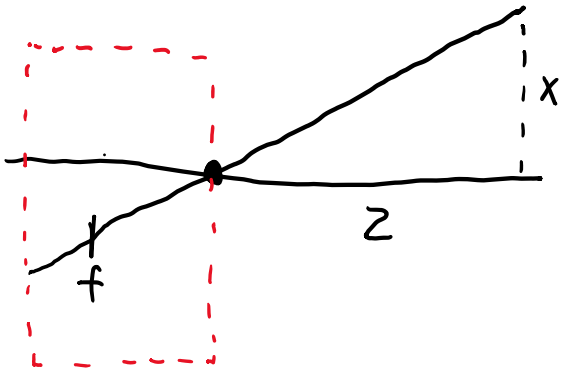
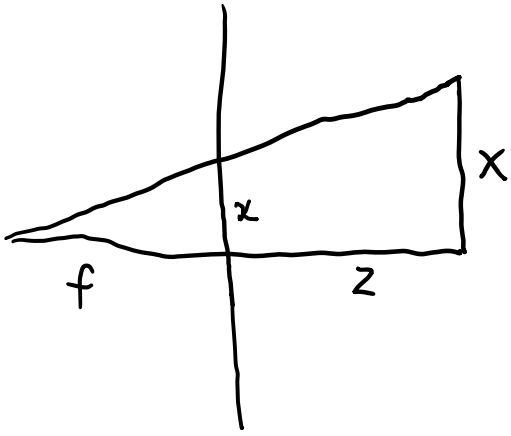
Problem 1:

Camera's Physical mode



Camera's Virtual Mode



Similar Triangles

$$x' = f\frac{x}{z}$$

$$y' = f\frac{y}{z}$$

Given $f = 16\,mm = 0.016\,m$

a.) $(x, y, z) = (2, 3, 4)$

$$x' = (0.016)\frac{(2)}{(4)} = 0.008\,m$$

$$y' = (0.016)\frac{(3)}{(4)} = 0.012\,m$$

$$(x, y) = (0.008, 0.012)\,m$$

b.) $(x, y, z) = (4, 6, 8)$

$$x' = (0.016)\frac{(4)}{(8)} = 0.008\,m$$

$$y' = (0.016)\frac{(6)}{(8)} = 0.012\,m$$

$$(x, y) = (0.008, 0.012)\,m$$

c.) $(X, Y, Z) = (-1, -2, 8)$

$X' = (0.016)\frac{(-1)}{(8)} = -0.002\,m$

$Y' = (0.016)\frac{(-2)}{(8)} = -0.004\,m$

$\boxed{(X, Y) = (-0.002, 0.004)\,m}$

d.) $(X, Y, Z) = (0, 0, 4)$

$X' = (0.016)\frac{(0)}{(4)} = 0\,m$

$Y' = (0.016)\frac{(0)}{(4)} = 0\,m$

$\boxed{(X, Y) = (0, 0)\,m}$

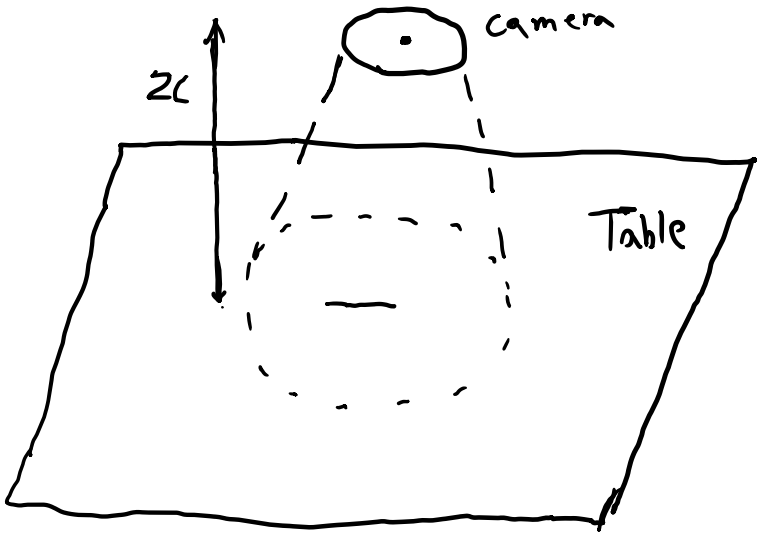
e.) $(0,0)$ $(0.008, 0.012)$ $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

