Assignment 1

Computer Vision - IT-524

Milind Padalkar
ID: 201121015

22 August 2012

1. Perform image rotation at an arbitrary angle using bilinear interpolation

```
function rot_img = IT524_imrotate_fn(img,theta)
%% Function to perform image rotation for a given angle "theta" about the
% center of the image
%-------------------------------------------------------------------------
% Perform image rotation in 5 steps
% 1. Translate the image center to the origin
% 2. Rotate about the origin
% 3. Translate the origin to its original location
% 4. Transfer values to the newly calculated coordinates
% 5. Fill missing values using bilinear interpolation
%-------------------------------------------------------------------------
%   Inputs   :      1. Input image (img)
%                   2. Angle of rotation in degrees (theta)
%   Outputs  :      1. Rotated image (rot_img)
%   Usage    :      rot_img = mgp_rotate_fn(img,theta)
%% Function starts here

%% Set default parameters
if(exist('theta','var')==0)
    theta = 90;                  % Angle of rotataion in degrees
end
theta = mod(theta,360);

%% Read image size
[ht,wd,dp] = size(img);
if(dp>1)
    img = rgb2gray(img);
end

%% Combination of steps 1,2,3 gives us the following equation
% [x1;y1] = [cos(theta) sin(theta); -sin(theta) cos(theta)]*[x-x0;y-y0] + [x0;y0]
% x0 = ht/2, y0 = wd/2
x0 = round(ht/2);
y0 = round(wd/2);
Rot_mat = [cosd(theta) sind(theta); -sind(theta) cosd(theta)];

%% Here, we actually know (x1,y1) as they are the coordinates of the
```

```
% rotated image. Thus, we need to find (x,y) i.e. coordinates in the input
% image, from which (x1,y1) are calculated. Thus, the above equation needs
% to be modified as follows
% Let R = [cos(theta) sin(theta); -sin(theta) cos(theta)]
% Thus, [x1-x0;y1-y0] = R*[x-x0;y-y0]
% i.e. inv(R)*[x1-x0;y1-y0] = inv(R)*R*[x-x0;y-y0]
% i.e. inv(R)*[x1-x0;y1-y0] = [x-x0;y-y0]
% i.e. inv(R)*[x1-x0;y1-y0] + [x0;y0] = [x;y]
% i.e. [x;y] = inv(R)*[x1-x0;y1-y0] + [x0;y0]
% If (x,y) are integers, then the value can be directly copied from (x,y) to (x1,y1)
% else value at (x1,y1) can be interpolate using neighbours of (x,y)
invRot_mat = pinv(Rot_mat);

%% Generate a matrix of all coordinates such that newXY = [xa ya;xb yb;...;xn,yn]'
xrows = 1:ht;
ycols = 1:wd;

xRows = xrows(ones(wd,1),:);
xRows = reshape(xRows,1,numel(xRows));

yCols = ycols(ones(ht,1),:);
yCols = reshape(yCols',1,numel(yCols));

newXY = [xRows;yCols];

X0Y0 = [x0 y0];
X0Y0 = X0Y0(ones(size(newXY,2),1),:)';

%% Calculate the new coordinates
XY = invRot_mat*(newXY-X0Y0) + X0Y0;

%% Transfer values from old to new coordinates
rot_img = 0.*img;
for px = 1:size(newXY,2)
    x = XY(1,px);
    y = XY(2,px);
    x1 = newXY(1,px);
    y1 = newXY(2,px);
    % Check if the new coordinate is in the actual sized image
    if(x>0 && x<=ht && y>0 && y<=wd)
        % If exact location is available then copy
        if(mod(x*10,10)==0 && mod(y*10,10)==0)
            rot_img(x1,y1) = img(x,y);
        else
        % else calculate using bilinear interpolation
            rot_img(x1,y1) = mgp_bilinear_interpolate(img,x,y);
        end
    end
end

%% Show images
```

```
hdl_in = figure; imshow(img), title('Input Image');
hdl_out = figure; imshow(rot_img), title(sprintf('Image rotated by angle = %d degrees',theta));

%% Save the results
print(hdl_in,'-v','-dpng', sprintf('./1_input.png'));
print(hdl_out,'-v','-dpng', sprintf('./1_%d.png',theta));
close all;
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%% Function for demonstrating bilinear interpolation
% Date      : 01-08-2012
% Creator   : Milind Padalkar
%   Inputs  :   1. Input image
%               2. X coordinate at which value is to be determined
%               3. X coordinate at which value is to be determined
%   Assumptions : Both x and y are non-integers
%   Outputs :   1. Value at img(x,y)
%   Usage   :   px_val = mgp_bilinear_interpolate(img,x,y)

%% Starting here
function px_val = mgp_bilinear_interpolate(img,x,y)

%% Some theory
% Similar to linear interpolation
% |<---ty--->|
% P----------T1-----------Q
% |          T(x,y)       | tx
% |                       |
% R----------T2-----------S
%
% Using linear interpolation determine,
% 1. T1 between P and Q
% 2. T2 between R and S
% 3. T between T1 and T2 which is the final desired result

%% Implementataion

if(floor(x)>0 && floor(y)>0)
    p = img(floor(x),floor(y));
else
    p = min(img(:));
end

if(floor(x)>0 && ceil(y)<=size(img,2))
    q = img(floor(x),ceil(y));
else
    q = min(img(:));
end

if(ceil(x)<=size(img,1) && floor(y)>0)
```

```
    r = img(ceil(x),floor(y));
else
    r = min(img(:));
end

if(ceil(x)<=size(img,1) && ceil(y)<=size(img,2))
    s = img(ceil(x),ceil(y));
else
    s = min(img(:));
end

tx = x-floor(x);
ty = y-floor(y);

t1 = mgp_linear_interpolate(p,q,ty);
t2 = mgp_linear_interpolate(r,s,ty);
t = mgp_linear_interpolate(t1,t2,tx);

px_val = t;
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%% Function for demonstrating linear interpolation
% Date      : 01-08-2012
% Creator   : Milind Padalkar
%   Inputs  :   1. Starting value 'p' at point 'P'
%               2. Ending value 'q'at point 'Q'
%               3. Distance of 'R' from 'P'
%   Assumptions     :   Distance between p and q is 1
%   Outputs :   1. Value at 'R' i.e. r
%   Usage   :   r = mgp_linear_interpolate(p,q,ptor)

%% Starting here
function r = mgp_linear_interpolate(p,q,ptor)

%% Some theory
% Given points 'P' and 'Q' with values 'p' and 'q' at 'T' units apart, i.e.
% (q-p)/T = 1 unit
% we intend to find the value 'r' of a point 'R' between them (or anywhere on
% the line joining 'P' and 'Q') which is at 't' units from 'P'. Thus even
% (r-p)/t = 1 unit
% Hence, r = (((q-p)/T)*t)+p
T = 1;
t = ptor;
r = (((q-p)/T)*t)+p;
%--------------------------------------------------------------------------
```

