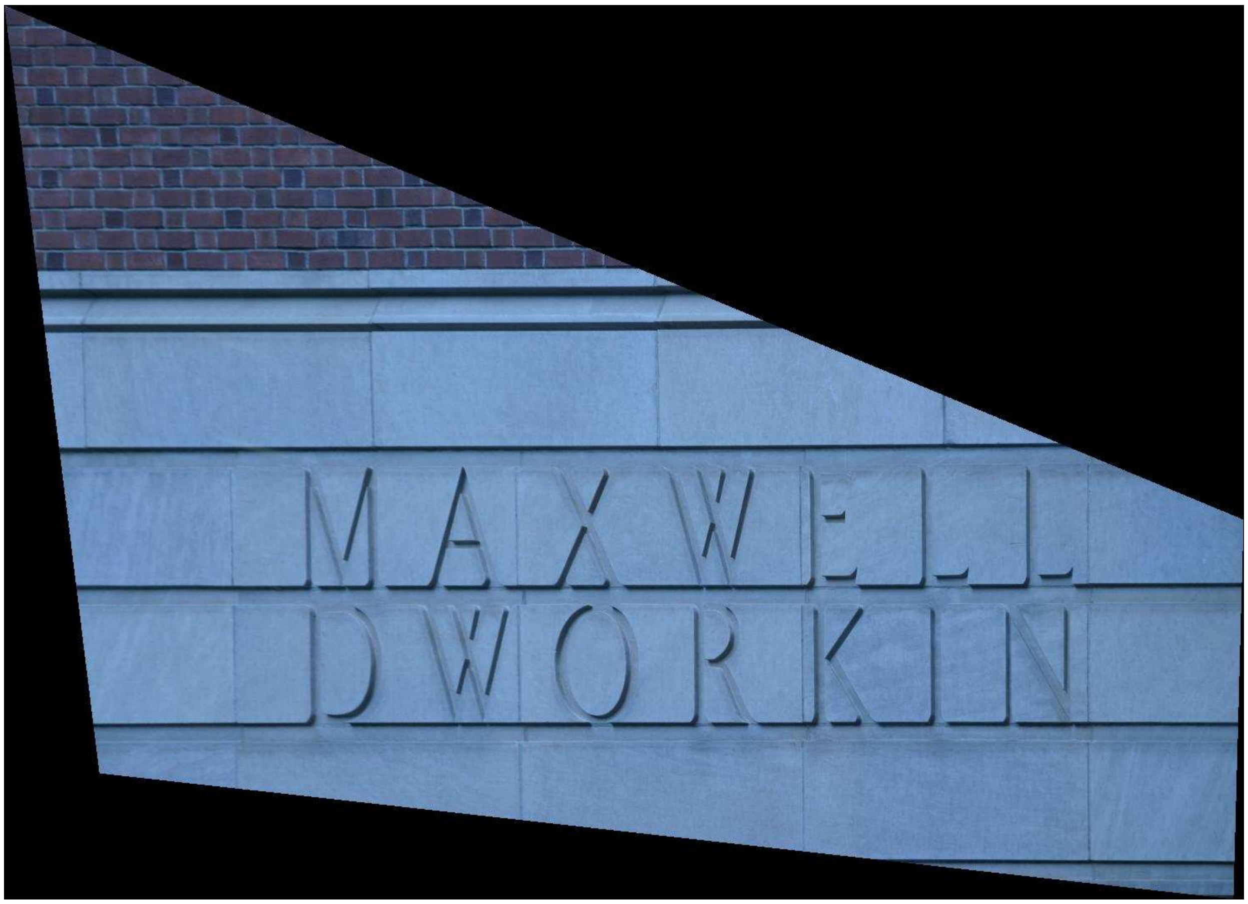

Figure 2: Rectification of the Maxwell-Dworkin picture.

```
xmax = max(extns(:, 1));
ymin = min(extns(:, 2));
ymax = max(extns(:, 2));

% Compute grid and corresponding co—ordinates
[x y] = meshgrid([floor(xmin):ceil(xmax)], [floor(ymin):ceil(ymax)]);
X2h = [x(:) y(:) ones(numel(x),1)];

% Form Image
numC = size(cimg, 3);
Iout = zeros([size(x) numC]);
blendmap = Iout;

% ——— From central image
for i = 1:numC
Iout(:,:,i) = reshape(interp2(cimg(:,:,i),x(:),y(:)),size(x));
5
end;
blendmap(¬isnan(Iout)) = 1;
Iout(isnan(Iout)) = 0;

% ——— From other images
for j = 1:numI
X1h = transpose(Hs{j} \ transpose(X2h));
X1 = X1h(:, 1:2) ./ repmat(X1h(:, 3), [1 2]);
temp_o = zeros([size(x) numC]);

for i = 1:numC
    temp_o(:, :, i) = reshape(interp2(imgs{j}(:, :, i), X1(:, 1), X1(:, 2), linear), ...
        size(x));
end;
blendmap(¬isnan(temp_o)) = blendmap(¬isnan(temp_o)) + 1;
temp_o(isnan(temp_o)) = 0;
Iout = Iout + temp_o;
end;
Iout = Iout ./ max(blendmap, 1);
```

