Assignment 2

CAP 6419 - 3D Computer Vision

Abhianshu Singla
Dept. of Computer Science

Affine and Metric Rectification of an image

The image is first rectified affinely and then rectified metrically in two different steps. Rectification is accomplished based on the invariants in that particular transformation.
Projective Transformation can be decomposed as:

$$H = H_s H_A H_P = \begin{bmatrix} sR & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} k & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} I & 0 \\ \mathbb{v}^T & v \end{bmatrix} = \begin{bmatrix} A & t \\ \mathbb{v}^T & v \end{bmatrix}$$

1. Affine Rectification

In Affine transformation, the line at infinity remains at infinity. Hence, parallel lines are still parallel after affine transformation. Another useful invariant under affine transformation is the distance between ratios of lengths on parallel lines. However, in projective transformation, line at infinity is no longer at $[0\ 0\ 1]^T$. The line at infinity can be computed by intersecting imaged parallel lines. If we map the line at infinity back to $[0\ 0\ 1]^T$, then we can obtain the affinely rectified image. The image is rectified as below if it is now $[l_1\ l_2\ l_3]^T$:

$$H = H_A \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{pmatrix}$$

where H_A is affine transformation

Verification:
We can verify that $l_\infty = [0\ 0\ 1]^T$ will map back to $[l_1\ l_2\ l_3]^T$ after applying H^{-T}.

$$H^{-T} l_\infty = \begin{pmatrix} 1 & 0 & l_1 \\ 0 & 1 & l_2 \\ 0 & 0 & l_3 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = [l_1\ l_2\ l_3]^T$$

Algorithm:

1. Mark 4 points $m_i = [x_i\ y_i\ 1]^T$ such that there are two pairs of interesting parallel lines such that $l_1 = m_1 \times m_2$ and $l_2 = m_3 \times m_4$. Similarly, $l_3 = m_1 \times m_3$ and $l_4 = m_2 \times m_4$. Parallel lines: $l_1 \parallel l_3$ and $l_2 \parallel l_4$. For marking the points, I used MATLAB getline function as it returns the end point of the line. Using getline function is better than using getpoints function because a line is drawn between the adjacent points which gives a visual feeling of selecting correct points.
2. Determine the Line at infinity with the help of above four parallel lines. Points at infinity are:

$$m_{1\infty} = (m_1 \times m_2) \times (m_3 \times m_4)$$
$$m_{2\infty} = (m_1 \times m_3) \times (m_2 \times m_4)$$
$$l_\infty = (m_{1\infty} \times m_{2\infty})$$

3. Suppose l_∞ is $[l_1\ l_2\ l_3]^T$. Make matrix H such that:

$$H = H_A \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1/l_3 & l_2/l_3 & 1 \end{pmatrix}$$

4. Apply matrix H to the image such that $I_A = H.I$ where I is the original image and I_A is the affinely rectified image. I used the imtransform function to transform the image to affinely rectified image. I also used maketform to get the tform for H_A.

Matlab Code:

```
function H_p = affine(Input,Output)
% Read the image
I = imread(Input);
imshow(I)

% Get 4 points such that they form 2 pairs of parallel lines
[xi,yi] = getline;

m1 = [xi(1) yi(1) 1];
m2 = [xi(2) yi(2) 1];
m3 = [xi(3) yi(3) 1];
m4 = [xi(4) yi(4) 1];

% Obtain the position of vanishing line
p_infinity1 = cross(cross(m1,m2),cross(m3,m4));
p_infinity2 = cross(cross(m1,m4),cross(m2,m3));
l_inf = cross(p_infinity1,p_infinity2);

% Affine Rectification
H_p = [ 1 0 0; 0 1 0; l_inf(1)/l_inf(3) l_inf(2)/l_inf(3) 1];

temp = maketform('projective', transpose(H_p));
I_a = imtransform(I,temp);

% Show and Save the image
myout = imshow(I_a);
saveas(myout,Output);
return ;
```

2. Metric Rectification

In Similarity transformation, ratios of line segments and angles remain invariant. Also, circular points (complex conjugate points I and J) is also fixed in metric/similarity transformation. So, metric rectified image can be obtained if we map the circular points back to their canonical positions.

$$\mathrm{I} = \begin{pmatrix} 1 \\ \mathrm{i} \\ 0 \end{pmatrix} \text{ and } \mathrm{J} = \begin{pmatrix} 1 \\ -\mathrm{i} \\ 0 \end{pmatrix}$$