$\sqrt{(0.008 - 0)^2 + (0.012 - 0)^2} = \sqrt{.000208} = \boxed{0.014\,m}$

f.) $(0,0)$ $(0.008, 0.012)$

$\sqrt{(0.008 - 0)^2 + (0.012 - 0)^2} = \boxed{0.014\,m}$

Problem 2:



Perspective projection is a linear projection where three dimensional objects are projected on a picture plane. The objects which are closer to the camera appears larger than the objects which are away from the camera. Since the distance of the table from the camera's point of projection is fixed. Its length will remain unchanged in the picture. Changing the location of the nail on the table does not change the distance between the nail and the camera; therefore, the length of the nail as it appears in the image does not depend on the nail's location on the table.

Problem 3:

$$g = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad h = \begin{bmatrix} 1 & -1 \\ 2 & -2 \end{bmatrix}$$

find f so that, $(I * g) * h = I * f$

convolution is associative,

$\therefore (I*g)*h = I*(g*h)$

$= I*f$ so $f = (g*h)$

flip filter (bottom to top, right to left)

Then perform cross-correlation

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{bmatrix} -2 & 2 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array} \begin{bmatrix} 1 & 2 & 0 \\ 3 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -8 & 2 & 6 \\ -8 & 3 & 6 \\ -2 & 1 & 1 \end{bmatrix}$$

$f(0,0) = (-2\cdot 4) + (2\cdot 0) + (-1\cdot 0) + (1\cdot 0) = -8$

$f(0,1) = (-2\cdot 3) + (4\cdot 2) + (0\cdot 0) + (4\cdot 0) + (1\cdot 0) + (0\cdot 0) = 2$

$f(0,2) = (3\cdot 2) + (0\cdot 4) + (1\cdot 0) + (0\cdot 0) = 6$

$f(1,0) = (-2\cdot 2) + (2\cdot 0) + (-1\cdot 4) + (1\cdot 0) + (0\cdot 0) + (0\cdot 0) = -8$

$f(1,1) = (-2\cdot 1) + (2\cdot 2) + (0\cdot 0) + (3\cdot -1) + (4\cdot 1) + (0\cdot 0) + (0\cdot 0) + (0\cdot 0) + (0\cdot 0) = 3$

$f(1,2) = (2\cdot 1) + (0\cdot 0) + (4\cdot 1) + (0\cdot 0) + (0\cdot 0) + (0\cdot 0) = 6$

$f(2,0) = (2 \cdot -1) + (0 \cdot 1) + (4 \cdot 0) + (0 \cdot 0) = -2$

$f(2,1) = (1 \cdot -1) + (2 \cdot 1) + (0 \cdot 0) + (3 \cdot 0) + (4 \cdot 0) + (0 \cdot 0) = 1$

$f(2,2) = (1 \cdot 1) + (2 \cdot 0) + (3 \cdot 0) + (4 \cdot 0) = 1$

$$\therefore f = \begin{bmatrix} -8 & 2 & 6 \\ -8 & 3 & 6 \\ -2 & 1 & 1 \end{bmatrix}$$

Problem 4:

The camera center and the given line (to which the other 2 lines are parallel) defines a plane. On the other hand, the image projection plane is, z = f.

The intersection of these two planes is the line of image projection parametrized by t (which is scaled by z=f):

$$(x(t), y(t)) = f\left(\frac{x_0 + at}{z_0 + ct}, \frac{y_0 + bt}{z_0 + ct}\right)$$

If c is not equal to 0, then the vanishing point is given by

$$(x_v, y_v) = (x(t), y(t))\Big|_{t \to \infty} = f\left(\frac{a}{c}, \frac{b}{c}\right)$$

However, if c=0, the vanishing point is at infinity.

