Image Understanding

Module 02 part 04: CS 472, CS 572

Image Transformations

Image Rotation

If camera is not aligned properly with the scene, the captured image will not be as desired. This sometimes needs a fix using an image editing software which allows the user to rotate the image in a desired angle. For example, if camera horizontal axis is not aligned with horizon while clicking a sunset photo. The image appeared to be imperfect.

What is captured

What is expected

Image Rotation

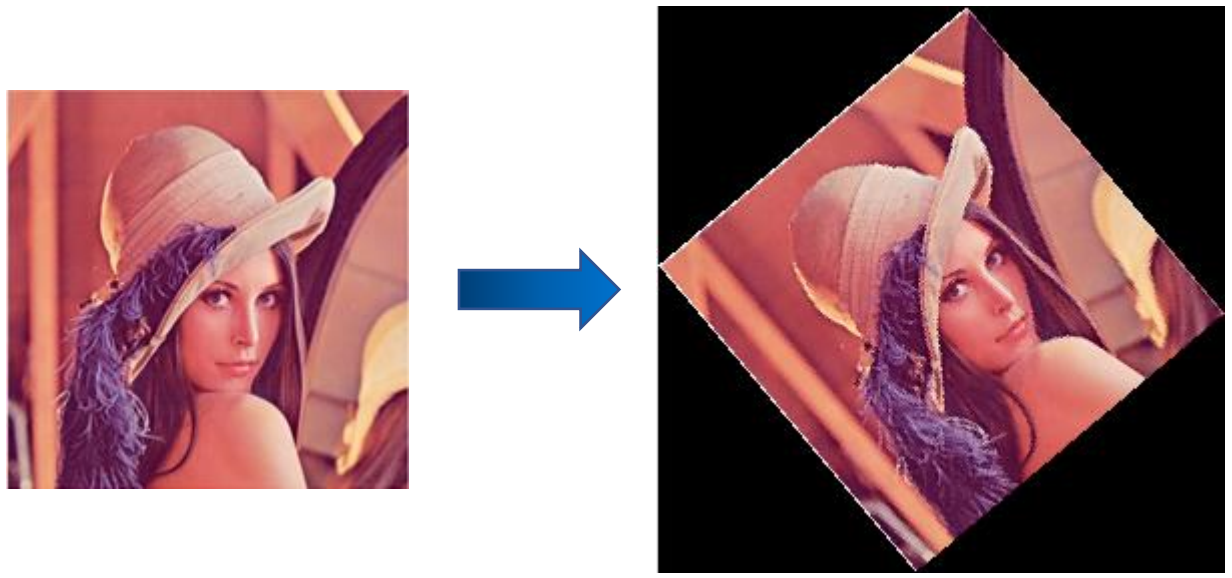
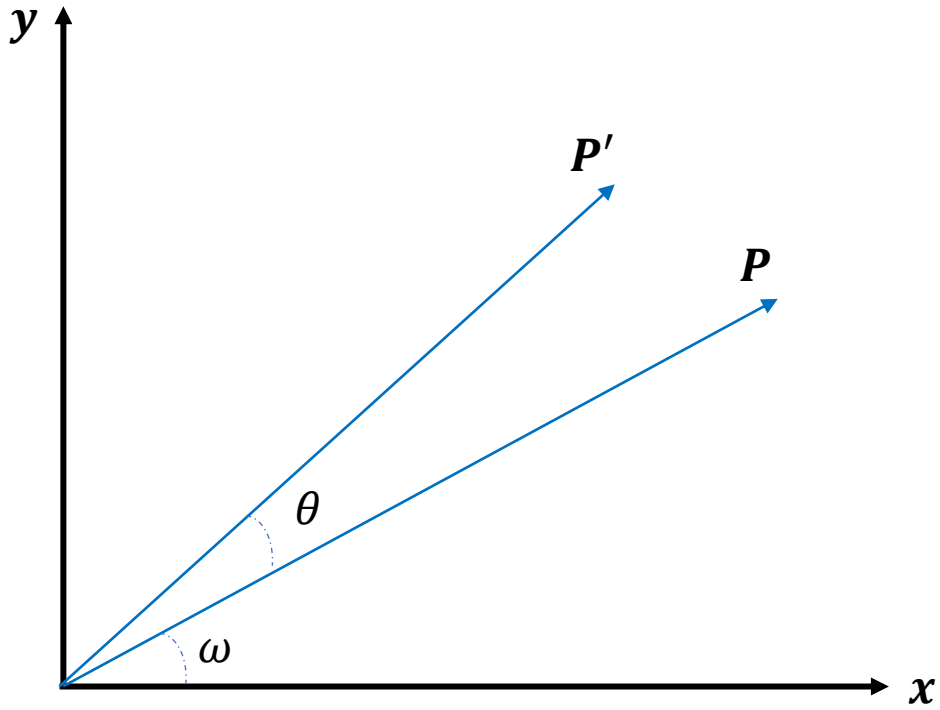


