Computer Vision
Problem Set 1

Name: 王官皓　　ID:1140329057

---

1. (1) Comparing with the **affine transform**, describe the invariants of **homography**.

Solution: **Affine** transformation preserves

†. parallel lines

††. ratio of lengths of two parallel line segments

†††. ratio of area

while **homography(perspective)** transformation preserves

††††. ***cross ratio*** of four collinear points

Proof: Given Affine transformation

$$\mathrm{x}' = H_A\mathrm{x} = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \overline{\mathrm{x}}$$

and Homography transformation

$$\mathrm{x}' = H_P\mathrm{x} = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{v}^T & v \end{bmatrix} \overline{\mathrm{x}}$$

where $\overline{\mathrm{x}}$ denotes the augmented vector w.r.t. world coordinates.

†. Rewrite the transform equation as $\mathrm{x}\prime = A\overline{\mathrm{x}} + \mathbf{t}$, suppose that $l : \mathbf{p} + s\mathbf{u}$ and $m : \mathbf{q} + s\mathbf{v}$,$t \in \mathbb{R}$, are parallel lines. Then $\mathbf{v} = k\mathbf{u}$ for some $k \in \mathbb{R}$. Therefore

$$\begin{aligned} H_A(\mathbf{p} + s\mathbf{u}) &= A(\mathbf{p} + s\mathbf{u}) + \mathbf{t} = (A\mathbf{p} + \mathbf{t}) + s(A\mathbf{u}) = \mathbf{p}_1 + s\mathbf{u}_1 \\ H_A(\mathbf{q} + s\mathbf{v}) &= A(\mathbf{q} + s(k\mathbf{u})) + \mathbf{t} = (A\mathbf{q} + \mathbf{t}) + s(Ak\mathbf{u}) = \mathbf{q}_1 + s(k\mathbf{u}_1) \end{aligned}$$

That is, l adn m are mapped to lines l_1 and m_1 that are parallel. ■

††. Consider parallel line segments, S_1 and S_2, given in vector form as $S_i : \mathbf{p}_i + s\mathbf{u}_i, t \in [0, 1]$. Because they are parallel, $\mathbf{u}_2 = k\mathbf{u}_1, k \in \mathbb{R}$. As $|\mathbf{u}_i|$ is the length of S_i, the ratio of lengths of S_2 and S_1 is $|k|$. From †, S_i is mapped into a segment of length $|A\mathbf{u}_i|$. Since $A\mathbf{u}_2 = A(k\mathbf{u}_1) = k(A\mathbf{u}_1), |A\mathbf{u}_2| = |k||A\mathbf{u}_1|$, which shows that the ratio of lengths of $H_A(S_1)$ and $H_A(S_1)$ is also $|k|$. ■

† † †. W.L.O.G, consider two **triangles** on a 2D plane with vertices

$$A_i = (x_{i1}, y_{i1}), B_i = (x_{i2}, y_{i2}), C_i = (x_{i3}, y_{i3}), i = 1, 2$$

from linear algebra theorems, the area of a triangle is

$$S_{\Delta A_i B_i C_i} = \frac{1}{2} \begin{vmatrix} x_{i1} & y_{i1} & 1 \\ x_{i2} & y_{i2} & 1 \\ x_{i3} & y_{i3} & 1 \end{vmatrix}$$

Suppose $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $\mathbf{t} = [t_1, t_2]^T$, after affine transform, a point (x, y) becomes $(ax + by + t_1, cx + dy + t_2)$. Therefore, the area of a triangle becomes

$$\begin{aligned} S_{\Delta A'_i B'_i C'_i} &= \frac{1}{2} \begin{vmatrix} ax_{i1} + by_{i1} + t_1 & cx_{i1} + dy_{i1} + t_2 & 1 \\ ax_{i2} + by_{i2} + t_1 & cx_{i2} + dy_{i2} + t_2 & 1 \\ ax_{i3} + by_{i3} + t_1 & cx_{i3} + dy_{i3} + t_2 & 1 \end{vmatrix} \\ &= \frac{1}{2} \begin{vmatrix} x_{i1} & y_{i1} & 1 \\ x_{i2} & y_{i2} & 1 \\ x_{i3} & y_{i3} & 1 \end{vmatrix} \cdot \begin{vmatrix} a & c & 0 \\ b & d & 0 \\ t_1 & t_2 & 1 \end{vmatrix} \\ &= S_{\Delta A_i B_i C_i} \cdot (ad - bc) \end{aligned}$$

