Student : NINH HAI DO
SID # 25949105

Submission : 06/02/2018
(2 days late)

CS280 - Homework #1

Problem 1 - Perspective Projection

1. Consider a plane based on two non-parallel lines l_1 & l_2:

$$l_1 : \quad X(\lambda) = A_1 + \lambda D_1$$
$$l_2 : \quad X(\lambda) = A_2 + \lambda D_2$$

without the loss of generality, we can take

$$D_1 = \begin{bmatrix} D_{1x} \\ D_{1y} \\ 1 \end{bmatrix} \qquad D_2 = \begin{bmatrix} D_{2x} \\ D_{2y} \\ 1 \end{bmatrix}$$

From the lecture, the two vanishing points are:

$$P_1 = \left(\frac{f D_{1x}}{\cancel{D_{1z}}^{1}}, \frac{f D_{1y}}{\cancel{D_{1z}}^{1}} \right) \qquad P_2 = \left(\frac{f D_{2x}}{\cancel{D_{2z}}^{1}}, \frac{f D_{2y}}{\cancel{D_{2z}}^{1}} \right)$$

i.e

$$P_1 = (f D_{1x}, f D_{1y}) \qquad P_2 = (f D_{2x}, f D_{2y})$$

$$(\text{since } D_{1z} = D_{2z} = 1)$$

Now we prove that any line on the plane has the vanishing point on the line $P_1 P_2$

Consider an arbitrary line l_3 on the plane. l_3 has the directional vector D_3

Since D_1, D_2 & D_3 are parallel to the plane so we can express D_3 in terms of D_1 & D_2:

$$D_3 = \alpha D_1 + \beta D_2 = \begin{bmatrix} \alpha D_{1x} + \beta D_{2x} \\ \alpha D_{1y} + \beta D_{2y} \\ \alpha + \beta \end{bmatrix}$$

Again, we can set $D_{3z} = 1$ by dividing the coordinates by $\alpha + \beta$.

$$D_3 = \begin{bmatrix} \frac{\alpha}{\alpha+\beta} D_{1x} + \frac{\beta}{\alpha+\beta} D_{2x} \\ \frac{\alpha}{\alpha+\beta} D_{1y} + \frac{\beta}{\alpha+\beta} D_{2y} \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha' D_{1x} + \beta' D_{2x} \\ \alpha' D_{1y} + \beta' D_{2y} \\ 1 \end{bmatrix} \qquad \alpha' + \beta' = 1$$

Omidding "1" we have:

$$D_3 = \begin{bmatrix} \alpha D_{1x} + \beta D_{2x} \\ \alpha D_{1y} + \beta D_{2y} \\ 1 \end{bmatrix} \qquad \text{where } \alpha + \beta = 1$$

The vanishing point of l_3 is:

$$P_3 = \left(f(\alpha D_{1x} + \beta D_{2x}), f(\alpha D_{1y} + \beta D_{2y}) \right)$$

We prove that P_1, P_2 & P_3 are colinear:

Consider 2 vector $\vec{P_1P_2}$ and $\vec{P_1P_3}$

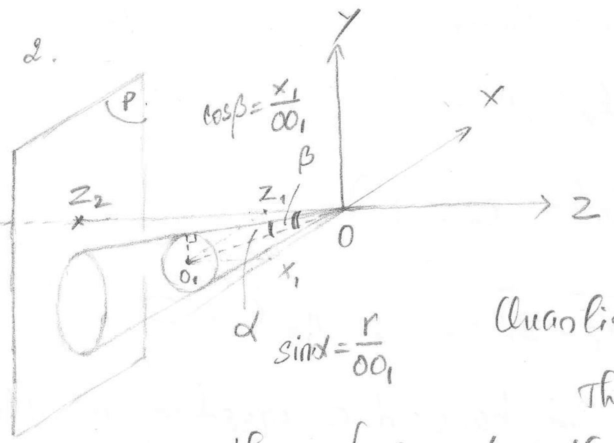
$$\vec{P_1P_2} = P_2 - P_1 = \left(f(D_{2x} - D_{1x}), f(D_{2y} - D_{1y}) \right)$$

$$\begin{aligned}\vec{P_1P_3} = P_3 - P_1 &= \left(f(\alpha D_{1x} + \beta D_{2x} - D_{1x}), f(\alpha D_{1y} + \beta D_{2y} - D_{1y}) \right) \\ &= \left(f(-\beta D_{1x} + \beta D_{2x}), f(-\beta D_{1y} + \beta D_{2y}) \right) \\ &= -\beta \left(f(D_{2x} - D_{1x}), f(D_{2y} - D_{1y}) \right) \\ &= -\beta \vec{P_1P_2}\end{aligned}$$

$\Rightarrow$ $\vec{P_1P_2} \parallel \vec{P_1P_3}$ or P_1, P_2, P_3 are colinear

$\Rightarrow$ P_3 is on the vanishing line P_1P_2

2.



The sphere centered at $O_1(X_1, 0, Z_1)$

The image plane is the XY plane at $Z = Z_2$

Quanlitatively :

The light source at origin O projects the sphere onto the image plane (P). This creates a conic section. Since the sphere centered on XZ plance does not intersect with Y axis, the light rays from O through its edge are not parallel to Y axis, thus intersecting the plane P (assuming $r < Z_1$)

The plane P cut all the conic surface → the sihouette is an ellipse.

Qualitatively :

The equation of a line l going through a point $A(x_0, y_0, z_0)$ with the direction vector $\vec{d}(a, b, c)$ takes the form :

$$\frac{X - x_0}{a} = \frac{Y - y_0}{b} = \frac{Z - z_0}{c}$$

The line l_1 going through O and O_1 has the equation.

$$\frac{X}{X_1} = \frac{Z}{Z_1} \qquad Y = 0$$

The lines l_n going through O and tangent w/ the sphere make the conic surface, They have the directional vector $\vec{d_n}(a_n, b_n, c_n)$ making w/ the line l_1 (OO_1) an angel α

$$\Rightarrow \quad \vec{d_n} \cdot \vec{OO_1} = |\vec{d_n}|\,|\vec{OO_1}| \cos\alpha = a_n X_1 + b_n \cdot 0 + c_n Z_1$$

$$\sqrt{a_n^2 + b_n^2 + c_n^2}\;\sqrt{X_1^2 + Z_1^2}\;\sqrt{1 - \frac{r^2}{OO_1^2}} = a_n X_1 + c_n Z_1$$

$$\nwarrow X_1^2 + Z_1^2$$

$$\sqrt{a_n^2+b_n^2+c_n^2}\,\sqrt{X_1^2+Z_1^2-r^2} = a_n X_1 + c_n Z_1$$

we have the equation of the lines l_n is :

$$\frac{X}{a_n} = \frac{Y}{b_n} = \frac{Z}{1} \qquad (\text{we can take } c_n = 1)$$

subject to : $\sqrt{a_n^2+b_n^2+1}\,\sqrt{X_1^2+Z_1^2-r^2} = a_n X_1 + Z_1$

l_n cut (P) at $Z = Z_2$, thus we have the equation of the contour of the silouhette is

$$\frac{X}{a_n} = \frac{Y}{b_n} = Z_2 \qquad Z = Z_2 \qquad (1)$$

subjecto $\sqrt{a_n^2+b_n^2+1}\,\sqrt{X_1^2+Z_1^2-r^2} = a_n X_1 + Z_1 \qquad (2)$

From (1) : $a_n = \dfrac{X}{Z_2}$ $\quad b_n = \dfrac{Y}{Z_2}$, substituting them into (2) gives.

$$\sqrt{\frac{X^2}{Z_2^2}+\frac{Y^2}{Z_2^2}+1}\,\sqrt{X_1^2+Z_1^2-r^2} = \frac{X}{Z_2}X_1 + Z_1$$

$$\cancel{\frac{X^2}{Z_2^2}X_1^2} + \frac{X^2}{Z_2^2}Z_1^2 - \frac{X^2}{Z_2^2}r^2 + \frac{Y^2}{Z_2^2}X_1^2 + \frac{Y^2}{Z_2^2}Z_1^2 - \frac{Y^2}{Z_2^2}r^2 + X_1^2 + \cancel{Z_1^2} - r^2$$

$$= \cancel{\frac{X^2}{Z_2^2}X_1^2} + 2\frac{X}{Z_2}X_1Z_1 + \cancel{Z_1^2}$$

$$X^2(Z_1^2-r^2) + Y^2(X_1^2+Z_1^2-r^2) - 2XX_1Z_1Z_2 + (X_1^2-r^2)Z_2^2 = 0$$

$$\left(X\sqrt{Z_1^2-r^2} - \frac{X_1Z_1Z_2}{\sqrt{Z_1^2-r^2}}\right)^2 + Y^2(X_1^2+Z_1^2-r^2) = \frac{X_1^2Z_1^2Z_2^2}{Z_1^2-r^2} - (X_1^2-r^2)Z_2^2$$

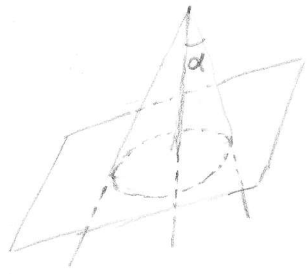
This is the equation of ellipse in 2D-XY if satisfied :

$Z_1 > r$ and $X_1^2 + Z_1^2 > r^2$ (then RHS > 0 automatically)

Thus the silouhette can be an ellipse, a hyperbola or a parabola :

+ ellipse : $z_1 > r \quad x_1^2 + z_1^2 > r^2$.
+ parabola : $z_1 = r$
+ hyperbola : $z_1 < r \quad x_1^2 + z_1^2 > r^2$.