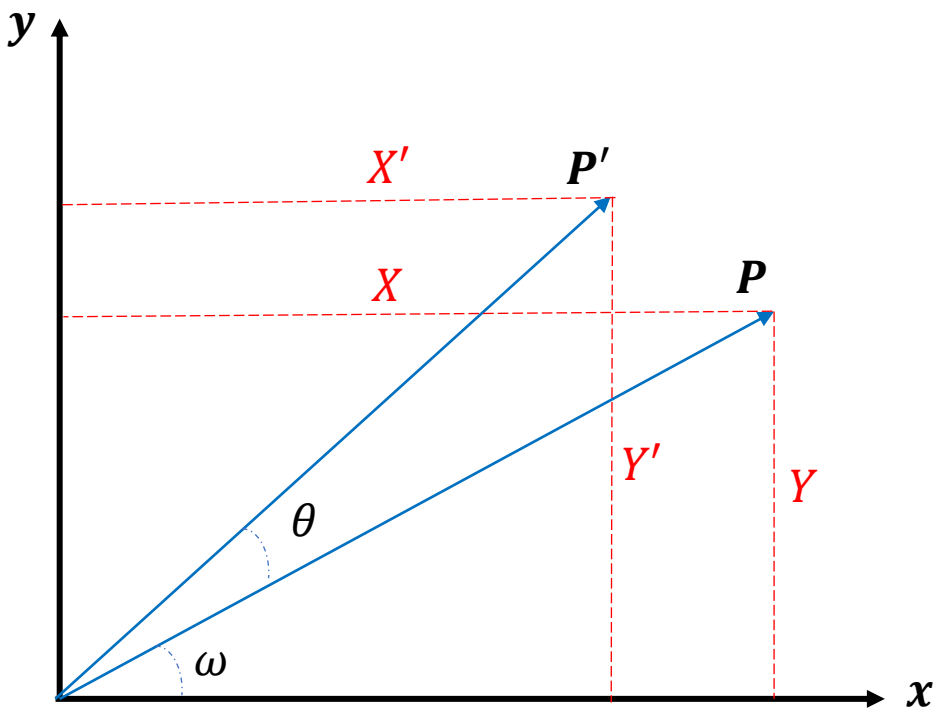
Image rotation



Rotation of a point

Image rotation can be achieved by rotating each pixel of an image with an angle θ. The plot in the right is showing a pixel $\mathbf{P}$ can be defined as a vector in the image coordinate system. The rotation of $\mathbf{P}$ with an angle θ results a different vector $\mathbf{P}'$.

Image Rotation



Rotation of a point

$$\boldsymbol{P} = \{X, Y\}, \quad \boldsymbol{P'} = \{X', Y'\},$$
$$X = \mathrm{R}\cos(\omega), \quad Y = \mathrm{R}\sin(\omega)$$

$$X' = \mathrm{R}\cos(\theta + \omega) = \mathrm{R}\cos(\theta)cos(\omega) - \mathrm{R}\sin(\theta)sin(\omega)$$
$$Y' = \mathrm{R}\sin(\theta + \omega) = \mathrm{R}\sin(\theta)cos(\omega) + \mathrm{R}\cos(\theta)sin(\omega)$$

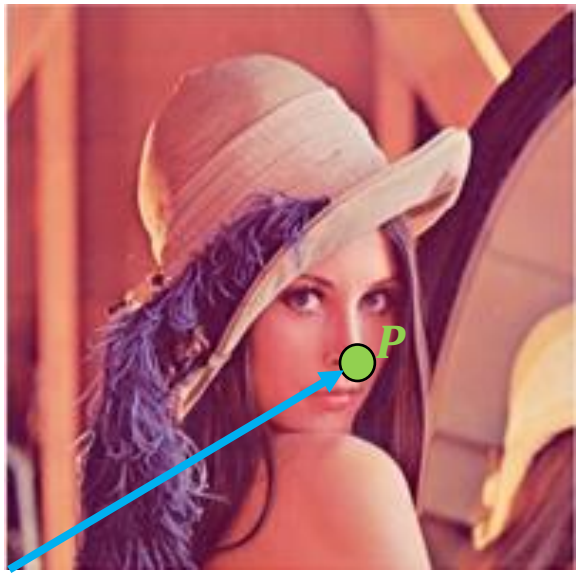
$$X' = \mathrm{X}\cos(\theta) - \mathrm{Y}\sin(\theta)$$
$$Y' = \mathrm{X}\sin(\theta) + \mathrm{Y}\cos(\theta)$$