The term $(ad - bc)$, i.e. |A|, can be canceled out when we calculate the ratio of the areas of two triangles. Therefore, the ratio of two triangles will be an invariant in Affine transformation. This conclusion can be generalized to **polygons** and **arbitrary closed regions**, as well as higher dimension cases. ■

† † ††. W.L.O.G, consider Projective transformation in $\mathcal{P}\mathbb{R}^2$

Definition(cross ratio): Given four collinear points

$$A = (x_1, y_1), B = (x_2, y_2), C = (x_3, y_3), D = (x_4, y_4)$$

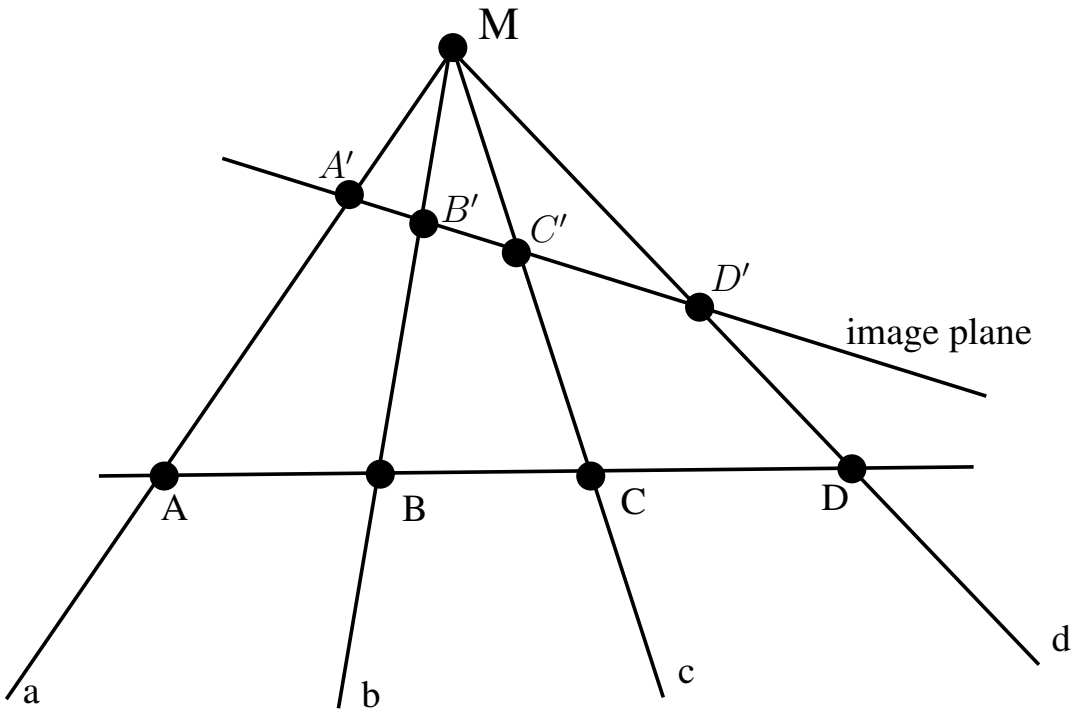
the **cross ratio** is defined as

$$(A, B; C, D) = \frac{|AC| \cdot |BD|}{|AD| \cdot |BC|}$$

Given four concurrent lines a, b, c, d intersecting on point M, the cross ratio of four concurrent lines is defined as

$$(a, b; c, d) = \frac{\sin(\angle cMa) \cdot \sin(\angle dMb)}{\sin(\angle cMb) \cdot \sin(\angle dMa)}$$