Problem 5:

a)

```
import numpy as np #importing the numpy module into the short name np

a = np.array([[1,2,3], [4,5,6], [7,8,9]]) #creates a 3x3 matrix
b = a[2,:] #since python starts from a zero index, this will set be equal to the
3rd row of the a matix
c = a.reshape(-
1) #c will contain a single row of all the unchanged elements of matrix a because
 we specified an unknown
                    #number of rows
f = np.random.randn(5,1) #create a 5x1 matrix filled with random values as per st
andard normal distribution.
g = f[f>0] #g will only be a 3x1 matrix because it will only store values greater
 than 0 from f's matrix
x = np.zeros(10)+0.5 #x will be an array of 10 "padded" zeroes that also have 0.5
 added to each element (10 entries
                        # of 0.5)
y = 0.5*np.ones(len(x)) #y will be an array the same length as x but padded with
1s instead. Each element is then
                        # multiplied by 0.5
z = x + y               #matrix addition of x and y are stored in z (An array of
10 1s)
a = np.arange(1,100) #creates an array of values from 1 to 99 with even spacing w
ith the default step size of 1
b = a[::-1] #Starting from the back of the array a, copy all values into b
c = np.random.permutation(10) #creates a randomly permuted array with a range of
10
```

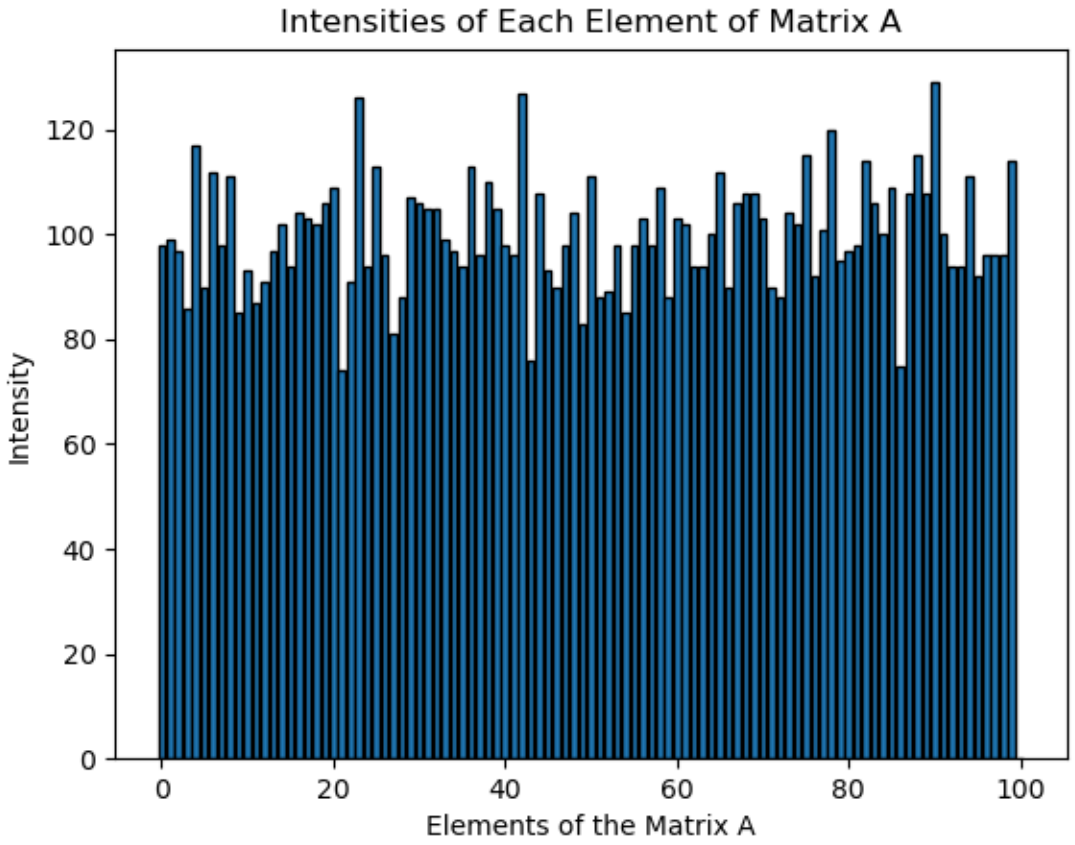
b)

```
import numpy as np

y = np.array([1, 2, 3, 4, 5, 6])
#print(y)

z = y.reshape(3,2)
#print(z)

r = np.where( z == np.max(z) )[0]
#print(r)
c = np.where( z == np.max(z) )[1]
#print(c)
```