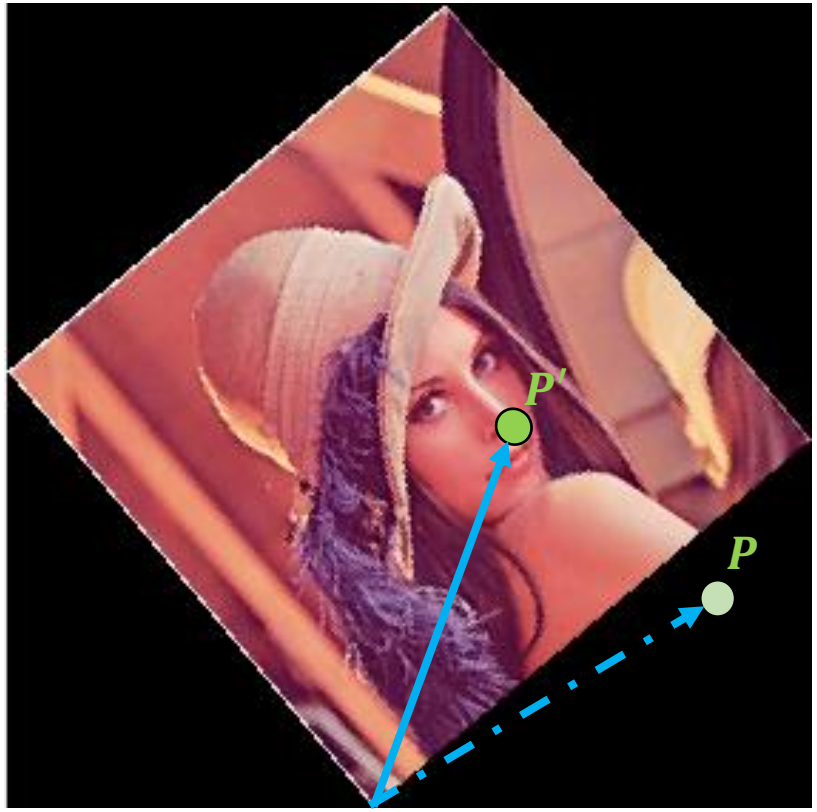
$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}$$

Matrix form of the equation

The equations in the right is used to define **P'** with respect to **P**. Where **P'** is rotation of vector **P** by an angle θ in the counterclockwise direction. The figure in the left illustrates the same.

Image Rotation





$$\boldsymbol{P} = \{X, Y\}, \quad \boldsymbol{P'} = \{X', Y'\},$$

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} \quad eqn.\,1$$

Rotate each pixel of an image by an angel θ

These are integers

These might not be integers

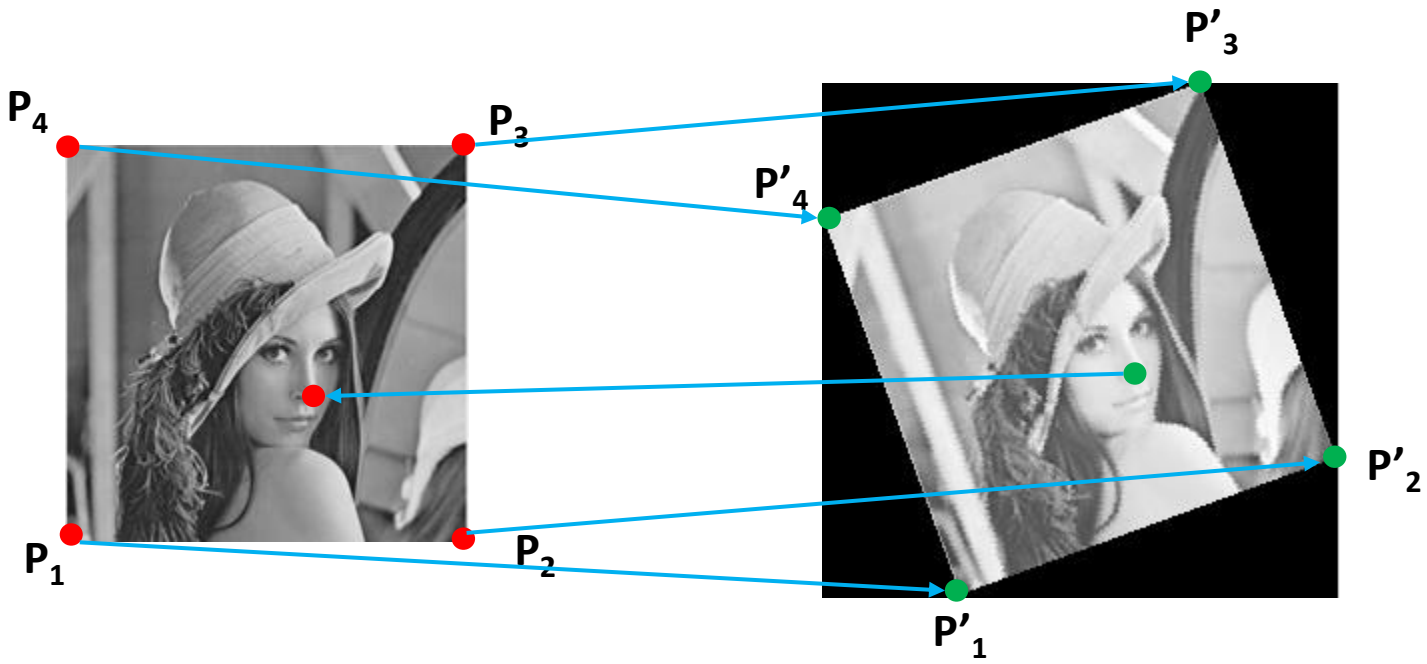
But to generate a new image, we want *X'*, *Y'* to be integers