The Projective transformation in $\mathcal{P}\mathbb{R}^2$ can be illustrated in the figure below, four collinear points A, B, C, D is transformed to points A', B', C', D'



Since

$$\begin{aligned} S_{\Delta CMA} &= \tfrac{1}{2}h \cdot |CA| = \frac{1}{2}|MC| \cdot |MA| \cdot \sin(cMa) \\ S_{\Delta CMB} &= \tfrac{1}{2}h \cdot |CB| = \frac{1}{2}|MC| \cdot |MB| \cdot \sin(cMb) \\ S_{\Delta DMA} &= \tfrac{1}{2}h \cdot |DA| = \frac{1}{2}|MD| \cdot |MA| \cdot \sin(dMa) \\ S_{\Delta DMB} &= \tfrac{1}{2}h \cdot |DB| = \frac{1}{2}|MD| \cdot |MB| \cdot \sin(dMb) \end{aligned}$$

the identity folows immediately

$$(a, b; c, d) = (A, B; C, D)$$

moreover

$$(A, B; C, D) = (A', B'; C', D')$$

holds as a corollary of the identity. ■

(2) Given a homography:$H = \begin{bmatrix} 1.707 & 0.586 & 1.0 \\ 2.707 & 8.242 & 2.0 \\ 1.0 & 2.0 & 1.0 \end{bmatrix}$, decompose it into a chain of similarity, affine and projective transformations orderly from left to right.(Hint: $H = H_S H_A H_P = \begin{bmatrix} sR & \mathbf{t}/v \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} K & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} I & \mathbf{0} \\ \mathbf{v}^T & v \end{bmatrix} = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{v}^T & v \end{bmatrix}$)

Solution: Suppose that

$$H = H_S H_A H_P = \begin{bmatrix} sR & \mathbf{t}/v \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} K & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} I & \mathbf{0} \\ \mathbf{v}^T & v \end{bmatrix} = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{v}^T & v \end{bmatrix}$$

By matrix multiplication, we have

$$\begin{aligned} \mathbf{t} &= [1.0, 2.0]^T \\ \mathbf{v}^T &= [1.0, 2.0] \\ v &= 1.0 \\ A &= sRK + \mathbf{t}\mathbf{v}^T/v \end{aligned}$$

where

$$\begin{aligned} R &= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \\ K &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \end{aligned}$$

Therefore

$$\begin{aligned} sRK &= \begin{bmatrix} 1.707 & 0.586 \\ 2.707 & 8.242 \end{bmatrix} - \begin{bmatrix} 1.0 & 2.0 \\ 2.0 & 4.0 \end{bmatrix} \\ &= \begin{bmatrix} 0.707 & -1.414 \\ 0.707 & 4.242 \end{bmatrix} \end{aligned}$$

from the multiple relations between each entry of the matrix and their numerical values, we can make the inference that $\theta = \pi/4, s = 1$, each entry of K can be solved w.r.t

$$\begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 0.707 & -1.414 \\ 0.707 & 4.242 \end{bmatrix}$$