4. An implementation of the robust version of `getH.m` that uses RANSAC is shown below.

```
%%
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 2
% Question 4
%
% Estimate a homography from correspondences found using RANSAC
%
function H = ransacGetH(X1, X2)

% threshold for inliers
thresh = 3;
N = size(X1, 1);
X1h = [X1 ones(N, 1)];
X2h = [X2 ones(N, 1)];
numi_o = 1;
meane_o = 0;
ilist_o = [];
i = 0;

num_samples = 10 ^ 15;
while i < num_samples

    % Estimate H from 4 random points
    idx = randperm(N, 4);
    H = getH(X1(idx, :), X2(idx, :));

    % Compute distance, number of inliers, etc
    xx2h = transpose(H * transpose(X1h));
    xx2 = xx2h(:, 1:2) ./ repmat(xx2h(:, 3), [1 2]);
    xx1h = transpose(H \ transpose(X2h));
    xx1 = xx1h(:, 1:2) ./ repmat(xx1h(:, 3), [1 2]);
    dists = sqrt(sum((X2 - xx2) .^ 2, 2)) + sqrt(sum((X1 - xx1) .^ 2, 2));
    ilist = find(dists < 2 * thresh);
    numi = length(ilist);
    meane = mean(dists(numi));

    % Update number of samples
    i = i + 1;
    num_samples = log(0.001) / log(1 - (numi / N) ^ 4);

    % Update if answer s optimal
    if numi < numi_o
        continue;
    end;

    if (numi == numi_o) && (meane > meane_o)
        continue;
    end;

    ilist_o = ilist;
    meane_o = meane;
    numi_o = numi;
end;

H = getH(X1(ilist, :), X2(ilist, :));
```

In figure 3, we show the image resulting from stitching the three images provided together, using the functions posted above.

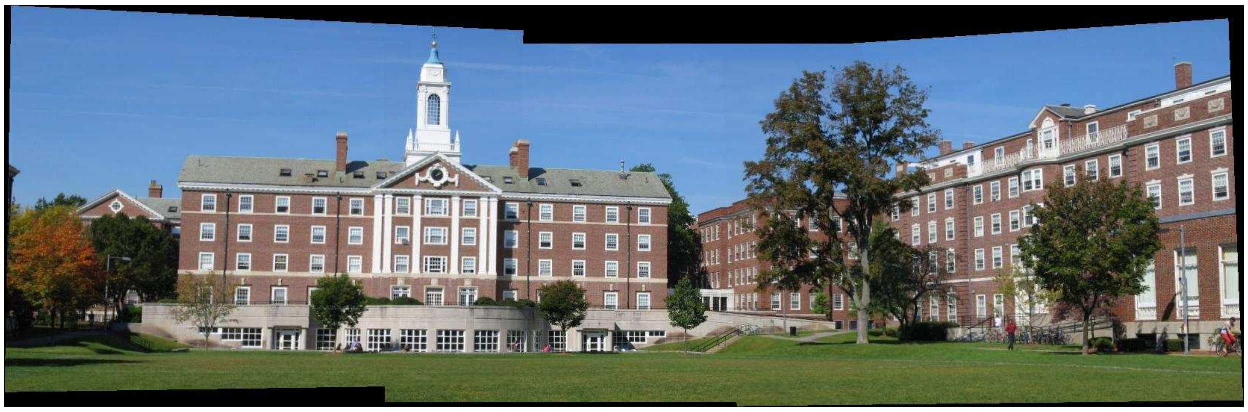

Figure 3: Stitched images as produced by the functions for questions 3 and 4.

References

[1] Forsyth, D., and Ponce, J. *Computer Vision: A modern approach.* Prentice Hall, 2003.

[2] Hartley, R., and Zisserman, A. *Multiple view geometry in computer vision.* Cambridge Univ Press, 2003.