Results

Input Image

Image rotated by angle = 30 degrees

Image rotated by angle = 45 degrees

Image rotated by angle = 135 degrees

Image rotated by angle = 290 degrees

Image rotated by angle = 350 degrees

Function for rotation using nearest-neighbour interpolation

```
function [out_img,A_mat] = sp_rot_nearest_hgs_fn(in_img,theta)
%% Program to rotate image using nearest-neighbour interpolation and a
% homogeneous equation i.e. Output = A*Input. Function to perform image
% rotation for a given angle "theta" about the center of the image.
% ************************************************************************
% **May take long time for large image size (even 256x256)*****************
% ************************************************************************
%------------------------------------------------------------------------
% Perform image rotation in the follwing steps
% 1. Translate the image center to the origin
% 2. Rotate about the origin
% 3. Translate the origin to its original location
% 4. Create the transformation matrix for Nearest-Neighbour Interpolation
% 5. vec(Output) = A*vec(Input)
% 5. Reshape vec(Output) to get the final output
%------------------------------------------------------------------------
%   Inputs   :      1. Input image (in_img)
%                   2. Angle of rotation in degrees (theta)
%   Outputs  :      1. Rotated image (out_img) in "double" format
%                   2. Transformation Matrix (A_mat)
%   Usage    :      [out_img,A_mat] = sp_rot_nearest_hgs_fn(in_img,theta)

%% Set default parameters
if(exist('theta','var')==0)
    theta = 90;                  % Angle of rotataion in degrees
end
theta = mod(theta,360);

%% Set image
[rows,cols,dp] = size(in_img);
if(dp>1)
    in_img = rgb2gray(in_img);
end
pd_ht = round(rows/2);
pd_wd = round(cols/2);

% Perform zeropadding
pd_img = zeros((rows+2*pd_ht),(cols+2*pd_wd));
pd_img(pd_ht+1:pd_ht+rows,pd_wd+1:pd_wd+cols) = in_img;

%% Combination of steps 1,2,3 gives us the following equation
% [x1;y1] = [cos(theta) sin(theta); -sin(theta) cos(theta)]*[x-x0;y-y0] + [x0;y0]
% x0 = ht/2, y0 = wd/2
x0 = round(rows/2);
y0 = round(cols/2);
Rot_mat = [cosd(theta) sind(theta); -sind(theta) cosd(theta)];

%% Here, we actually know (x1,y1) as they are the coordinates of the
% rotated image. Thus, we need to find (x,y) i.e. coordinates in the input
% image, from which (x1,y1) are calculated. Thus, the above equation needs
```

```
% to be modified as follows
% Let R = [cos(theta) sin(theta); -sin(theta) cos(theta)]
% Thus, [x1-x0;y1-y0] = R*[x-x0;y-y0]
% i.e. inv(R)*[x1-x0;y1-y0] = inv(R)*R*[x-x0;y-y0]
% i.e. inv(R)*[x1-x0;y1-y0] = [x-x0;y-y0]
% i.e. inv(R)*[x1-x0;y1-y0] + [x0;y0] = [x;y]
% i.e. [x;y] = inv(R)*[x1-x0;y1-y0] + [x0;y0]
% If (x,y) are integers, then the value can be directly copied from (x,y) to (x1,y1)
% else value at (x1,y1) can be interpolate using neighbours of (x,y)
invRot_mat = pinv(Rot_mat);

%% Generate a matrix of all coordinates such that newXY = [xa ya;xb yb;...;xn,yn]'
xrows = 1:rows;
ycols = 1:cols;