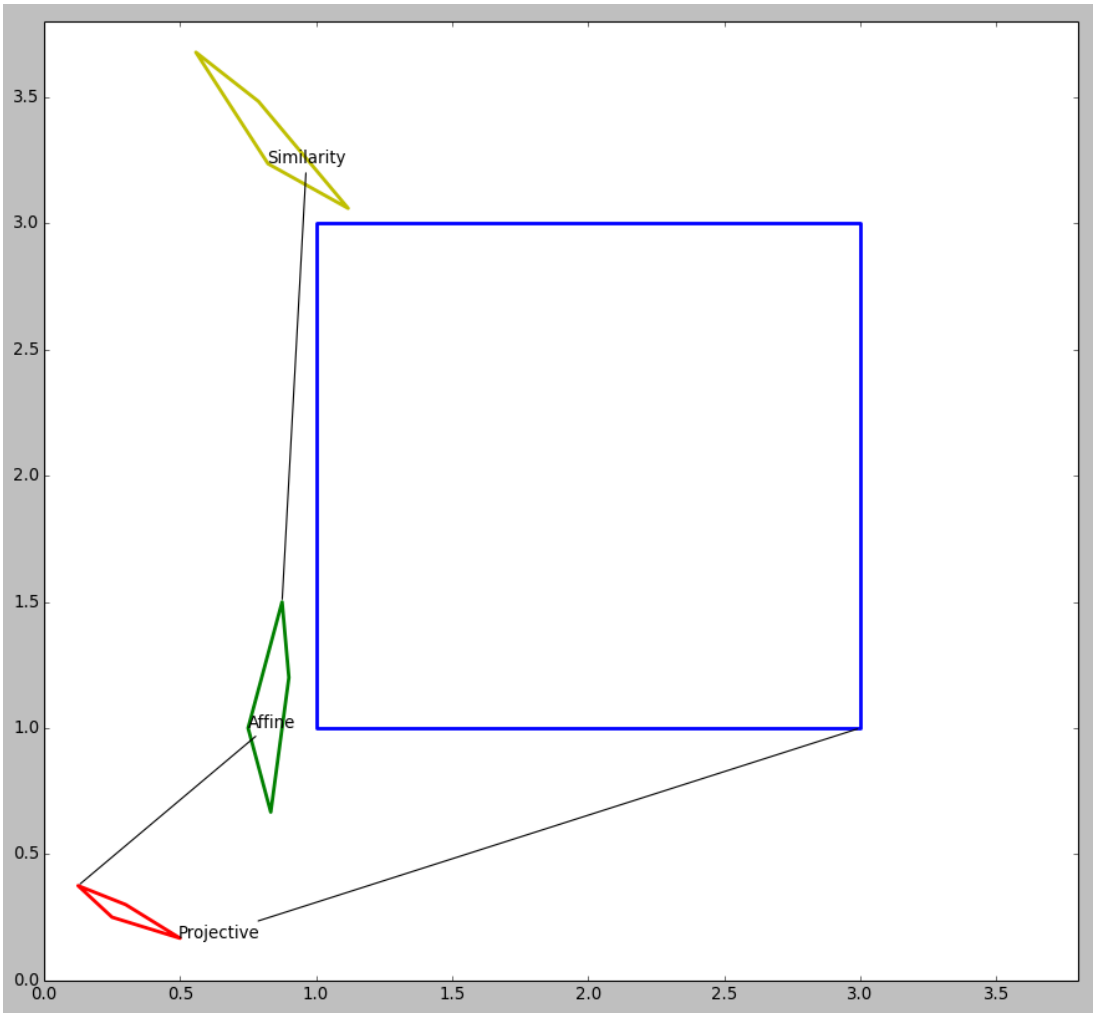
which gives that $a = 1, b = 2, c = 0, d = 4$, therefore

$$H = \begin{bmatrix} 0.707 & -0.707 & 1.0 \\ 0.707 & 0.707 & 2.0 \\ 0 & 0 & 1.0 \end{bmatrix} \begin{bmatrix} 1.0 & 2.0 & 0 \\ 0 & 4.0 & 0 \\ 0 & 0 & 1.0 \end{bmatrix} \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 1.0 & 0 \\ 1.0 & 2.0 & 1.0 \end{bmatrix}$$

which can be verified conveniently.

(3) Given a rectangle R with four vertices $[(1,1);(1,3);(3,3);(3,1)]$ in 2D plane, plot the transformed figures one by one under the transformation chain obtained in (2).

Solution: The following figure clearly reveal the transformation process step by step(Projective→Affine→Similarity).



2. Prove that a 2D Gaussian filter can be extended by two 1D Gaussian.

Proof 1D Gaussian filter's general formula is

$$G(x;\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

and expression for 2D Gaussian

$$G(x,y;\sigma) = \frac{1}{2\pi\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

which could be rewitten as

$$G(x,y;\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{y^2}{2\sigma^2}} = G(x;\sigma)G(y;\sigma)$$

which means 2D filter window is separated into a couple of 1D ones(**separability**). In practice it means that to apply Gaussian filter to an image it is enough to filter it in horizontal direction with 1D Gaussian and then filter the result with the same Gaussian in vertical direction. ■

3. Given an 8×8 image $f(x, y)$ and a 3×3 filter $h(x, y)$, shown respectively as below, calculate the convolution and correlation respectively. Label the value with green color in the result images that corresponds to the source light blue pixels.