The eccentricity :
$$e = \frac{\cos\beta}{\cos\alpha}$$

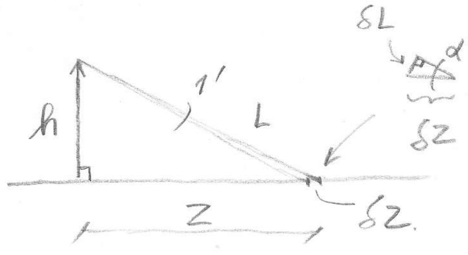


where α is the angel of conic section :
$$\cos\alpha = \sqrt{1 - \frac{r^2}{OO_1^2}} = \sqrt{\frac{x_1^2 + z_1^2 - r^2}{x_1^2 + z_1^2}}$$

β is the angel of the cutting plane (P) and the axis OO_1 of the conic section :
$$\cos\beta = \frac{x_1}{OO_1} = \frac{x_1}{\sqrt{x_1^2 + z_1^2}}$$

$$\Rightarrow e = \frac{x_1}{\sqrt{x_1^2 + z_1^2 - r^2}}$$

3.



$$\delta Z = \frac{\delta L}{\sin\alpha} \qquad \sin\alpha = \frac{h}{L}$$

$$\delta L \simeq L \times \mathrm{rad}(1')$$

$$\Rightarrow \delta Z = \frac{L \times \mathrm{rad}(1')}{h/L} = \frac{L^2 \ \mathrm{rad}(1')}{h}$$

$$\delta Z = \frac{(h^2 + Z^2) \times 0.0002909}{h}$$

Problem 2 - Rotations

1.

$$\hat{s} \times \vec{b} = S\vec{b}$$

$$\hat{s} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix} \qquad S = \begin{bmatrix} 0 & -S_3 & S_2 \\ S_3 & 0 & -S_1 \\ -S_2 & S_1 & 0 \end{bmatrix}$$

2.

$$R = e^{\phi S} = I + \phi S + \frac{(\phi S)^2}{2!} + \frac{(\phi S)^3}{3!} + \frac{(\phi S)^4}{4!} + \cdots$$

$$S = \begin{bmatrix} 0 & -S_3 & S_2 \\ S_3 & 0 & -S_1 \\ -S_2 & S_1 & 0 \end{bmatrix}$$

$$S^2 = \begin{bmatrix} 0 & -S_3 & S_2 \\ S_3 & 0 & -S_1 \\ -S_2 & S_1 & 0 \end{bmatrix} \begin{bmatrix} 0 & -S_3 & S_2 \\ S_3 & 0 & -S_1 \\ -S_2 & S_1 & 0 \end{bmatrix} = \begin{bmatrix} -S_3^2-S_2^2 & S_1S_2 & S_1S_3 \\ S_1S_2 & -S_3^2-S_1^2 & S_2S_3 \\ S_1S_3 & S_2S_3 & -S_2^2-S_1^2 \end{bmatrix}$$

$$S^3 = S^2 \cdot S = \begin{bmatrix} -S_3^2-S_2^2 & S_1S_2 & S_1S_3 \\ S_1S_2 & -S_3^2-S_1^2 & S_2S_3 \\ S_1S_3 & S_2S_3 & -S_2^2-S_1^2 \end{bmatrix} \begin{bmatrix} 0 & -S_3 & S_2 \\ S_3 & 0 & -S_1 \\ -S_2 & S_1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & S_3^3+S_2^2S_3+S_1^2S_3 & -S_2S_3^2-S_2^3-S_1^2S_2 \\ -S_3^3-S_1^2S_3-S_2^2S_3 & 0 & S_1S_2^2+S_1S_3^2+S_1^3 \\ S_2S_3^2+S_2^3+S_1^2S_2 & -S_1S_3^2-S_1S_2^2-S_1^3 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & S_3 & -S_2 \\ -S_3 & 0 & S_1 \\ S_2 & -S_1 & 0 \end{bmatrix} = -S \qquad (\text{since } S_1^2+S_2^2+S_3^2=1)$$

$$\Rightarrow \quad S^4 = -S^2 \qquad S^5 = -SS^2 = S \qquad S^6 = S^2 \qquad S^7 = -S \quad \cdots$$

$$\Rightarrow \quad R = I + \phi S + \frac{\phi^2 S^2}{2!} - \frac{\phi^3 S}{3!} - \frac{\phi^4 S^2}{4!} + \frac{\phi^5 S}{5!} + \frac{\phi^6 S^2}{6!} - \frac{\phi^7 S}{7!} - \cdots$$

p.T

$$= I + \underbrace{\left(\phi - \frac{\phi^3}{3!} + \frac{\phi^5}{5!} - \frac{\phi^7}{7!} + \dots\right)}_{\sin\phi} S + \underbrace{\left(\frac{\phi^2}{2!} - \frac{\phi^4}{4!} + \frac{\phi^6}{6!} - \frac{\phi^8}{8!} + \dots\right)}_{1-\cos\phi} S^2$$

$$\Rightarrow \quad R = I + \sin\phi\, S + (1-\cos\phi)\, S^2$$

3. Plot : See Appendix 1

Coding : See appendix 3

4. Coding : See appendix 3

Analytical Verification :

$s = [0.6, 0.8, 0]$ $u = [-0.8, 0.6, 0]$ $v = [0, 0, 1]$

Point : $p = [0, 1, 0]$

$\phi = 0$:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Since R is the identity matrix, we can say immediately :

+ eigenvalues = 1 , 1 , 1
+ eigenvector = $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

$$\begin{matrix} \swarrow & \downarrow & \downarrow \\ 0.6s - 0.8v & 0.8s + 0.6v & v \end{matrix}$$

$\phi = \frac{\pi}{12}$:

$$R = \begin{bmatrix} 0.978 & 0.016 & 0.207 \\ 0.016 & 0.988 & -0.155 \\ -0.207 & 0.155 & 0.969 \end{bmatrix}$$

$$\det(R - \lambda I) = 0$$

$$\det\left(\begin{bmatrix} 0.978 - \lambda & 0.016 & 0.207 \\ 0.016 & 0.98 - \lambda & -0.155 \\ -0.207 & 0.155 & 0.969 - \lambda \end{bmatrix}\right) = 0$$

$\Rightarrow \lambda = 0.966 + 0.259i \; , \; 0.966 - 0.259i \; , \; 1$

$\lambda = 0.966 + 0.259i$: $\quad Rv = \lambda v$

$\Rightarrow \quad v = [0.566i\ ,\ -0.566i\ ,\ 0.6]^T$

$= 0.113i \times s - 0.792i \times u + 0.6v$

$\lambda = 0.966 - 0.259i$: $\quad Rv = \lambda v$

$\Rightarrow \quad v = [-0.424i\ ,\ 0.424i\ ,\ 0.8]^T$

$= 0.085i \cdot s + 0.594i \cdot u + 0.8v$

$\lambda = 1$: $\quad Rv = \lambda v$

$\Rightarrow \quad v = [-0.707\ ,\ -0.707\ ,\ 0]$

$= -0.99s + 0.14u$

$\phi = \frac{\pi}{8}$:

$$R = \begin{bmatrix} 0.951 & 0.037 & 0.306 \\ 0.037 & 0.973 & -0.23 \\ -0.306 & 0.23 & 0.924 \end{bmatrix}$$