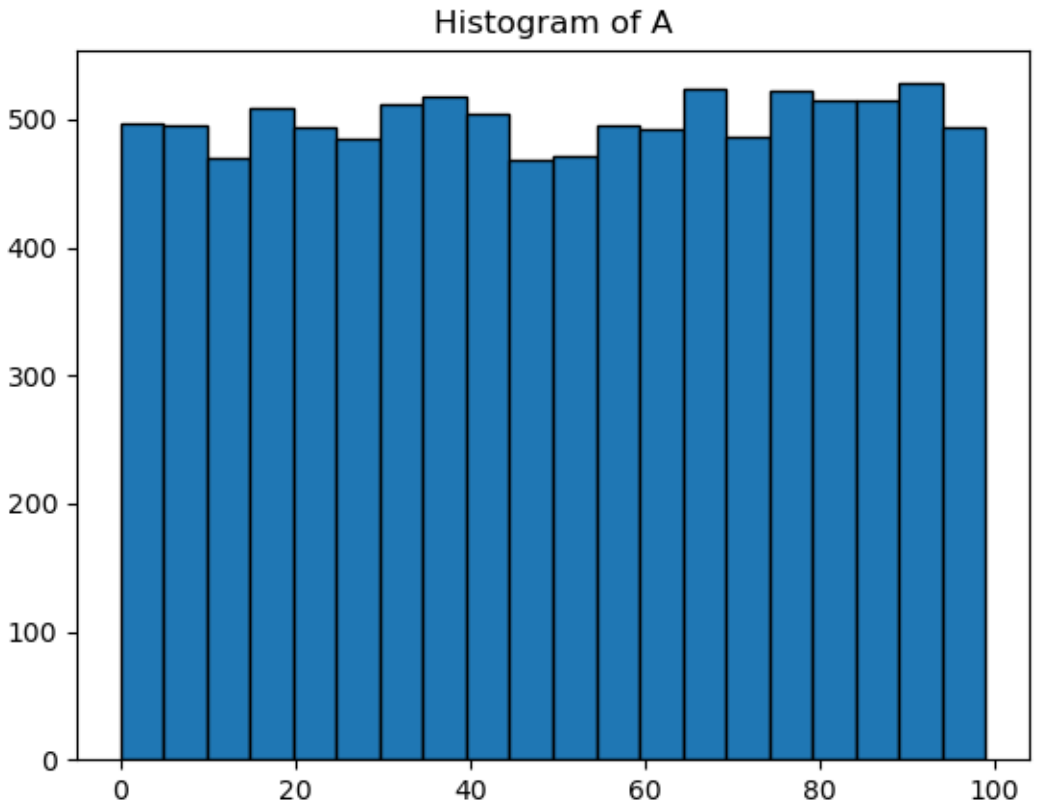
```
x = np.max(z)
#print(x)

v = np.array([1, 8, 8, 2, 1, 3, 9, 8])
x = np.count_nonzero(v == 1)
#print(x)

dice = np.random.randint(1,7, size = 12)
#print(dice)
```

c)