xRows = xrows(ones(cols,1),:);
xRows = reshape(xRows',1,numel(xRows));

yCols = ycols(ones(rows,1),:);
yCols = reshape(yCols,1,numel(yCols));

newXY = [xRows;yCols];

X0Y0 = [x0 y0];
X0Y0 = X0Y0(ones(size(newXY,2),1),:)';

%% Calculate the new coordinates
XY = invRot_mat*(newXY-X0Y0) + X0Y0;

%% Create the transformation matrix in the homogeneous form
A_mat = spalloc(numel(in_img),numel(pd_img),numel(in_img));
fprintf('Creating transformation matrix. Please be patient...');
for elm = 1:size(XY,2)
    rX = XY(1,elm);
    rY = XY(2,elm);
    A_mat(elm,:) = getNNrowVecSP(rX,rY,pd_img,pd_ht,pd_wd);
end
fprintf('Done!\n');

%% Reshape Input
pd_img = reshape(pd_img,numel(pd_img),1);

%% Perform Transformation
out_img = A_mat*pd_img;

%% Reshape output
out_img = reshape(out_img,rows,cols);

%% Display output
%% Display output
hdl_in = figure; imshow(in_img), title('Input Image');
```

```
print(hdl_in,'-v','-dpng', sprintf('./nn_input.png'));
hdl_out = figure; imshow(uint8(out_img)), title(sprintf('Rotated image at angle = %d degrees',theta));
print(hdl_out,'-v','-dpng', sprintf('./nn_%d.png',theta));
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
function cur_A_row = getNNrowVecSP(rX,rY,in_map,startX,startY)
%% Function to select the index of the nearest integral neighbour in a
% row-vector, corresponding to a given real (fractional) coordinate.
%   Inputs      :   1. X-coordinate (Real)
%                   2. Y-coordinate (Real)
%                   3. Large matrix from which index is to be selected
%                   4. Starting X-coordinate of the corresponding smaller
%                   matrix
%                   5. Starting Y-coordinate of the corresponding smaller
%                   matrix
%               Thus, the pixel (rX,rY) is located at in_map(startX+rX,startY,rY)
%   Outputs      :  1. A row vector having 1 at index of (rX,rY) in
%                      'in_map', with all other elements = 0
%   Usage        : cur_A_row = getNNrowVec(rX,rY,in_map,startX,startY)
%--------------------------------------------------------------------------

%% Get nearest low neighbour (i.e. Top Left)
low_intX = floor(rX);
low_intY = floor(rY);

%% Calculate distance of (rX,rY) from Top Left Neighbour
dx = rX - low_intX;
dy = rY - low_intY;

%% Get Neighbour-distance matrix such that [x,y,distance]
ngbr = zeros(4,3);
% Top left
ngbr(1,1) = low_intX;
ngbr(1,2) = low_intY;
% Top right
ngbr(2,1) = low_intX;
ngbr(2,2) = low_intY + 1;
% Bottom right
ngbr(3,1) = low_intX + 1;
ngbr(3,2) = low_intY + 1;
% Bottom left
ngbr(4,1) = low_intX + 1;
ngbr(4,2) = low_intY;

%% Calculate distances
ngbr(1,3) = sqrt(dx.^2 + dy.^2);
ngbr(2,3) = sqrt(dx.^2 + (1-dy).^2);
ngbr(3,3) = sqrt((1-dx).^2 + (1-dy).^2);
ngbr(4,3) = sqrt((1-dx).^2 + dy.^2);
```

```
%% Find nearest neighbour
nn = sortrows(ngbr,3);
nnx = nn(1,1);
nny = nn(1,2);

%% Select index of this neighbour
in_map = sparse(0.*in_map);
in_map((nnx+startX),(nny+startY)) = 1;
cur_A_row = reshape(in_map,1,numel(in_map));
```

Input Image

Rotated image at angle = 30 degrees