$\det(R - \lambda I) = 0$

$$\det\left(\begin{bmatrix} 0.951-\lambda & 0.037 & 0.306 \\ 0.037 & 0.973-\lambda & -0.23 \\ -0.306 & 0.23 & 0.924-\lambda \end{bmatrix}\right) = 0$$

$\Rightarrow \lambda = 0.934 + 0.383i\ ,\ 0.934 - 0.383i\ ,\ 1$

$\lambda = 0.934 + 0.383i$: $\quad Rv = \lambda v$

$\Rightarrow \quad v = [-0.566i\ ,\ 0.566i\ ,\ 0.6]^T$

$= 0.113i \cdot s - 0.792i \cdot u + 0.6v$

$\lambda = 0.934 - 0.383i$: $\quad Rv = \lambda v$

$\Rightarrow \quad v = [0.424i\ ,\ -0.424i\ ,\ 0.8]^T$

$= -0.085i \cdot s - 0.594i \cdot u + 0.8v$

$\lambda = 1$: $\quad Rv = \lambda v$

$\Rightarrow \quad v = [0.707\ ,\ 0.707\ ,\ 0]^T$

$= 0.99s - 0.14u$

$\phi = \frac{\pi}{6}$: $R = \begin{bmatrix} 0.914 & 0.064 & 0.4 \\ 0.064 & 0.952 & -0.3 \\ -0.4 & 0.3 & 0.866 \end{bmatrix}$

$\det(R - \lambda I) = 0$

$\det\left(\begin{bmatrix} 0.914-\lambda & 0.064 & 0.4 \\ 0.064 & 0.952-\lambda & -0.3 \\ -0.4 & 0.3 & 0.866-\lambda \end{bmatrix}\right) = 0$

$\Rightarrow \lambda = 0.866 + 0.5i, \ 0.866 - 0.5i, \ 1$

$\lambda = 0.866 + 0.5i$: $Rv = \lambda v$

$\Rightarrow v = [0.566i, -0.566i, 0.6]^T$

$= -0.113i \cdot s - 0.292i \cdot u + 0.6v$

$\lambda = 0.866 - 0.5i$: $Rv = \lambda v$

$\Rightarrow v = [-0.424i, 0.424i, 0.8]^T$

$= 0.085i\, s + 0.594i \cdot u + 0.8v$

$\lambda = 1$: $Rv = \lambda v$

$\Rightarrow v = [-0.707, -0.707, 0]$

$= -0.99s + 0.14u$

$\phi = \frac{\pi}{4}$: $R = \begin{bmatrix} 0.813 & 0.14 & 0.566 \\ 0.14 & 0.895 & -0.424 \\ -0.566 & 0.424 & 0.707 \end{bmatrix}$

$\det(R - \lambda I) = 0$

$\det\left(\begin{bmatrix} 0.813-\lambda & 0.14 & 0.566 \\ 0.14 & 0.995-\lambda & -0.424 \\ -0.566 & 0.424 & 0.707-\lambda \end{bmatrix}\right) = 0$

$\Rightarrow \lambda = 0.707 + 0.707i, \ 0.707 - 0.707i, 1$

$\lambda = 0.707 + 0.707i \quad : \quad Rv = \lambda v$

$\Rightarrow \quad v = [0.566i \, , \, -0.566i \, , \, 0.6]$

$= -0.1131i \cdot s - 0.792i \cdot u + 0.6v$

$\lambda = 0.707 - 0.707i \quad : \quad Rv = \lambda v$

$\Rightarrow \quad v = [-0.424i \, , \, 0.424i \, , \, 0.8]^T$

$= 0.085i \cdot s + 0.594i \cdot u + 0.8v$

$\lambda = 1 \quad : \quad Ru = \lambda u$

$\Rightarrow \quad v = [-0.707 \, , \, -0.707 \, , \, 0$

$= -0.99s + 0.14u$

$$\phi = \frac{\pi}{2} : \quad R = \begin{bmatrix} 0.36 & 0.48 & +0.8 \\ 0.48 & 0.64 & -0.6 \\ -0.8 & 0.6 & 0 \end{bmatrix}$$

$\det(R - \lambda I) = 0$

$$\det\left(\begin{bmatrix} 0.36-\lambda & 0.48 & 0.8 \\ +0.48 & 0.64-\lambda & -0.6 \\ -0.8 & 0.6 & 0-\lambda \end{bmatrix}\right) = 0$$

$\Rightarrow \quad \lambda = i \, , \, -i \, , \, 1$

$\lambda = i \quad : \quad Rv = \lambda v$

$\Rightarrow \quad v = [-0.566i \, , \, 0.566i \, , \, 0.6]^T$

$= 0.113i \cdot s - 0.792i \cdot u + 0.6v.$

$\lambda = -i \quad : \quad Rv = \lambda v.$

$\Rightarrow \quad v = [0.424i \, , \, -0.424i \, , \, 0.8]^T$

$= -0.085i \cdot s - 0.594i \cdot u + 0.8v.$

$\lambda = 1 \quad : \quad Rv = \lambda v$

$\Rightarrow \quad v = [0.707 \, , \, 0.707 \, , \, 0]^T$

$= 0.99s - 0.14u$

$\phi = \pi$: $R = \begin{bmatrix} -0.28 & 0.96 & 0 \\ 0.96 & 0.28 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

$\det(R - \lambda I) = 0$

$\det\left(\begin{bmatrix} -0.28-\lambda & 0.96 & 0 \\ 0.96 & 0.28-\lambda & 0 \\ 0 & 0 & -\lambda \end{bmatrix}\right) = 0$

$\Rightarrow \lambda = -1, 1, -1$

$\lambda = -1$: $Rv = \lambda v$

$\Rightarrow v = [-0.8 \quad 0.6 \quad 0.386]^T$

$= u + 0.386 u.$

$\lambda = 1$: $Rv = \lambda u$

$\Rightarrow v = [0.6 \quad 0.8 \quad -0.29]^T$

$= s - 0.29 v.$

$\lambda = -1$: $Rv = \lambda u$

$\Rightarrow v = [0 \quad 0 \quad 0.876]$

$= 0.876 v.$

$\phi = \frac{3\pi}{4}$: $R = \begin{bmatrix} 0.36 & 0.48 & -0.8 \\ 0.48 & 0.64 & 0.6 \\ 0.8 & -0.6 & 0 \end{bmatrix}$

$\det(R - \lambda I) = 0$

$\det\left(\begin{bmatrix} 0.36-\lambda & 0.48 & -0.8 \\ 0.48 & 0.64-\lambda & 0.6 \\ 0.8 & -0.6 & -\lambda \end{bmatrix}\right) = 0$

$\Rightarrow \quad \lambda = i, -i, 1$

eigen vectors & their suv representation is same as $\phi = \frac{\pi}{4}$

5. From 2. :

$$R = I + \sin\phi\, S + (1-\cos\phi) S^2$$

$$\Rightarrow tr(R) = \overset{\nearrow 3}{tr(I)} + \sin\phi\, tr(S) + (1-\cos\phi)\, tr(S^2)$$

$$S = \begin{bmatrix} 0 & -s_3 & s_2 \\ s_3 & 0 & s_1 \\ -s_2 & s_1 & 0 \end{bmatrix} \Rightarrow tr(S) = 0$$

$$S^2 = \begin{bmatrix} -s_3^2-s_2^2 & s_1 s_2 & s_1 s_3 \\ s_1 s_2 & -s_3^2-s_1^2 & s_2 s_3 \\ s_1 s_3 & s_2 s_3 & -s_2^2-s_1^2 \end{bmatrix} \Rightarrow tr(S^2) = -s_3^2-s_2^2-s_3^2-s_1^2-s_2^2-s_1^2 = -2$$

since $s_1^2+s_2^2+s_3^2 = 1$

$$\Rightarrow tr(R) = 3 + 0 + (1-\cos\phi)(-2) = 1 + 2\cos\phi$$

$$\Rightarrow \cos\phi = \frac{1}{2}(tr(R) - 1)$$

6. Coding of function rot_to_ax_phi.py : see Appendix 3

Basically, given

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$$\cos\phi = \frac{1}{2}(trace(R) - 1) \Rightarrow \phi = \arccos(\cos\phi)$$

$$\hat{s} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} = \frac{1}{2\sin\phi} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}$$