Suppose Conic dual to circular points is:

$$C_\infty^* = IJ^T + JI^T$$

$$C_\infty^* = \begin{pmatrix}1\\ i\\ 0\end{pmatrix}(1 \quad -i \quad 0) + \begin{pmatrix}1\\ -i\\ 0\end{pmatrix}(1 \quad i \quad 0) = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}$$

Angles on Projective Planes:

In Euclidean space, the angle between two lines can be determined by:

$$cos\theta = \frac{l_1 m_1 + l_2 m_2}{\sqrt{({l_1}^2 + {l_2}^2)\,({m_1}^2 + {m_2}^2)}}$$

Similarly, in Projective Space, the angle between two lines $l = [l_1\ l_2\ l_3]^T\ and\ m = [m_1\ m_2\ m_3]^T$ can be determined by:

$$cos\theta = \frac{l^T C_\infty^* m}{\sqrt{(l^T C_\infty^* l)\,(m^T C_\infty^* m)}}$$

If lines $l\ and\ m$ are orthogonal to each other in world space, then $\theta = 90°\ and\ cos\theta = 0$.

$$l_1 m_1 + l_2 m_2 = l^T C_\infty^* m = 0$$

Suppose lines $l'and\ m'$ in the affinely rectified image correspond to the orthogonal lines $l\ and\ m$ in the actual world plane. Also, dual conic is:

$$C_\infty^{*\prime} = \begin{bmatrix}S & \mathbf{0}\\ \mathbf{0} & 0\end{bmatrix}$$

where S = KK^T is a symmetric matrix with 3 independent variables.

$$S = \begin{bmatrix}a & b\\ b & c\end{bmatrix} = \begin{bmatrix}a/c & b/c\\ b/c & 1\end{bmatrix}$$

As scaling is not important, so we need two linear equations to solve S (2 DOF).

$$l'^T C_\infty^{*\prime} m' = 0$$

$$(l_1' \quad l_2' \quad l_3')\begin{bmatrix}S & \mathbf{0}\\ \mathbf{0} & 0\end{bmatrix}\begin{pmatrix}m_1'\\ m_2'\\ m_3'\end{pmatrix} = 0$$

$$(l_1'm_1', l_1'm_2' + l_2'm_1', l_2'm_2')S = 0$$

These two pairs of orthogonal lines should not be parallel because it will be equivalent and thus results in only 1 linear equation.

Algorithm:

1. Mark 8 points $p_i = [x_i\ y_i\ 1]^T$ such that there are two pairs of orthogonal lines such that $l_1 = p_1 \times p_2$ and $m_2 = p_3 \times p_4$. Similarly, $l_1 = p_5 \times p_6$ and $m_2 = p_7 \times p_8$. $l_1 \perp m_1\ and\ l_2 \perp m_2$
2. Solve linear equation AS = B where $A = \begin{bmatrix}l_{11}'m_{11}' & l_{11}'m_{12}' + l_{12}'m_{11}'\\ l_{21}'m_{21}' & l_{21}'m_{22}' + l_{22}'m_{21}'\end{bmatrix}$ and $B = \begin{bmatrix}l_{12}'m_{12}'\\ l_{22}'m_{22}'\end{bmatrix}$. l_{ij}' represents j^{th} coordinate of line l_i'.
3. Solve $S = KK^T$ using singular value decomposition (svd).
4. After getting K, form matrix H.
$$H = \begin{pmatrix}k_{11} & k_{12} & 0\\ k_{21} & k_{22} & 0\\ 0 & 0 & 1\end{pmatrix}$$
5. Apply matrix H to the affinely rectified image.