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.2	0.1

Solution: Suppose that $\otimes$ denote the correlation operation, and $\star$ denote the convolution operation, with operands $f(x, y)$ and $w(x, y)$ denote the image and filter, respectively.

Given a 3×3 filter

$w(-1,-1)$	$w(-1,0)$	$w(-1,1)$
$w(0,-1)$	$w(0,0)$	$w(0,1)$
$w(1,-1)$	$w(1,0)$	$w(1,1)$

the definition of correlation is

$$w(x, y) \otimes f(x, y) = \sum_{s=-1}^{1} \sum_{t=-1}^{1} w(s, t) f(x + s, y + t)$$

while the definition of convolution is

$$w(x, y) \star f(x, y) = \sum_{s=-1}^{1} \sum_{t=-1}^{1} w(s, t) f(x - s, y - t)$$

To simplify the notation, we can only use the correlation equation, since convolution can be implemented equivently by rotating $w(x, y)$ by $180°$. OpenCV2 provides a function cv2.filter2D() to apply[1] a filter(kernel) to an image, which could be used to solve the problem. Before the correlation or convolution, the image is padded with 1-pixel width border, and each pixel is expanded to 50×50 square region for ease of visualization the result.

Correlation result

30.7	53.7	76.7	95.4	102.7	104.8	106.3	80.
42.1	75.	104.1	127.2	137.1	141.5	145.6	107.6
42.7	74.6	101.	121.	130.8	138.	145.4	107.9
42.8	72.	94.4	113.2	125.2	136.1	145.	107.9
42.	67.2	84.9	102.6	117.2	131.3	141.9	106.7
41.	62.	74.4	90.	106.	122.1	135.2	102.9
40.2	58.	65.2	76.7	91.5	108.4	124.2	97.
25.	36.	39.	44.9	53.8	64.4	75.2	58.3

[1] cv2.filter2D() does actually compute correlation, though the brief description in OpenCV2 documentation claims that it "Convolves an image with the kernel", which is a little tricky.;-)

Convolution result

26.1	47.2	66.9	83.1	90.1	92.	93.2	66.7
41.9	74.5	104.3	128.4	138.4	142.5	145.8	107.2
42.6	74.8	101.7	122.6	132.1	138.6	144.7	107.8
42.5	72.6	96.	115.	126.1	136.1	145.2	108.2
41.8	68.2	87.2	105.	118.8	132.2	142.9	106.8
41.	62.9	76.6	92.7	108.6	124.2	136.9	103.7
40.1	58.3	66.8	79.2	94.3	111.1	126.9	98.3
30.	41.	44.8	51.9	62.2	74.6	86.8	70.9

