Answer 1 A:

The matrix equations that indicate rotation, scaling and translation for a 2D vector (x,y) are as follows:

- Rotation :

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

- Translation:

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

- Scaling:

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Where, Θ = rotation angle;
(tx, ty) = translation vector
(Sx,Sy) = scale factor

Answer 2A.

We consider a simple pinhole camera model to transfer the world coordinates of any point (X,Y,Z) to the image coordinates (x,y) under perspective projection.

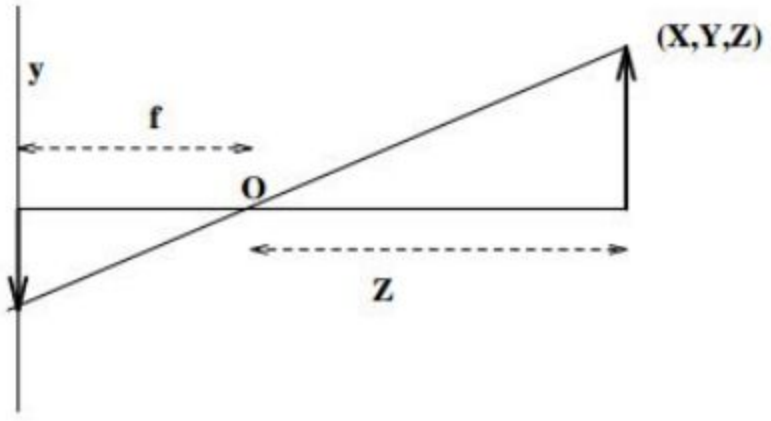

Fig1. Pinhole camera model with origin at lens

From the figure, we can get the x and y coordinates of the image as:

$$x = -f\frac{X}{Z} \qquad y = -f\frac{Y}{Z}$$

To derive the projection matrix, we need to transform the cartesian world coordinates (X,Y,Z) into the homogenous world coordinates (kX, kY, kZ, k).

Consider the camera parameters as below:

- Image coordinates (x_{image}, y_{image})
- Image center (o_x, o_y)
- Camera coordinates (x_{camera}, y_{camera})
- Real world coordinates (X, Y, Z)
- Focal length f
- Effective size of pixel in millimeter (k_x, k_y)

We can transform the camera coordinates to image coordinates using the following matrix equation.

$$\begin{bmatrix} x_{image} \\ y_{image} \\ 1 \end{bmatrix} = \begin{bmatrix} k_x & 0 & o_x \\ 0 & k_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{camera} \\ y_{camera} \\ 1 \end{bmatrix}$$

Also, we can write this as:

$$\begin{bmatrix} U_{image} \\ V_{image} \\ S \end{bmatrix} = \begin{bmatrix} k_x & 0 & o_x \\ 0 & k_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} U_{camera} \\ V_{camera} \\ S \end{bmatrix} \quad U_{camera} = -fX,\ V_{camera} = -fY \text{ and } S = Z$$

So, finally the world coordinates can be transformed to the image coordinates as:

$$\begin{bmatrix} U_{image} \\ V_{image} \\ S \end{bmatrix} = \begin{bmatrix} k_x & 0 & o_x \\ 0 & k_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

This matrix equation can be simplified to the following equation:

$$\begin{bmatrix} U_{image} \\ V_{image} \\ S \end{bmatrix} = \begin{bmatrix} -fk_x & 0 & o_x & 0 \\ 0 & -fk_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad \text{.....(i)}$$

Consider $f_x = fk_x$ *and* $f_y = fk_y$, then we can write the camera projection matrix C as

$$C = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

In the above matrix, f_x, f_y and o_x, o_y are considered intrinsic camera parameters that do not depend on camera position in the real world.
For extrinsic camera parameters, we consider rotation and translation of camera in real world.
we can write the transformation equation from world coordinates to camera coordinates first by aligning the world coordinates to the camera coordinates using rotation matrix and then shifting the origin.

$$\mathbf{X}_{camera} = R\mathbf{X}_{world} + T$$

$$\begin{bmatrix} X_{camera} \\ Y_{camera} \\ Z_{camera} \\ 1 \end{bmatrix} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{world} \\ Y_{world} \\ Z_{world} \\ 1 \end{bmatrix}$$

where t_x, t_y , t_z and $r_{1,1}$ … $r_{3,3}$ are extrinsic camera parameters. These are defined as the transformation matrices.