The problem with the eqn. 1 is $X' and\ Y'$ might not be integers but pixel indexes are always integers. Therefore, $X' and\ Y'$ are needed to be converted to the closest integers using round() function. This sometimes degrades the quality of the rotated image.

Image Rotation

Solution! Inverse matrix multiplication

$$\begin{bmatrix} X \\ Y \end{bmatrix} = inv(\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}) \begin{bmatrix} X' \\ Y' \end{bmatrix}$$



Algorithm:

Calculate border pixel of rotated image

For each pixel in the rotated image, calculate location in the original image

Interpolate pixel value from neighbor pixels in the original image

The solution to the problem discussed before is use a inverse rotation matrix to calculate $Xand\ Y$ for any pair of integer $X'and\ Y'$. Therefore, even if $Xand\ Y$ are not integers, the value at $Xand\ Y$ location can be calculated using interpolation of nearby pixel locations. The equation above shows the inverse matrix multiplication to calculate $Xand\ Y$. The algorithm in the right is used to perform image rotation.

Image Rotation: Code

```
%Load image and display
img = imread('Lena_color.png');
figure(1), imshow(img);

%RGB tl gray scale conversion
imgGray = rgb2gray(img);
figure(2), imshow(imgGray);

%Rotation matrix based on angle
theta = 20;
rotMat = [cos( (pi*theta)/180 ), -sin( (pi*theta)/180 ); ...
    sin( (pi*theta)/180 ), cos( (pi*theta)/180 )];

%****** Compute the size of the rotated image *******
[row col] = size(imgGray);
P1 = [0; 0];
P2 = [col; 0];
P3 = [col; row];
P4 = [0; row];

P1_rot = rotMat * P1;
P2_rot = rotMat * P2;
P3_rot = rotMat * P3;
P4_rot = rotMat * P4;

x_min = round( min([P1_rot(1), P2_rot(1), P3_rot(1), P4_rot(1)]) );
x_max = round( max([P1_rot(1), P2_rot(1), P3_rot(1), P4_rot(1)]) );
y_min = round( min([P1_rot(2), P2_rot(2), P3_rot(2), P4_rot(2)]) );
y_max = round( max([P1_rot(2), P2_rot(2), P3_rot(2), P4_rot(2)]) );

row_rot = y_max - y_min + 1;
col_rot = x_max - x_min + 1;
%*********************************
```

Image Rotation: Code

```
%****************************************
%For each pixel of the rotated image compute
%the mapping pixel in the original image

img_rot = zeros(row_rot, col_rot);
for r = 1:row_rot
    for c = 1:col_rot
        P = rotMat\[c+x_min-1; r+y_min-1];  %Inverse matrix multiplication
        X = P(1); Y = P(2);

        %Pixel out of boundary of original image populated with zero
        if(X >= col || X <= 1 || Y >= row || Y <= 1)
            img_rot(row_rot - r + 1, c) = 0;
        else
            V1 = img(row - ceil(Y) + 1, ceil(X));
            V2 = img(row - ceil(Y) + 1, floor(X));
            V3 = img(row - floor(Y) + 1, ceil(X));
            V4 = img(row - floor(Y) + 1, floor(X));
            img_rot(row_rot - r + 1, c) = round( (double(V1) + double(V2) + double(V3) + …
                                                  double(V4))/4 );
        end
    end
end
%**********************************

figure(3), imshow(uint8(img_rot))
```