Matlab Code:

```
function H_a = metric(Input1, Input2, Output, H_p)

% Read the image
I = imread(Input2);
imshow(I);

% Get 8 points such that they form 2 pairs of orthogonal lines
[x,y] = getline;

l1 = cross([x(1) y(1) 1],[x(2) y(2) 1]);
m1 = cross([x(3) y(3) 1],[x(4) y(4) 1]);
l2 = cross([x(5) y(5) 1],[x(6) y(6) 1]);
m2 = cross([x(7) y(7) 1],[x(8) y(8) 1]);

% Solve Ax_ = B
A = [l1(1)*m1(1) l1(1)*m1(2) + l1(2)*m1(1); l2(1)*m2(1) l2(1)*m2(2) +
l2(2)*m2(1)];
B = [-l1(2)*m1(2); -l2(2)*m2(2)];
x_ = linsolve(A,B);
S = [x_(1) x_(2); x_(2) 1];

% Get the value of K
% KK' = S
% K = U * sqrt(D) * U'
% V = U'
[U,D,V] = svd(S);
K = U * sqrt(D) * V;

% Form matrix H_a
H_a = transpose([K(1,1) K(1,2) 0; K(2,1) K(2,2) 0; 0 0 1]);
if H_a(1,1) < 0
    H_a(1,1) = -H_a(1,1);

elseif H_a(2,2) < 0
    H_a(2,2) = -H_a(2,2);
end

% Metric Rectification
H = H_p * H_a
temp = maketform('projective',transpose(H))
I_e = imtransform(Input1, temp);

% Show and Save the image
myout = imshow(I_e);
saveas(myout,Output);
return;
```

Problems:

1. I didn't know initially how to use imtransform and maketform functions work. So, working with them was challenging for me.
2. Using imtransform(imwarp) two times results in two frames in a single image, so as to fit the rectified image back in the original image size. So, I used imtransform on the original image with $H = H_a H_p$ to get only one frame.

Conclusion:

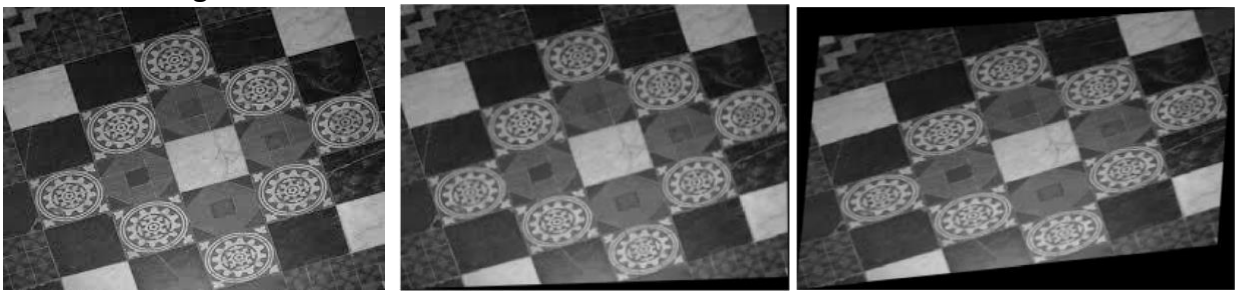
1. In projective transformation, line at infinity (vanishing line) is mapped to a fixed line. So, parallel lines can be visualized to be converging at some fixed point.
2. After affine rectification, parallel lines become parallel and visually doesn't seem to converge at a point other than infinity.
3. In affine transformation, circular points also change their location from canonical location. So, perpendicular lines don't seem to be perpendicular anymore.
4. After metric rectification, circular points map to their canonical positions and angle between the lines is rectified.

Discussion:

1. There is another method to get the metric rectified image in one step using dual conic $C_\infty^{*\prime}$. However, to solve it directly, we need 5 pairs of orthogonal lines which can be a daunting experience to get all five pairs correctly. So, we used the method in which the process is stratified into two steps.
2. Selecting the points highly impacts the rectification. If the image consists only one plane, i.e., only floor or only one side of wall, then it is easy to select points and the results are also promising. However, if the image consists of more than one plane like 2 sides of wall and a floor, then choosing the points in different planes results in very different rectified image.
3. Sometimes, different points selection results in splitting the image or changing the direction of image.
4. Image Dimension also changes after affine and metric rectification to adjust the correct geometric values.