Problem 3

$$\vec{v}_i = \begin{bmatrix} v_{ix} \\ v_{iy} \\ 1 \end{bmatrix} = \begin{bmatrix} V_{ix}/V_{iz} \\ V_{iy}/V_{iz} \\ 1 \end{bmatrix} \quad \text{where} \quad \begin{bmatrix} V_{ix} \\ V_{iy} \\ V_{iz} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{ix} \\ u_{iy} \\ 1 \end{bmatrix} = H\vec{u}_i$$

(*)

1. $H^* = \operatorname{argmin}_H \sum_{i=1}^{4} \| T(\vec{u}_i) - \vec{v}_i \|^2$ wher $\vec{v}_i = T(\vec{u}_i)$

(a) 2D affine transformation

$$\vec{v} = A\vec{u} + \vec{t}$$

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

We can re-write in the form:

$$\begin{bmatrix} v_x \\ v_y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ 1 \end{bmatrix}$$

Comparing this one to (*) we see that we can set $h_{31} = h_{32} = 0$, and thus $V_{iz} = 1$. The equation for affine transformation reduces to:

$$\begin{bmatrix} v_{ix} \\ v_{iy} \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_{ix} \\ u_{iy} \\ 1 \end{bmatrix}$$

$$h_{11} u_{ix} + h_{12} u_{iy} + h_{13} = v_{ix}$$

$$h_{21} u_{ix} + h_{22} u_{iy} + h_{23} = v_{iy}$$

In the matrix form:

$$\underbrace{\begin{bmatrix} u_{ix} & u_{iy} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & u_{ix} & u_{iy} & 1 \end{bmatrix}}_{X_i} \underbrace{\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \end{bmatrix}}_{w} = \underbrace{\begin{bmatrix} v_{ix} \\ v_{iy} \end{bmatrix}}_{y_i}$$

We have 6 unknowns h's $\rightarrow$ need 6 eqns $\rightarrow$ need 3 points.
As long as we have 3 points, we can form 6 eqns, stack them on top of each other to get the form

$$Xw = y$$

The problem reduces to $w^* = \text{argmin}_w \| Xw - y \|^2$

Solution: $w^* = (X^TX)^{-1} X^T y$

(b) Homography

$$\begin{bmatrix} V_{ix} \\ V_{iy} \\ V_{iz} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{ix} \\ u_{iy} \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} h_{11} u_{ix} + h_{12} u_{iy} + h_{13} \\ h_{21} u_{ix} + h_{22} u_{iy} + h_{23} \\ h_{31} u_{ix} + h_{32} u_{iy} + 1 \end{bmatrix}$$

We have:

$$\vec{v}_i = \begin{bmatrix} v_{ix} \\ v_{iy} \\ 1 \end{bmatrix} = \begin{bmatrix} V_{ix}/V_{iz} \\ V_{iy}/V_{iz} \\ 1 \end{bmatrix}$$

$$\Rightarrow \quad v_{ix} = \frac{h_{11} u_{ix} + h_{12} u_{iy} + h_{13}}{h_{31} u_{ix} + h_{32} u_{iy} + 1}$$

$$v_{iy} = \frac{h_{21} u_{ix} + h_{22} u_{iy} + h_{23}}{h_{31} u_{ix} + h_{32} u_{iy} + 1}$$

Re-arrange the two eqns, we get:

$$h_{11} u_{ix} + h_{12} u_{iy} + h_{13} \qquad - h_{31} u_{ix} v_{ix} - h_{32} u_{iy} v_{ix} = v_{ix}$$

$$h_{21} u_{ix} + h_{22} u_{iy} + h_{23} - h_{31} u_{ix} v_{iy} - h_{32} u_{iy} v_{iy} = v_{iy}$$

Or in matrix form:

$$\underbrace{\begin{bmatrix} u_{ix} & u_{iy} & 1 & 0 & 0 & 0 & -u_{ix} v_{ix} & -u_{iy} v_{ix} \\ 0 & 0 & 0 & u_{ix} & u_{iy} & 1 & -u_{ix} v_{iy} & -u_{iy} v_{iy} \end{bmatrix}}_{X_i} \underbrace{\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}}_{w} = \underbrace{\begin{bmatrix} v_{ix} \\ v_{iy} \end{bmatrix}}_{y_i}$$

We have 8 unknowns h's $\rightarrow$ need 8 eqns $\rightarrow$ need 4 points.
we can stack the eqns on top of each other to get the form

$$Xw = y$$

The problem reduces to $w^* = \arg\min_w \|Xw - y\|^2$

Solution : $w^* = (X^T X)^{-1} X^T y$

2. The constraints for affine transform is

$$h_{31} = h_{32} = 0$$

and the other elements of H satisfy:

$$h^* = (U^T U)^{-1} U^T V \quad \text{for 3 points}$$

where U is the matrix formed by 3 input points

V is the flattened vector of 3 output points

The constraint for homography transformation is

$$h^* = (U^T U)^{-1} U^T V \quad \text{for 4 points}$$

where U is the matrix formed by 4 input points

V is the flattened vector of 3 output points

3. The affine transform is <u>NOT</u> able to exactly transform the points from one image to another because it conserves the parallelism while different images have different perspectives

(parallel lines in one image may not be parallel in the others)

The homography is able. It deforms the perspective in one image to adapt the others.

4. Images : See Appendix 3

Coding : See Appendix 4

Observations : the affine transform produces the parallelogram output that does not fit different surfaces in the target images

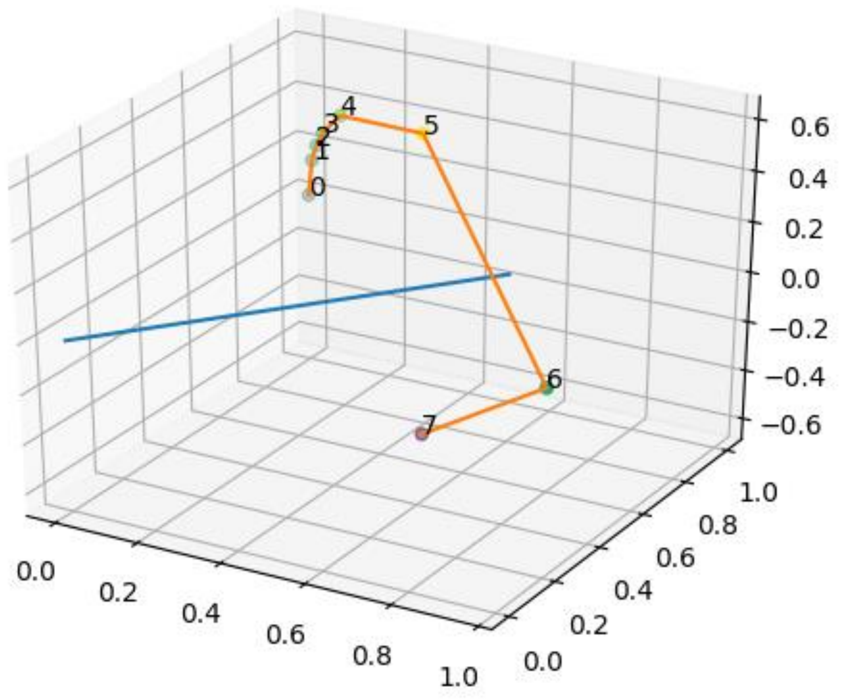
The homography transform produces the deformed output that perfectly fit different surfaced in the target images.