Visualized Results

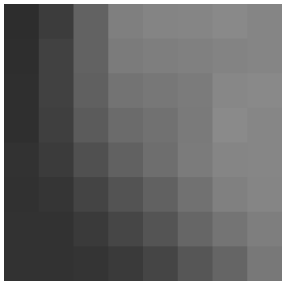


Figure 1: Origin

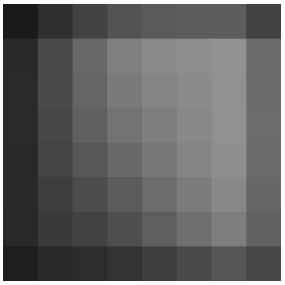


Figure 2: Convolution

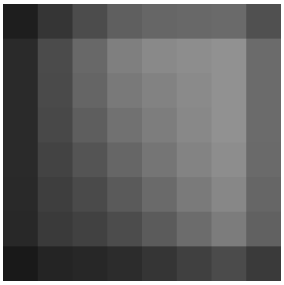
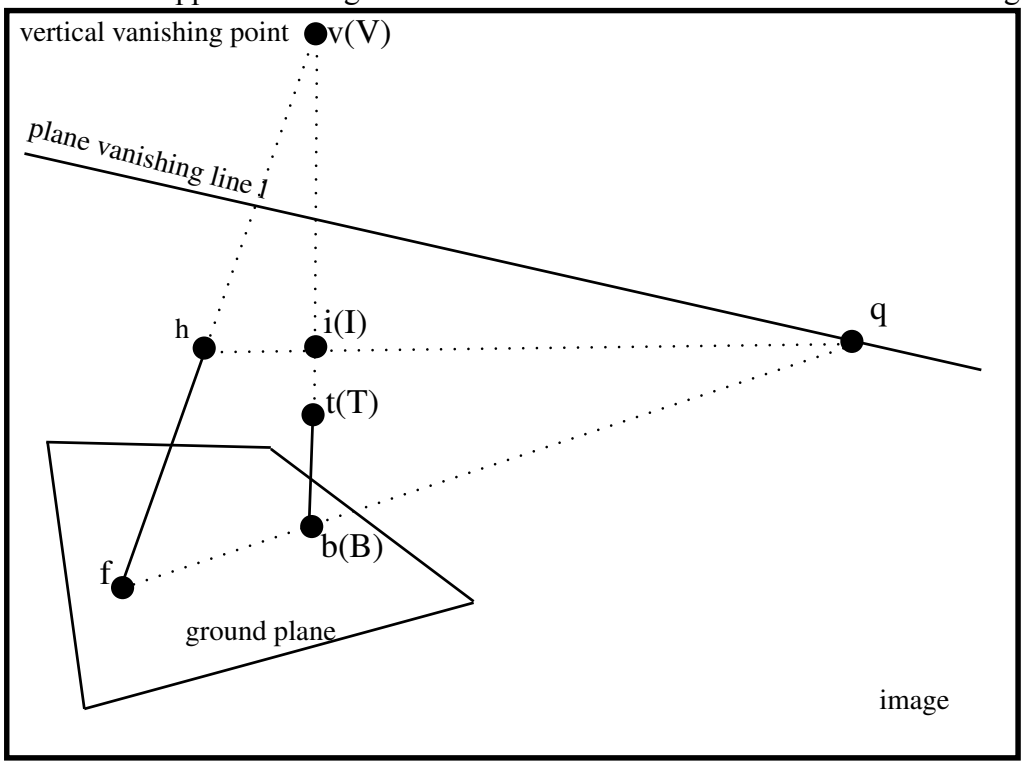


Figure 3: Correlation

4. Given the vanishing line of the ground plane l, the vertical vanishing point v, and a known reference height in a scene (e.g. the height of a cabinet), design an algorithm to compute the height of a person standing by the cabinet according to the image of this scene in which this person's foot and head are visible.

Solution: Suppose an image taken from a camera is illustrated in the following figure.



The basic geometry involved is illustrated as it appears in the image, the line l is the plane vanishing line, and v the vertical vanishing point, which is the image of point V at infinite distance from the ground plane in the vertical direction[2]. Image of reference object (segment tb) and the image of the person to be measured (segment hf) are also shown. The intersection of the line through the two base points f and b with the vanishing line l is the point q, and the intersection of the line through h and q with the line through v and b defines the point i. The point i is the image of a point I in the world at distance IB from the reference plane, which is the height of the person.(This follows because the line fq and hq are parallel in the scene[3].)

From 1. (1). †††, a permuted version[4] of cross-ratio of four collinear points is projective invariant. In real world coordinates,

$$(I,T;B,V) = \frac{|I-B|}{|T-B|} \cdot \frac{|V(\infty)-T|}{|V(\infty)-I|} = \frac{|I-B|}{|T-B|}$$

In image coordinates,

$$(i,t;b,v) = \frac{|i-b|}{|t-b|} \cdot \frac{|v-t|}{|v-i|}$$

using projective invariant,

$$\frac{|I-B|}{|T-B|} = \frac{|i-b|}{|t-b|} \cdot \frac{|v-t|}{|v-i|}$$

Therefore, the height of the person can be calculated

$$|I-B| = |T-B| \cdot \frac{|i-b|}{|t-b|} \cdot \frac{|v-t|}{|v-i|}$$

since the reference height is already known and other terms can be obtained from the image.