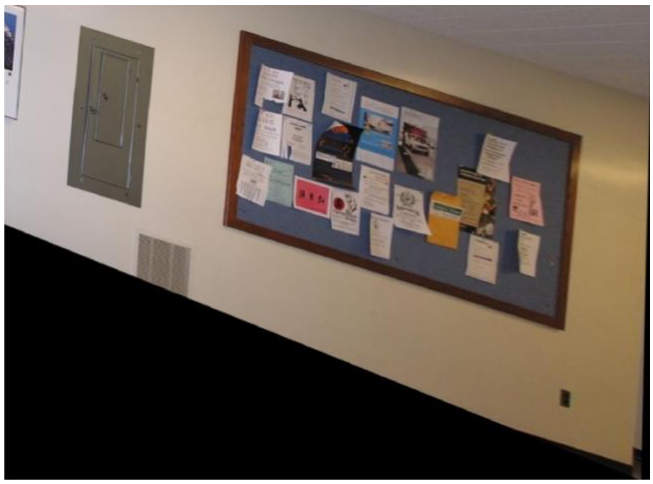
Results:

Image	Affine Rectification	Metric Rectification

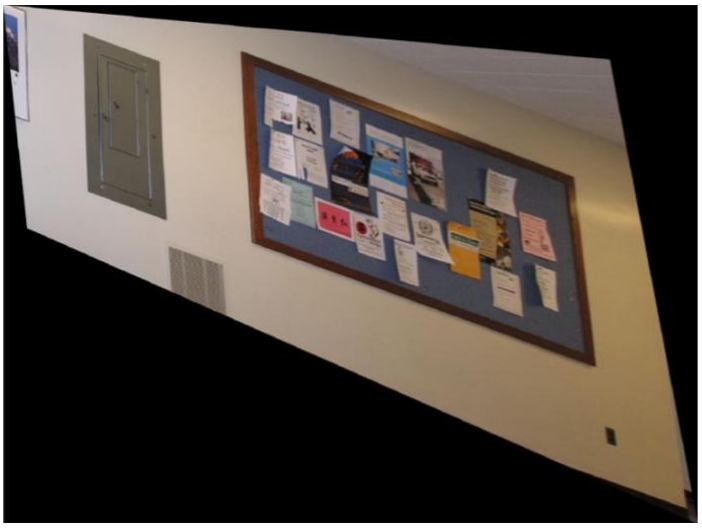


Image

Affine Rectification



Metric Rectification



Image

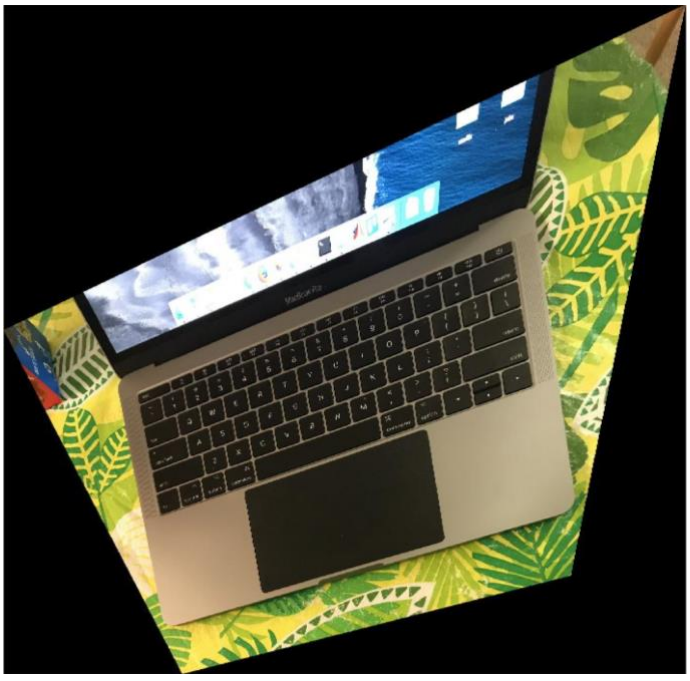
Affine Rectification

Metric Rectification

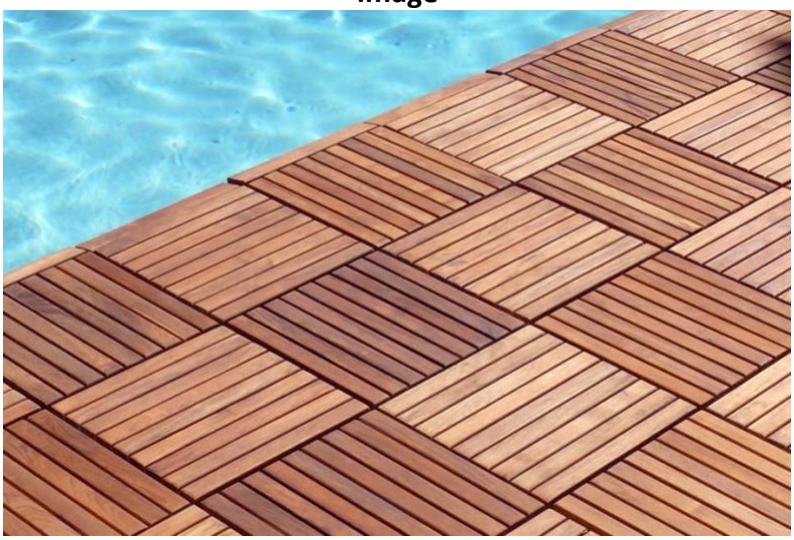
Image

Affine Rectification

Metric Rectification



Image



Affine Rectification