5. Images : See Appendix 3

Coding : See Appendix 4

Appendix 1

Problem 2.3



```
s = [0.6 0.8 0. ]
u = [-0.8  0.6  0. ]
v = [0 0 1]
Point: [0 1 0]

Phi = 0.0:
The rotation matrix R:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
Eigenvalues:
[1. 1. 1.]
Eigenvector: [1. 0. 0.]
SUV representation: 0.6s - 0.8u + 0.0v
Eigenvector: [0. 1. 0.]
SUV representation: 0.7999999999999998s + 0.6000000000000001u + 0.0v
Eigenvector: [0. 0. 1.]
SUV representation: 0.0s + 0.0u + 1.0v

Phi = 0.2617993877991494:
```

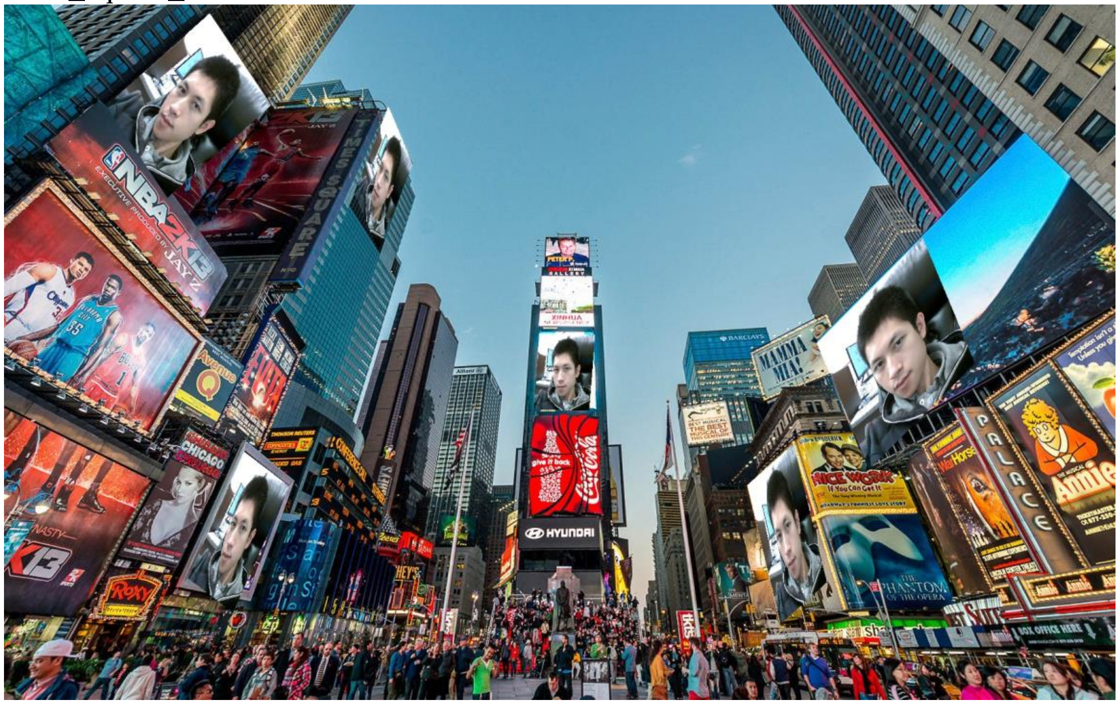
```
The rotation matrix R:
[[ 0.97819253  0.0163556   0.20705524]
 [ 0.0163556   0.9877333  -0.15529143]
 [-0.20705524  0.15529143  0.96592583]]
Eigenvalues:
[0.96592583+0.25881905j 0.96592583-0.25881905j 1.        +0.        j]
Eigenvector: [2.02615702e-15+0.56568542j 2.02615702e-15-0.56568542j
 6.00000000e-01+0.        j]
SUV representation: (2.8366198279172746e-15-0.11313708498984759j)s + (-4.052314039881822e-
16-0.791959594928934j)u + (0.6000000000000006+0j)v
Eigenvector: [-1.38777878e-16-0.42426407j -1.38777878e-16+0.42426407j
  8.00000000e-01+0.        j]
SUV representation: (-1.9428902930940237e-16+0.08485281374238546j)s + (2.77555756156289e-
17+0.5939696961966988j)u + (0.7999999999999993+0j)v
Eigenvector: [-7.07106781e-01+0.j -7.07106781e-01-0.j -1.19297591e-16+0.j]
SUV representation: (-0.9899494936611666+0j)s + (0.14142135623730945+0j)u + (-
1.1929759116026628e-16+0j)v

Phi = 0.39269908169872414:
The rotation matrix R:
[[ 0.9512829   0.03653782  0.30614675]
 [ 0.03653782  0.97259663 -0.22961006]
 [-0.30614675  0.22961006  0.92387953]]
Eigenvalues:
[0.92387953+0.38268343j 0.92387953-0.38268343j 1.        +0.        j]
Eigenvector: [2.77555756e-17-0.56568542j 2.77555756e-17+0.56568542j
 6.00000000e-01+0.        j]
SUV representation: (3.8857805861880476e-17+0.11313708498984743j)s + (-5.5511151231257815e-
18+0.7919595949289329j)u + (0.6000000000000001+0j)v
Eigenvector: [-1.38777878e-16+0.42426407j -1.38777878e-16-0.42426407j
  8.00000000e-01+0.        j]
SUV representation: (-1.9428902930940237e-16-0.08485281374238557j)s + (2.77555756156289e-17-
0.5939696961966994j)u + (0.7999999999999998+0j)v
Eigenvector: [ 7.07106781e-01+0.j  7.07106781e-01-0.j -1.22728786e-16+0.j]
SUV representation: (0.989949493661167+0j)s + (-0.14142135623730956+0j)u + (-
1.227287863770378e-16+0j)v

Phi = 0.5235987755982988:
The rotation matrix R:
[[ 0.91425626  0.06430781  0.4       ]
 [ 0.06430781  0.95176915 -0.3       ]
 [-0.4         0.3         0.8660254 ]]
Eigenvalues:
[0.8660254+0.5j 0.8660254-0.5j 1.       +0. j]
Eigenvector: [7.07767178e-16+0.56568542j 7.07767178e-16-0.56568542j
 6.00000000e-01+0.        j]
SUV representation: (9.908740494779952e-16-0.11313708498984748j)s + (-1.4155343563970745e-16-
0.791959594928933j)u + (0.6000000000000003+0j)v
Eigenvector: [-3.12250226e-16-0.42426407j -3.12250226e-16+0.42426407j
```

```
  8.00000000e-01+0.        j]
SUV representation: (-4.3715031594615534e-16+0.08485281374238557j)s + (6.245004513516503e-
17+0.5939696961966995j)u + (0.7999999999999997+0j)v
Eigenvector: [-7.07106781e-01+0.j -7.07106781e-01-0.j -3.88353777e-16+0.j]
SUV representation: (-0.989949493661167+0j)s + (0.14142135623730956+0j)u + (-
3.8835377691477215e-16+0j)v

Phi = 0.7853981633974483:
The rotation matrix R:
[[ 0.81254834  0.14058875  0.56568542]
 [ 0.14058875  0.89455844 -0.42426407]
 [-0.56568542  0.42426407  0.70710678]]
Eigenvalues:
[0.70710678+0.70710678j 0.70710678-0.70710678j 1.        +0.        j]
Eigenvector: [1.2490009e-16+0.56568542j 1.2490009e-16-0.56568542j
 6.0000000e-01+0.        j]
SUV representation: (1.7486012637846213e-16-0.11313708498984754j)s + (-2.498001805406602e-
17-0.7919595949289333j)u + (0.6000000000000001+0j)v
Eigenvector: [-4.16333634e-17-0.42426407j -4.16333634e-17+0.42426407j
  8.00000000e-01+0.        j]
SUV representation: (-5.828670879282071e-17+0.08485281374238562j)s + (8.326672684688675e-
18+0.5939696961967j)u + (0.7999999999999999+0j)v
Eigenvector: [-7.07106781e-01+0.j -7.07106781e-01-0.j  1.60936741e-16+0.j]
SUV representation: (-0.9899494936611666+0j)s + (0.14142135623730945+0j)u +
(1.6093674092148539e-16+0j)v

Phi = 1.5707963267948966:
The rotation matrix R:
[[ 3.60000000e-01  4.80000000e-01  8.00000000e-01]
 [ 4.80000000e-01  6.40000000e-01 -6.00000000e-01]
 [-8.00000000e-01  6.00000000e-01  1.11022302e-16]]
Eigenvalues:
[1.11022302e-16+1.j 1.11022302e-16-1.j 1.00000000e+00+0.j]
Eigenvector: [-3.88578059e-16-0.56568542j -3.88578059e-16+0.56568542j
  6.00000000e-01+0.        j]
SUV representation: (-5.440092820663266e-16+0.11313708498984754j)s + (7.771561172376093e-
17+0.7919595949289335j)u + (0.6+0j)v
Eigenvector: [2.22044605e-16+0.42426407j 2.22044605e-16-0.42426407j
 8.00000000e-01+0.        j]
SUV representation: (3.108624468950438e-16-0.08485281374238562j)s + (-4.440892098500625e-17-
0.5939696961966998j)u + (0.8+0j)v
Eigenvector: [ 7.07106781e-01+0.j  7.07106781e-01-0.j -2.64436103e-16+0.j]
SUV representation: (0.9899494936611662+0j)s + (-0.14142135623730945+0j)u + (-
2.6443610263447547e-16+0j)v

Phi = 3.141592653589793:
The rotation matrix R:
[[-2.80000000e-01  9.60000000e-01  9.79717439e-17]
 [ 9.60000000e-01  2.80000000e-01 -7.34788079e-17]
 [-9.79717439e-17  7.34788079e-17 -1.00000000e+00]]
Eigenvalues:
[-1.  1. -1.]
Eigenvector: [-0.8         0.6         0.38635862]
SUV representation: -1.1102230246251565e-16s + 1.0000000000000002u + 0.3863586229535932v
Eigenvector: [ 0.6         0.8        -0.28976897]
SUV representation: 0.9999999999999999s + 1.1102230246251565e-16u + -0.2897689672151948v
Eigenvector: [-6.12323400e-17 -8.16431199e-17  8.75648879e-01]
SUV representation: -1.020538999289461e-16s + -2.465190328815662e-32u + 0.8756488794650756v

Phi = 4.71238898038469:
The rotation matrix R:
[[ 3.60000000e-01  4.80000000e-01 -8.00000000e-01]
 [ 4.80000000e-01  6.40000000e-01  6.00000000e-01]
 [ 8.00000000e-01 -6.00000000e-01 -2.22044605e-16]]
Eigenvalues:
[-4.4408921e-16+1.j -4.4408921e-16-1.j  1.0000000e+00+0.j]
Eigenvector: [-6.9388939e-17+0.56568542j -6.9388939e-17-0.56568542j
 -6.0000000e-01+0.        j]
SUV representation: (-9.714451465470118e-17-0.11313708498984754j)s + (1.3877787807814457e-17-
0.7919595949289335j)u + (-0.6+0j)v
Eigenvector: [ 8.32667268e-17-0.42426407j  8.32667268e-17+0.42426407j
 -8.00000000e-01+0.        j]
SUV representation: (1.1657341758564142e-16+0.08485281374238562j)s + (-1.6653345369377348e-
17+0.5939696961967j)u + (-0.8+0j)v
Eigenvector: [ 7.07106781e-01+0.j  7.07106781e-01-0.j -9.81955080e-17+0.j]
SUV representation: (0.9899494936611668+0j)s + (-0.1414213562373095+0j)u + (-
9.819550802290167e-17+0j)v
```