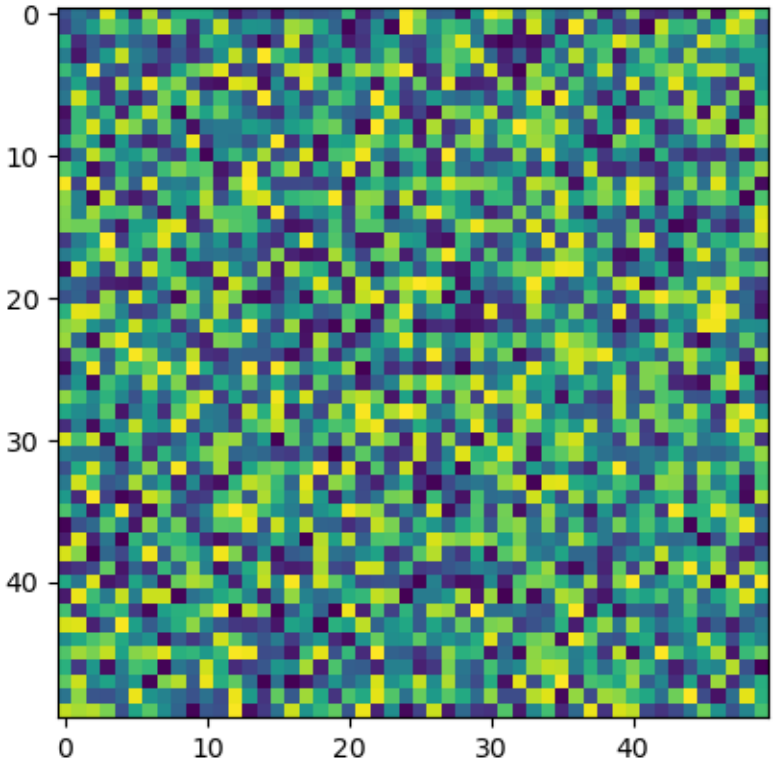
1.



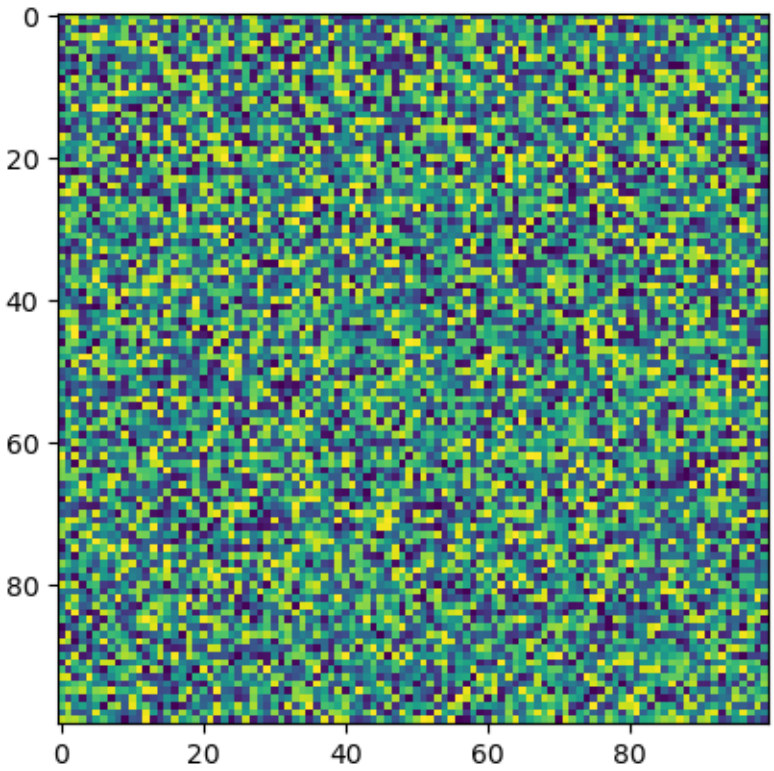
2.