Metric Rectification

Image

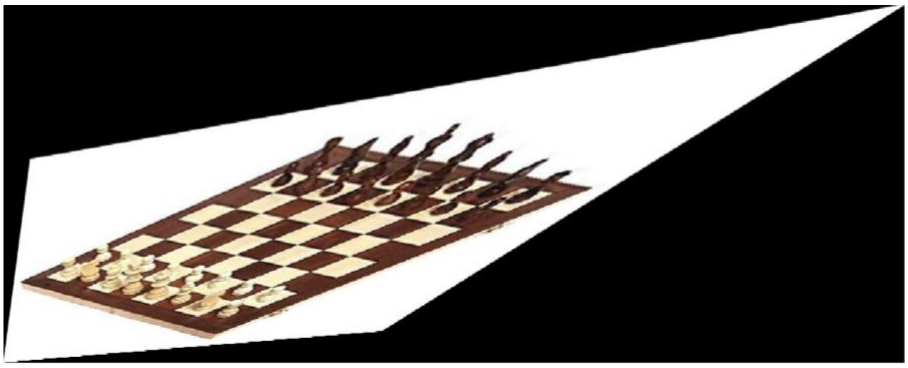
Affine Rectification

Metric Rectification

Image

Affine Rectification

Metric Rectification

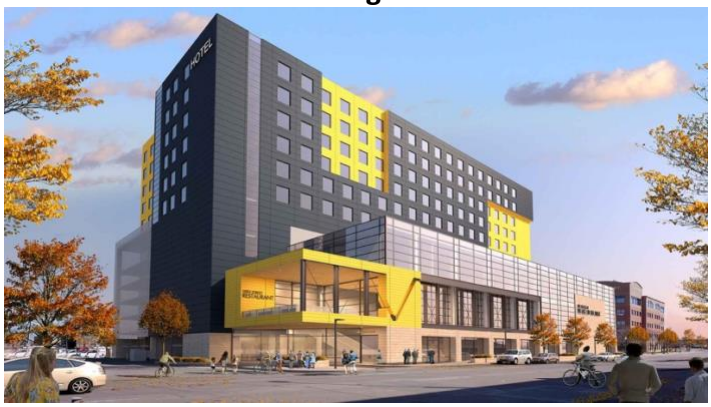


Image

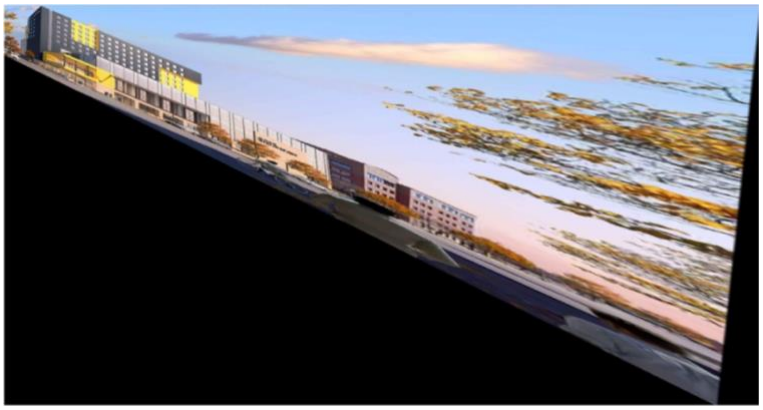
Affine Rectification

Metric Rectification

Image

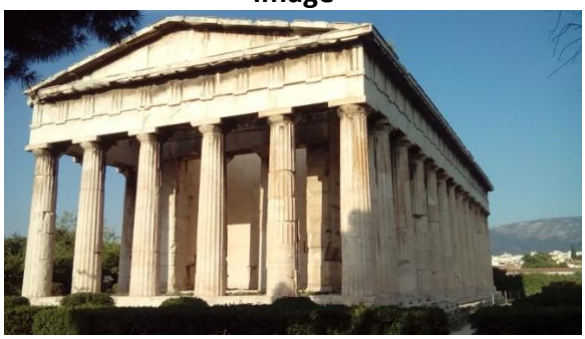


Affine Rectification



Metric Rectification

Image