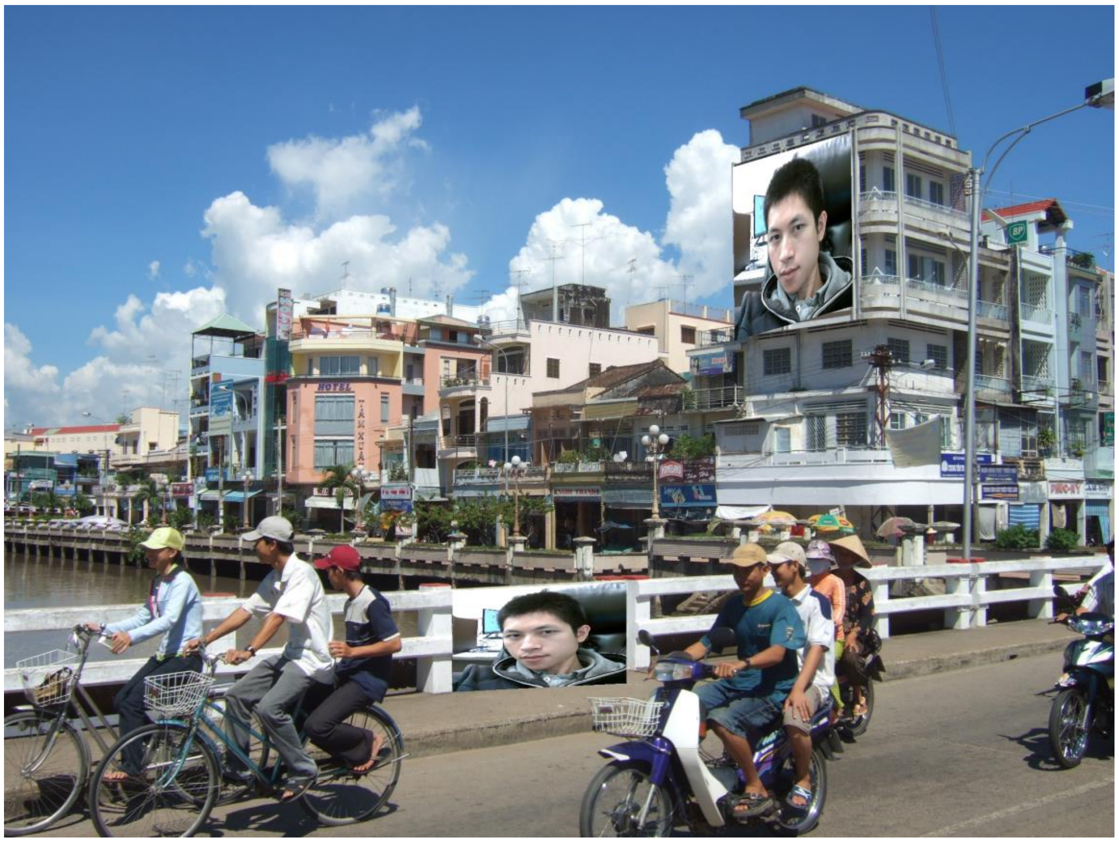
Appendix 2

Problem 3.4

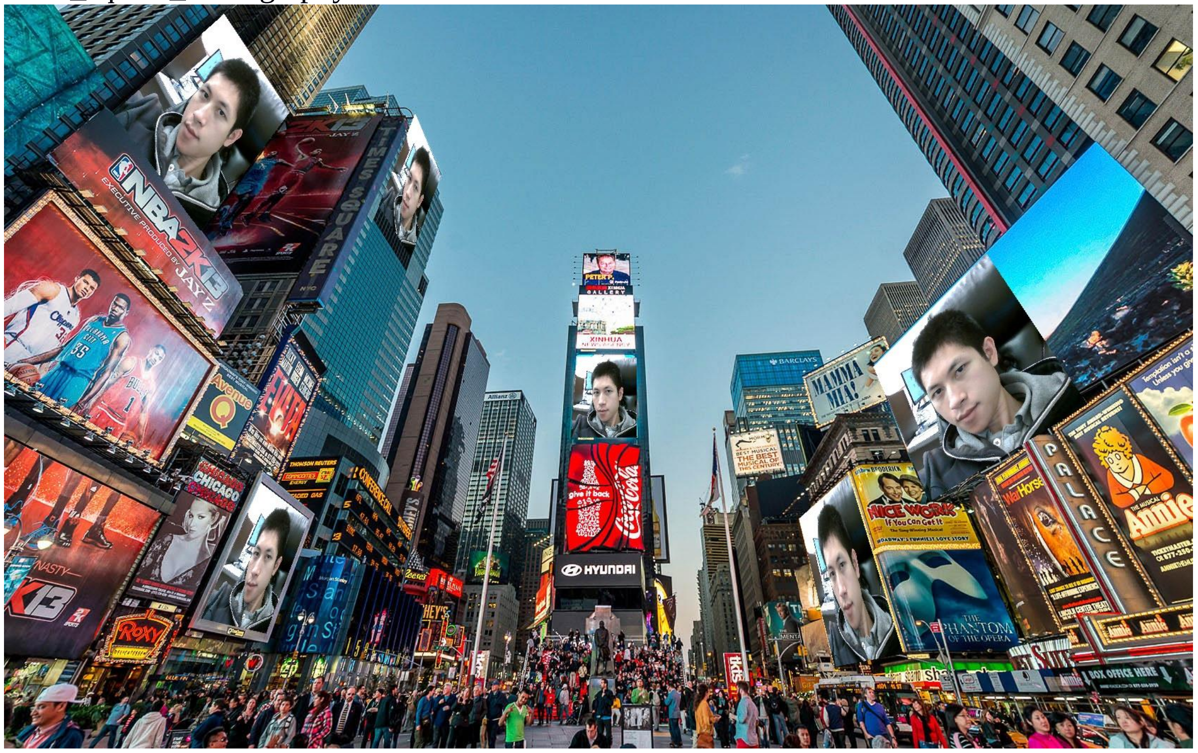
times_square_affine:



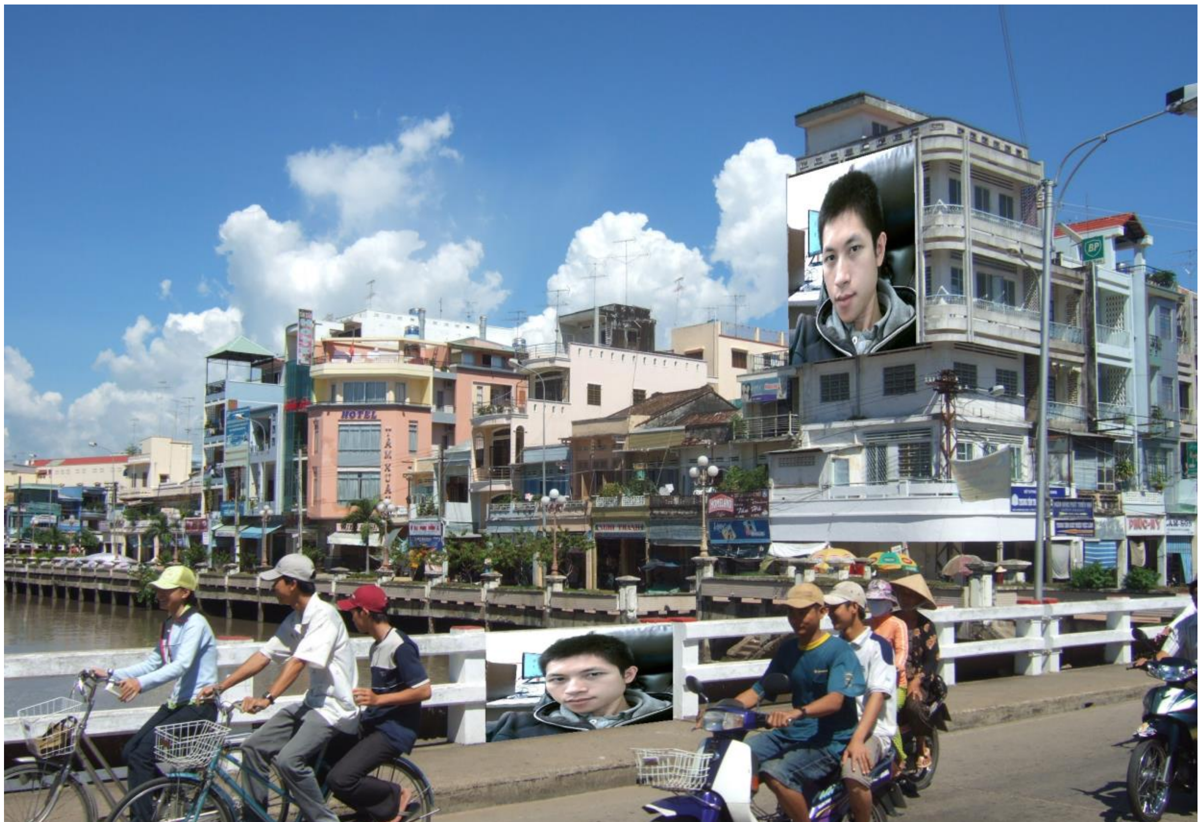
hometown_affine:



times_square_homography:

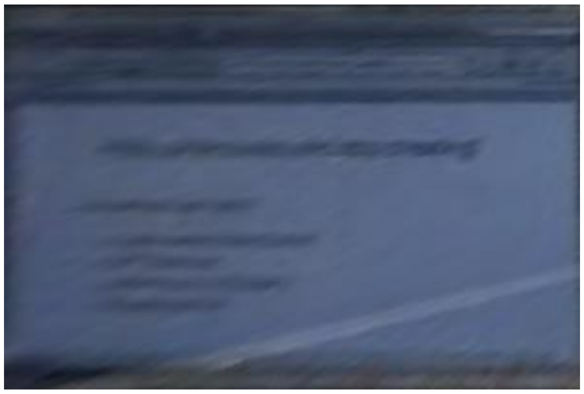


hometown_homography:

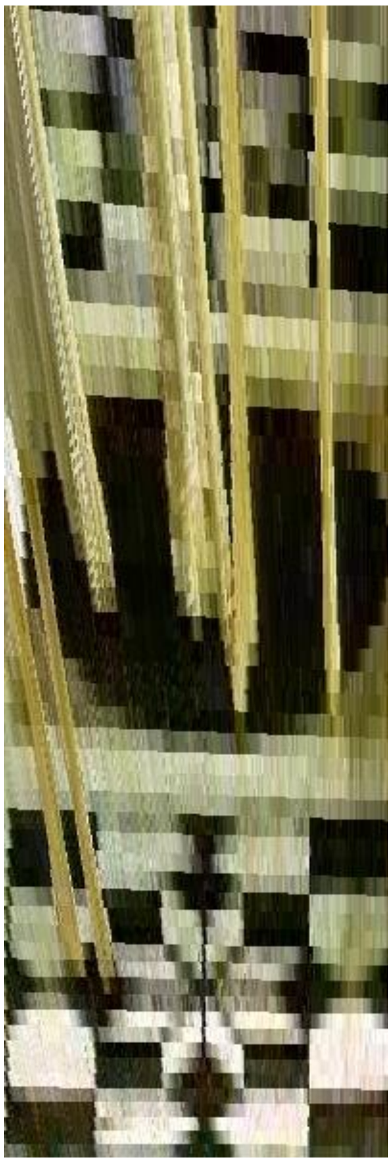


Problem 3.5

computer_screen_rectified_homography:



flagellation_rectified_homography:



computer_screen_rectified_affine:

lagellation_rectified_affine:

Appendix 3

Coding of problem 2: compute_R.py and rot_to_ax_phi.py

```
import numpy as np
import numpy.linalg as LA
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

def compute_R(phi, s):
        S = [0, -s[2], s[1],
                s[2], 0, -s[0],
                -s[1], s[0], 0]
        S = np.array(S).reshape(3, 3)
        I = np.identity(3)
        R = I + np.sin(phi) * S + (1 - np.cos(phi)) * S.dot(S)
        return R



def find_suv_coordinates(s, u, v, vector):
        SUV = np.stack((s, u, v))
        SUV = SUV.T
        return LA.inv(SUV).dot(vector)



def rot_to_ax_phi(R):
        trR = R[0, 0] + R[1, 1] + R[2, 2]
        phi = np.arccos(0.5 * (trR - 1))
        ax = 1 / (2 * np.sin(phi)) \
                * np.array([R[2,1] - R[1,2], R[0,2] - R[2,0], R[1,0] - R[0,1]])
        return phi, ax



if __name__ == "__main__":
        origin = np.array([0, 0, 0])
        s = np.array([0.6, 0.8, 0])
        u = np.array([-0.8, 0.6, 0])
        v = np.array([0, 0, 1])
        p = np.array([0, 1, 0])
        p_list = []
        phi_list = [0, np.pi/12, np.pi/8, np.pi/6,
                        np.pi/4, np.pi/2, np.pi, 3*np.pi/2]
        phi_arr = np.array(phi_list)

        print("s = " + str(s))
        print("u = " + str(u))
        print("v = " + str(v))
        print("Point: " + str(p))
```

```
for phi in phi_arr:
    R = compute_R(phi, s)
    print("Phi = " + str(phi) + ":")
    print("The rotation matrix R:")
    print(R)
    p_new = R.dot(p)
    p_list.append(p_new)
    eigenvalues, eigenvectors = LA.eig(R)
    print("Eigenvalues:")
    print(eigenvalues)
    for vector in eigenvectors:
        print("Eigenvector: " + str(vector))
        x, y, z = find_suv_coordinates(s, u, v, vector)
        print("SUV representation: " + str(x) + "s + " + str(y) + "u + " + str(z) + "v")
    print()

rotation_axis = np.stack((origin, s))
rotated_points = np.stack(p_list)

fig = plt.figure()
ax = plt.axes(projection="3d")
ax.plot3D(rotation_axis[:,0], rotation_axis[:,1], rotation_axis[:,2])
c = rotated_points[:,0] + rotated_points[:,1] + rotated_points[:,2]
ax.plot3D(rotated_points[:,0], rotated_points[:,1], rotated_points[:,2])
ax.scatter(rotated_points[:,0], rotated_points[:,1], rotated_points[:,2], c=c)
for i in range(len(phi_list)):
    ax.text(rotated_points[i,0], rotated_points[i,1], rotated_points[i,2], str(i))
plt.show()
```