Rotations:

- Around x axis $$R^X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$
- Around y axis $$R^Y = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$$
- Around z axis $$R^Z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation by (t_x, t_y , t_z) $$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now we can express equation (i) as,

$$\begin{bmatrix} U_{image} \\ V_{image} \\ S \end{bmatrix} = \begin{bmatrix} -fk_x & 0 & o_x & 0 \\ 0 & -fk_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{world} \\ Y_{world} \\ Z_{world} \\ 1 \end{bmatrix}$$

Source: Class Notes

Answer 2B

Consider a Linear equation $\boldsymbol{Ax} = 0$, where $\boldsymbol{A}$ is m *n matrix and $\boldsymbol{x}$ is n*1 vector. The equation has addition constraint i.e. $||\boldsymbol{x}|| = 1$ to avoid trivial solution.
Our goal is to min($||\boldsymbol{Ax}||$) subject to $||\boldsymbol{x}|| = 1$.

We use Lagrange multiplier to minimize this function subjected to constraint. So we get,

$$\text{Min } (((\boldsymbol{Ax})^T \boldsymbol{Ax} + \lambda (1 - \boldsymbol{x}^T\boldsymbol{x})) \quad \text{.....enq(i)}$$

Taking the first derivative to this function and setting to zero , we get

$$2\boldsymbol{A}^T \boldsymbol{Ax} - 2\lambda \boldsymbol{x} = 0$$
$$\boldsymbol{A}^T \boldsymbol{Ax} = \lambda \boldsymbol{x} \quad \text{…….eqn(ii)}$$

From the definition of eigen vector, we can say $\boldsymbol{x}$ is an eigen vector of $\boldsymbol{A}^T \boldsymbol{A}$.

Substituting eqn(ii) to eqn (i), we get

$$\min(\lambda \boldsymbol{x}^T\boldsymbol{x} + \lambda (1- \boldsymbol{x}^T\boldsymbol{x})))$$
$$\min(\lambda)$$

Hence, $\boldsymbol{x}$ is an eigen vector of $\boldsymbol{A}^T \boldsymbol{A}$ corresponding to small eigen value.

Answer 1B

1.Rotation

Each pixel in an image has a coordinate pair (x,y) describing its position on two orthogonal axes from defined origin O. We take the RGB values at every (x,y) location, perform rotation as needed using rotation matrix, and then write these values in the new location ($x^{|}$,$y^{|}$).

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

But the destination location obtained from this rotation equation is real numbers that needs to be rounded back to integers again to be plotted. This rounding means sometimes certain pixels are missed completely whereas some are addressed more than once. When the pixels are missed, it results in hole.

This problem is solved by inverting the problem. Instead of finding the destination pixel, for each destination pixel, we find source pixel from original image and write the pixel value from that source pixel to destination pixel. This algorithm ensures that there are no any holes in destination pixel as shown in fig 1.

Results:

Fig1a. Original Image

Fig1b. Rotated image by 45 degree

Fig1c.rotated image by -45 degree

2. Scaling

We can scale image by multiplying each pixel coordinate with scale factor (sx, sy) to the original to get the new coordinate (X', Y'). And then copy the pixel intensity value from pixel (X, Y) to pixel (X', Y'). The obtained result due to scaling is shown in Fig2b.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Fig 2a. Original image

fig2b. Scaled image by scale factor = 2

3. Translation

We can translate each pixel coordinate by adding translation coordinate (tx, ty) to the original pixel coordinate (X, Y) to get the new coordinate (X', Y'). And then copy the pixel intensity value from pixel (X, Y) to pixel (X', Y'). The obtained result due to scaling is shown in Fig3b.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}$$