Affine Rectification

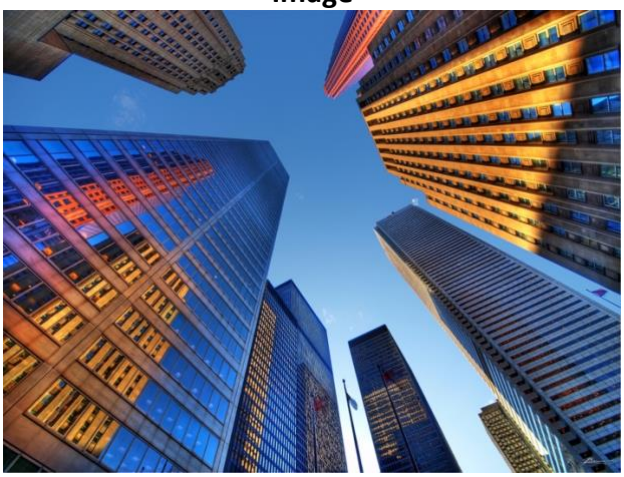
Metric Rectification

Image

Affine Rectification

Metric Rectification

Image

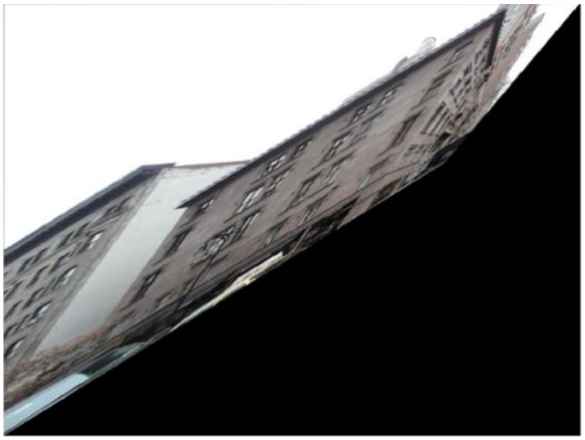


Affine Rectification

Metric Rectification

Image

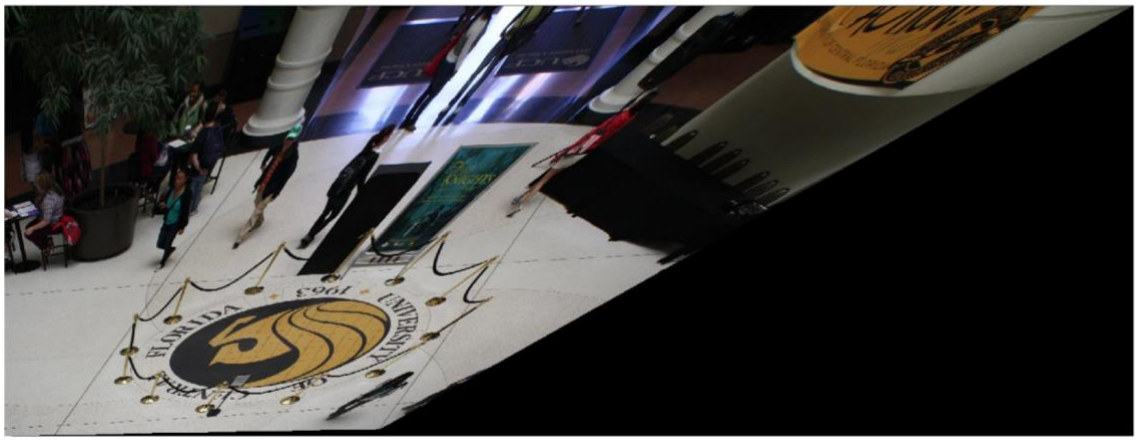
Affine Rectification



Metric Rectification

Image

Affine Rectification



Metric Rectification

Image

Affine Rectification

Metric Rectification