5. Find a method to remove the underlines in text documents; assuming that the documents had been well aligned with the images coordinates and thinned to skeletons of a single pixel width.

~~The Techniques of computer vision~~

Solution: Morphological transformations can deal with the problem in most cases.

Step1: Firstly, convert the image into an invert binary image. (the choice of threshold is pretty easy, since the text is well-distinguished from the background, e.g. 127 is OK to guarantee a good result.)

[2]The lower-case v denotes the image point, and the upper-case V denotes the point in real world coordinates, the same notation will also be used later for point i, t and b.

[3]Another perceptual method to catch the point is to imagine that the person walk to the base point B, e.g. the foot of the cabinet, thus the vertical vanishing point V, the head point of the person I,the top point of the cabinet T and the base point B are four collinear points.

[4] There is a number of definitions of the cross ratio. However, they all differ from each other by a suitable permutation of the coordinates. There are $4! = 24$ different orders, but only 6 distinct values, since some permutations leave the cross-ratio unaltered. In fact, exchanging any two pairs of coordinates preserves the cross-ratio, e.g.:

$$(v,i;t,b) = (i,v;b,t) = (t,b;v,i) = (b,t;i,v)$$

each of the definitions is equivent for cross-ratio invariant, here $(I,T;B,V)$ and $(i,t;b,v)$ are used.

Step2: Then an **erosion** operation is applied to the image, in order to detect the horizontal line, using the **Structuring Element(SE)** below

1	1	1	...	1	1	1

the horizontal line is shown in the following figure

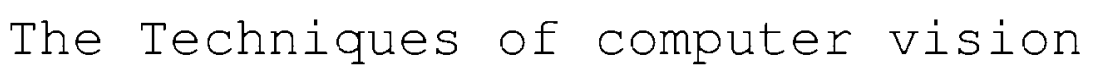
Step3: Substract the invert binary image by the detected line.

Step4: In order to connect the gap, a **dilation** is applied to the image obtained in **Step3**, using the SE below

1	1	1
1	1	1
1	1	1

Step5: Apply **erosion** again to shrink the text, using the same SE in **Step4**

Step6: Invert the image



However, when the horizontal line is overlapped with some part of the character, e.g. 'e', Morphological method cannot preserve the overlapped part of the character. Some prior knowledge need to be involved to recover the overlapped horizontal part of the text (e.g. character detection & recognition).

Appendix-Python Codes

1. (3) Transformation Visulization

```
#!/usr/bin/env python
# encoding: utf-8

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon

S = np.array([[0.707, -0.707, 1.0],
              [0.707,  0.707, 2.0],
              [0,          0, 1.0]])
A = np.array([[1.0, 2.0,   0],
              [0,   4.0,   0],
              [0,     0, 1.0]])
P = np.array([[1.0,   0,   0],
              [0,   1.0,   0],
              [1.0, 2.0, 1.0]])

# Original Rectangle
plt.figure()
augVerts = np.array([[1, 1, 1],
                     [1, 3, 1],
                     [3, 3, 1],
                     [3, 1, 1]])
origin = Polygon(augVerts[:, 0:2], closed=True,
                 fill=False, ec="b", lw=2.5)
plt.xlim(0, 3.8)
plt.ylim(0, 3.8)
plt.gca().add_patch(origin)

# Projective
Proj = np.dot(P, augVerts.transpose())
Proj = Proj / Proj[2]
ProjRect = Polygon(Proj.transpose()[:, 0:2], closed=True,
                   fill=False, ec="r", lw=2.5)
plt.gca().add_patch(ProjRect)
plt.annotate("Projective",
             (augVerts[3, 0], augVerts[3, 1]),
             (Proj[0, 3], Proj[1, 3]),
             arrowprops=dict(arrowstyle="-"))

# Affine
```

```
Affine = np.dot(A, Proj)
Affine = Affine / Affine[2]
AffineRect = Polygon(Affine.transpose()[:, 0:2],
                     closed=True, fill=False, ec="g", lw=2.5)
plt.gca().add_patch(AffineRect)
plt.annotate("Affine",
             (Proj[0, 1], Proj[1, 1]),
             (Affine[0, 0], Affine[1, 0]),
             arrowprops=dict(arrowstyle="-"))

#  Similarity Transform
Simi = np.dot(S, Affine)
Simi = Simi / Simi[2]
SimiRect = Polygon(Simi.transpose()[:, 0:2], closed=True,
                   fill=False, ec="y", lw=2.5)
plt.gca().add_patch(SimiRect)
plt.annotate("Similarity",
             (Affine[0, 1], Affine[1, 1]),
             (Simi[0, 0], Simi[1, 0]),
             arrowprops=dict(arrowstyle="-"))
plt.show()
```