Fig3a. Original Image

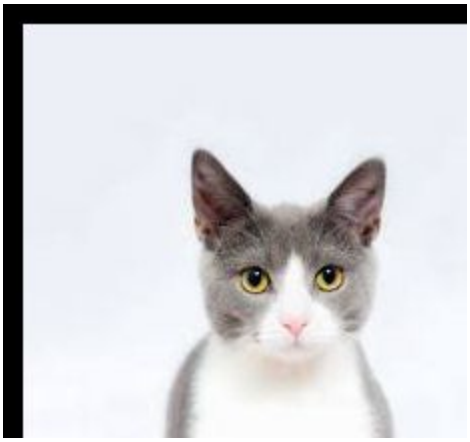

Fig3b. Translated image by (tx=10,ty=10)

3. We have Xh = b, where X is (20 * 5) matrix and b is (20 * 10) vector. The solution for h is (5 *1) vector such that the norm of the error e = Xh - b is minimum. First, we found the MMSE solution which is given by the below equation of matrix vector algebra.

$$\boldsymbol{h = (X^T X)^{-1} X^T b} \quad \text{……(i)}$$

The obtained result by this method is ***h*** = [[0.11816006][0.43006877] [-0.14024456] [0.26235196] [0.34986115]]

Second, we used the iterative solution to this least square problem. We start with an initial guess of vector h_0, and then we calculate error e. We iteratively calculate h by computing the gradient at this point by selecting appropriate step size parameter μ as shown in equation (ii). The iteration keeps on going until the error e is minimum. The decrease in the norm of the error as a function of the iteration is plotted as shown in fig4.

$$\boldsymbol{h_n = h_n - \mu * 2 * A^T * (A * h_n - b)} \quad \text{…… (ii)}$$

The obtained result by this method is ***h*** = [[0.11133435] [0.42259186] [-0.12438997] [0.25966145] [0.3487464]].

The solution from both the method are found to be almost similar.

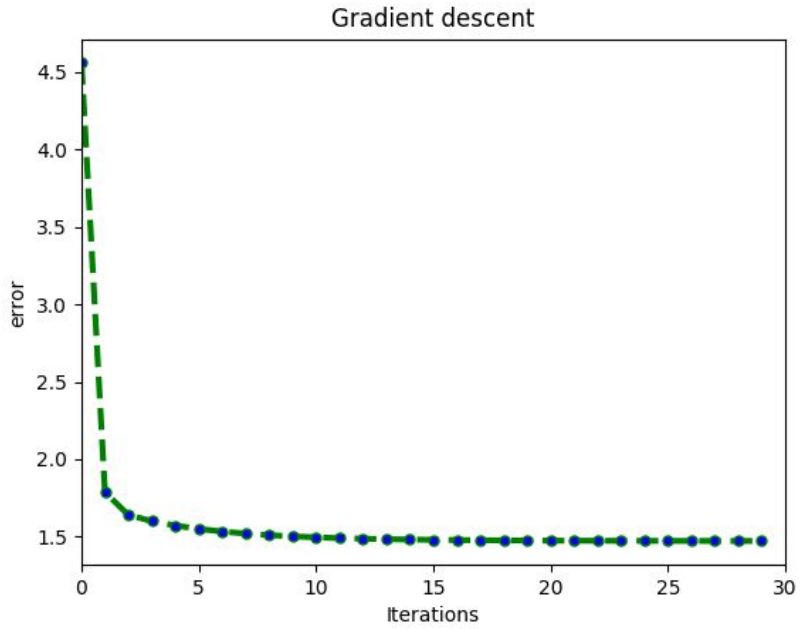

Fig4. Decrease in the norm of the error as a function of the iteration with μ =0.025

.

Instructions to run the program

1. Install the following required dependencies
   $ sudo apt-get install python3-pip
   $ pip3 install numpy
   $ pip3 install matplotlib
   $ pip3 install opencv-python

2. Unzip the file “4805322.zip”
   $ sudo apt-get install unzip
   $ unzip 4805322.zip

3. From inside the code directory, run the python files
   $ python3 LinearFit.py
   $ python3 Transformations.py