Appendix 4

Coding of problem 3: affine_solve.py, homography_solve.py and homography_transform.py

```
import numpy as np
import numpy.linalg as LA
import matplotlib.pyplot as plt

times_square = "./hw1_package/hw1_package/images/times_square.jpg"
hometown = "./hw1_package/hw1_package/images/Rach-Gia-City-worth-exploring.jpg"
my_pic = "./hw1_package/hw1_package/images/my_pic.jpg"
computer_screen = "./hw1_package/hw1_package/images/computer_screen.png"
flagellation = "./hw1_package/hw1_package/images/the_flagellation.jpg"

times_square_surfaces = [[[497, 810], [499, 893], [611, 801], [610, 894]],
                                                    [[155, 581], [241,617], [284, 530], [355, 575]],
                                                    [[676, 363], [727, 430], [866, 279], [887, 370]],
                                                    [[726, 1124], [664, 1193], [934, 1174], [916,
1270]],
                                                    [[511, 1229], [352, 1388], [701, 1308], [564,
1530]],
                                                    [[14, 311], [152, 403], [153, 155], [319, 278]]]

hometown_surfaces = [[[300, 1344], [236, 1561], [619, 1348], [586, 1565]],
                                             [[1081, 827], [1065, 1147], [1267, 828], [1228, 1147]]]

computer_screen_surface = [[203, 634], [302, 864], [575, 690], [826, 903]]
computer_screen_surface_rectified = [[0, 0], [0, 300], [200, 0], [200, 300]]

flagellation_surface = [[627, 280], [626, 567], [676, 84], [676, 578]]
flagellation_surface_rectified = [[0, 0], [0, 200], [600, 0], [600, 200]]

my_pic_surface = [[0, 0], [0, 719], [719, 0], [719, 719]]

def affine_solve(u, v):
        X = []

        for j in range(u.shape[1]):
                X_temp = [[u[0,j], u[1,j], 1, 0, 0, 0],
                                    [0, 0, 0, u[0,j], u[1,j], 1]]
                X.extend(X_temp)

        X = np.array(X)
        y = v.T.flatten()
        h = LA.inv(X.T.dot(X)).dot(X.T).dot(y)
        h = np.concatenate((h, np.array([0, 0, 1])))
        H = h.reshape(3, 3)
        return H
```

```
def homography_solve(u, v):
    X = []

    for j in range(u.shape[1]):
        X_temp = [[u[0,j], u[1,j], 1, 0, 0, 0, -u[0,j]*v[0,j], -u[1,j]*v[0,j]],
                  [0, 0, 0, u[0,j], u[1,j], 1, -u[0,j]*v[1,j], -u[1,j]*v[1,j]]]
        X.extend(X_temp)

    X = np.array(X)
    y = v.T.flatten()
    h = LA.inv(X.T.dot(X)).dot(X.T).dot(y)
    h = np.concatenate((h, np.array([1])))
    H = h.reshape(3, 3)
    return H

def homography_transform(u, H):
    U = np.concatenate([u, np.ones([1, u.shape[1]])])
    V = H.dot(U)
    for j in range(V.shape[1]):
        V[:, j] = V[:, j] / V[2, j]
    v = V[:-1,:].astype(int)
    print(v)
    return v


if __name__ == "__main__":
    times_square_im = plt.imread(times_square)
    hometown_im = plt.imread(hometown)
    my_pic_im = plt.imread(my_pic)
    x_size, y_size, _ = my_pic_im.shape

    times_square_im_homo = np.copy(times_square_im)
    times_square_im_homo.setflags(write=1)
    for surface in times_square_surfaces:
        u = np.array(my_pic_surface).T
        v = np.array(surface).T
        H = homography_solve(u, v)
        U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
        V = homography_transform(U, H)

        for i in range(U.shape[1]):
            times_square_im_homo[V[0,i], V[1,i], :] = my_pic_im[U[0,i], U[1,i], :]

    plt.imsave("times_square_homography.jpg", times_square_im_homo, format="jpg")
    plt.imshow(times_square_im_homo)
    plt.show()

    hometown_im_homo = np.copy(hometown_im)
    hometown_im_homo.setflags(write=1)
    for surface in hometown_surfaces:
        u = np.array(my_pic_surface).T
        v = np.array(surface).T
        H = homography_solve(u, v)
        U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
        V = homography_transform(U, H)

        for i in range(U.shape[1]):
            hometown_im_homo[V[0,i], V[1,i], :] = my_pic_im[U[0,i], U[1,i], :]

    plt.imsave("Rach-Gia-City-worth-exploring_homography.jpg", hometown_im_homo,
format="jpg")
    plt.imshow(hometown_im_homo)
    plt.show()

    times_square_im_affine = np.copy(times_square_im)
    times_square_im_affine.setflags(write=1)
    for surface in times_square_surfaces:
        u = np.array(my_pic_surface[:-1]).T
        v = np.array(surface[:-1]).T
        H = affine_solve(u, v)
        U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
        V = homography_transform(U, H)

        for i in range(U.shape[1]):
            times_square_im_affine[V[0,i], V[1,i], :] = my_pic_im[U[0,i], U[1,i], :]

    plt.imsave("times_square_affine.jpg", times_square_im_affine, format="jpg")
    plt.imshow(times_square_im_affine)
    plt.show()

    hometown_im_affine = np.copy(hometown_im)
    hometown_im_affine.setflags(write=1)
    for surface in hometown_surfaces:
        u = np.array(my_pic_surface[:-1]).T
        v = np.array(surface[:-1]).T
        H = affine_solve(u, v)
        U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
        V = homography_transform(U, H)

        for i in range(U.shape[1]):
            hometown_im_affine[V[0,i], V[1,i], :] = my_pic_im[U[0,i], U[1,i], :]

    plt.imsave("Rach-Gia-City-worth-exploring_affine.jpg", hometown_im_affine, format="jpg")
    plt.imshow(hometown_im_affine)
    plt.show()

    #### Rectify computer screen and flagellation ####
```

```
	computer_screen_im = plt.imread(computer_screen)
	flagellation_im = plt.imread(flagellation)

	x_size, y_size = 200, 300
	computer_screen_im_rectified_homo = np.ones([x_size, y_size, 3], "float")
	u = np.array(computer_screen_surface_rectified).T
	v = np.array(computer_screen_surface).T
	H = homography_solve(u, v)
	U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
	V = homography_transform(U, H)

	for i in range(U.shape[1]):
		computer_screen_im_rectified_homo[U[0,i], U[1,i], :] = computer_screen_im[V[0,i],
V[1,i], :-1]

	plt.imsave("computer_screen_rectified_homography.jpg",
computer_screen_im_rectified_homo, format="jpg")
	plt.imshow(computer_screen_im_rectified_homo)
	plt.show()

	computer_screen_im_rectified_affine = np.ones([x_size, y_size, 3], "float")
	u = np.array(computer_screen_surface_rectified[:-1]).T
	v = np.array(computer_screen_surface[:-1]).T
	H = affine_solve(u, v)
	U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
	V = homography_transform(U, H)

	for i in range(U.shape[1]):
		computer_screen_im_rectified_affine[U[0,i], U[1,i], :] = computer_screen_im[V[0,i],
V[1,i], :-1]

	plt.imsave("computer_screen_rectified_affine.jpg", computer_screen_im_rectified_affine,
format="jpg")
	plt.imshow(computer_screen_im_rectified_affine)
	plt.show()


	x_size, y_size = 600, 200
	flagellation_im_rectified_homo = np.ones([x_size, y_size, 3], "uint8") * 255
	u = np.array(flagellation_surface_rectified).T
	v = np.array(flagellation_surface).T
	H = homography_solve(u, v)
	U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
	V = homography_transform(U, H)

	for i in range(U.shape[1]):
		flagellation_im_rectified_homo[U[0,i], U[1,i], :] = flagellation_im[V[0,i], V[1,i], :]
	plt.imsave("flagellation_rectified_homography.jpg", flagellation_im_rectified_homo,
format="jpg")
	plt.imshow(flagellation_im_rectified_homo)
	plt.show()

	x_size, y_size = 600, 200
	flagellation_im_rectified_affine = np.ones([x_size, y_size, 3], "uint8") * 255
	u = np.array(flagellation_surface_rectified[:-1]).T
	v = np.array(flagellation_surface[:-1]).T
	H = affine_solve(u, v)
	U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
	V = homography_transform(U, H)

	for i in range(U.shape[1]):
		flagellation_im_rectified_affine[U[0,i], U[1,i], :] = flagellation_im[V[0,i], V[1,i], :]

	plt.imsave("flagellation_rectified_affine.jpg", flagellation_im_rectified_affine, format="jpg")
	plt.imshow(flagellation_im_rectified_affine)
	plt.show()
```