3. Linear filter

```
#!/usr/bin/env python
# encoding: utf-8
import cv2
import numpy as np

def visImage(originImg):
    # for visualization
    visImage = np.zeros((400, 400), dtype=np.uint8)
    for i in np.arange(0, 8):
        for j in np.arange(0, 8):
            visImage[50*i:50*i+50,50*j:50*j+50]=originImg[i, j]
    return visImage


def main():
    image = np.array([
                      [45, 60, 98, 127, 132, 133, 137, 133],
                      [46, 65, 98, 123, 126, 128, 131, 133],
                      [47, 65, 96, 115, 119, 123, 135, 137],
                      [47, 63, 91, 107, 113, 122, 138, 134],
                      [50, 59, 80,  97, 110, 123, 133, 134],
```

```
                        [49, 53, 68,  83,  97, 113, 128, 133],
                        [50, 50, 58,  70,  84, 102, 116, 126],
                        [50, 50, 52,  58,  69,  86, 101, 120]
                        ], dtype=np.float)
    # padded border
    borderImage = cv2.copyMakeBorder(image,
                                     1, 1, 1, 1,
                                     cv2.BORDER_CONSTANT,
                                     0)
    # kernel definition
    corr_kernel = np.array([[0.1, 0.1, 0.1],
                            [0.1, 0.2, 0.1],
                            [0.1, 0.2, 0.1]])
    conv_kernel = cv2.flip(corr_kernel, -1)
    # filter the image
    corr_Image = cv2.filter2D(borderImage, -1, corr_kernel)
    conv_Image = cv2.filter2D(borderImage, -1, conv_kernel)
    # remove the border
    corr_Image = corr_Image[1:9, 1:9]
    conv_Image = conv_Image[1:9, 1:9]
    # Output results
    print corr_Image
    print conv_Image
    # Visulization
    visualOrig = visImage(image)
    visualCorr = visImage(corr_Image)
    visualConv = visImage(conv_Image)
    cv2.imwrite("visualOrig.png", visualOrig)
    cv2.imwrite("visualCorr.png", visualCorr)
    cv2.imwrite("visualConv.png", visualConv)

if __name__ == '__main__':
    main()
```

5. Remove Ruleline in Text Documents

```
#!/usr/bin/env python
# encoding: utf-8

import cv2
import numpy as np

img = cv2.imread('ocr.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
im_bw = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV)[1]
```

```
10 line_kernel = np.ones((1, im_bw.shape[1]/2), dtype=np.uint8)
11 x = cv2.morphologyEx(im_bw,
12                       cv2.MORPH_OPEN,
13                       line_kernel,
14                       iterations=1)
15 im_bw -= x
16 dil_kernel = np.ones((3, 3), dtype=np.uint8)
17 erd_kernel = np.ones((3, 3), dtype=np.uint8)
18 im_bw = cv2.dilate(im_bw, dil_kernel, iterations=1)
19 im_bw = cv2.erode(im_bw, erd_kernel, iterations=1)
20 im_bw = cv2.threshold(im_bw, 127, 255, cv2.THRESH_BINARY_INV)[1]
21 cv2.imshow('result', im_bw)
22 cv2.waitKey(0)
```