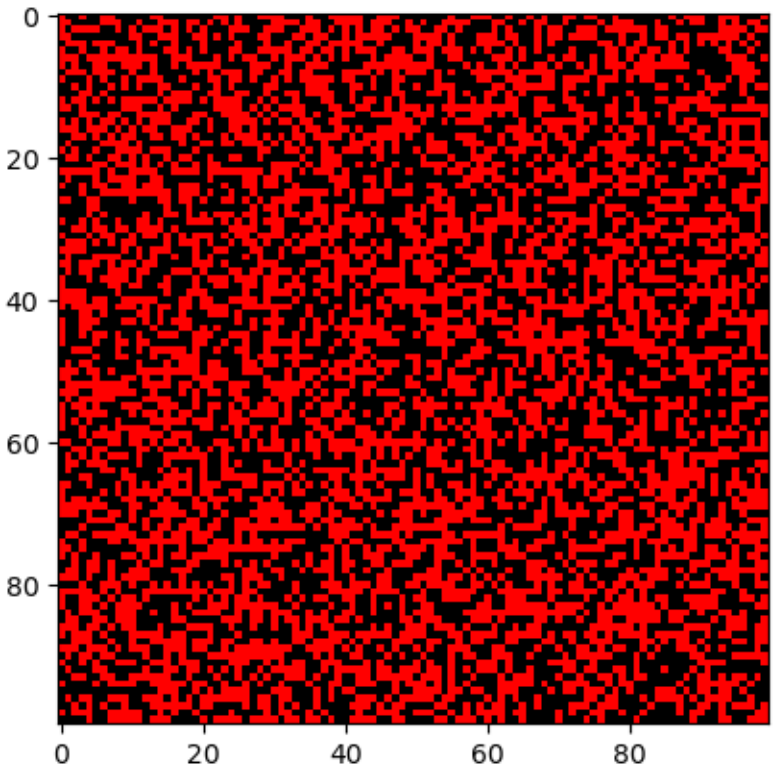
3.



4.



5.



Problem 6:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import scipy.stats as st

def dnorm(x, y, sd):
    return 1 / (np.sqrt(2 * np.pi) * sd) * np.e ** (-
np.power((x - y) / sd, 2) / 2)


def gauss_kern(kernlen, sigma):
    """Returns a 2D Gaussian kernel."""
    kern_1D = np.linspace(-(kernlen // 2), kernlen // 2, kernlen )
    for i in range(kernlen):
        kern_1D[i] = dnorm(kern_1D[i], 0, sigma)
    kern_2D = np.outer(kern_1D.T, kern_1D.T)
    kern_2D *= 1.0 / kern_2D.max()

    return kern_2D


#Load an image in grayscale
img = cv2.imread('inputP6.jpg', cv2.IMREAD_GRAYSCALE)

plt.figure("Before Gaussian Filtering")
plt.imshow(img, cmap = 'Greys_r')
#print(img.shape)


ksize = 5
denominator = 273.0

#apply a kernel
kernel =   gauss_kern(ksize, 1.414)*denominator
#print(kernel)

image_row, image_col = img.shape
kernel_row, kernel_col = kernel.shape

#create an output image
res = np.zeros(img.shape)

pad_height = int((kernel_row - 1) / 2)
pad_width = int((kernel_col - 1) / 2)
```

```
padded_image = np.zeros((image_row + (2 * pad_height), image_col + (2 * pad_width
)))

padded_image[pad_height:padded_image.shape[0] - pad_height, pad_width:padded_imag
e.shape[1] - pad_width] = img


for row in range(image_row):
    for col in range(image_col):
        res[row, col] = np.sum(kernel * padded_image[row:row + kernel_row, col:co
l + kernel_col])

plt.figure("After Gaussian Filtering")
plt.imshow(res, cmap = 'Greys_r')
plt.title("Output Image using {}X{} Kernel".format(kernel_row, kernel_col))
#print(res.shape)

plt.show()


plt.imsave('outputP6.png', res, cmap = 'Greys_r')
#image = Image.fromarray(res)
#image.save('outputP66.png')
```