ECE 4554 / 5554: Computer Vision: Homework 3

Name : *Ankit Parekh*
VT Username : *ankitparekh*
Program : *MS CPE (Non-Thesis)*

Problem 1) Solution:

2D Planar Perspective Transformation or 2D-Homography:

$$\begin{bmatrix} s.x' \\ s.y' \\ s \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \text{ where } H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$$

To normalize H; $\sqrt{\sum (h_{ij})^2} = 1$ which leaves us with 8 degrees of freedom instead of 9

a) **Computing Homography for a transformation:**

Condition: $n \geq 4$, where n is the number of corresponding pair of points

Considering we have n pairs of points (mappings):

$$(x_1', y_1') \leftrightarrow (x_1, y_1)$$
$$(x_2', y_2') \leftrightarrow (x_2, y_2)$$
$$\vdots$$
$$(x_n', y_n') \leftrightarrow (x_n, y_n)$$

We can represent the transformation as shown below:

$$\underbrace{\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix}}_{\text{Transformed Coordinates}} \equiv \underbrace{\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}}_{\text{Destination Image Coordinates}} = \underbrace{\begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}}_{H} \underbrace{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}_{\text{Source Image Co-ordinates}}$$

For a given pair (i) of corresponding pts.:

$$x'_{(i)} = \frac{\tilde{x}_{(i)}}{\tilde{z}_{(i)}} = \frac{h_{00}\, x_{(i)} + h_{01}\, y_{(i)} + h_{02}}{h_{20}\, x_{(i)} + h_{21}\, y_{(i)} + h_{22}} \quad \ldots\ldots \text{(I)}$$

$$y'_{(i)} = \frac{\tilde{y}_{(i)}}{\tilde{z}_{(i)}} = \frac{h_{10}\, x_{(i)} + h_{11}\, (y_{(i)}) + h_{12}}{h_{20}\, x_{(i)} + h_{21}\, y_{(i)} + h_{22}} \quad \ldots\ldots \text{(II)}$$

Rearranging terms in I & II we get :

$$\boxed{x'_{(i)}(h_{20}x_{(i)} + h_{21}y_{(i)} + h_{22}) = h_{00}x_{(i)} + h_{01}y_{(i)} + h_{02}}$$

$$\boxed{y'_{(i)}(h_{20}x_{(i)} + h_{21}y_{(i)} + h_{22}) = h_{10}x_{(i)} + h_{11}y_{(i)} + h_{12}}$$

Rearranging terms and writing in matrix notation:

$$\begin{bmatrix} x_{(i)} & y_{(i)} & 1 & 0 & 0 & 0 & -x'_{(i)}x_{(i)} & -x'_{(i)}y_{(i)} & -x'_{(i)} \\ 0 & 0 & 0 & x_{(i)} & y_{(i)} & 1 & -y'_{(i)}x_{(i)} & -y'_{(i)}y_{(i)} & -y'_{(i)} \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Combining the equations for all corresponding pts.:

$$\underbrace{\underset{2n\times 9}{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2x_2 & -x'_2y_2 & -x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2x_2 & -y'_2y_2 & -y'_2 \\ \vdots & & & & & \vdots & & & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{bmatrix}}}_{Q} \underbrace{\underset{9\times 1}{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}}}_{a} = \underset{2n\times 1}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}$$

$$0$$

We can write the above as : $\boxed{Qa = 0}$

Solving for a such that $||a||^2 = 1$. We can use least squares method to solve the problem.

Defining Least Squares Problem:

$$\boxed{\min_a ||Qa||^2 \text{ such that } ||a||^2 = 1}$$

We know that: $||Qa||^2 = (Qa)^T(Qa) = a^T Q^T Q a$

and $||a||^2 = a^T a = 1$

$\therefore$ We need to solve: $\min_a (a^T Q^T Q a)$ such that $a^T a = 1$

Defining Loss function: $L(a, \lambda) = a^T Q^T Q a - \lambda (a^T a - 1)$

and we need to find a that minimizes L

Taking derivative of $L(a, \lambda)$ w.r.t. a:

$$\boxed{2Q^TQa - 2\lambda a = 0}$$

$$\boxed{Q^TQa = \lambda a}$$ Eigen Value Problem

Eigen vector a with smallest eigen value λ of matrix Q^TQ minimizes the loss function $L(\lambda)$.

1b) Given set of key-point pairs:

$$(x_i', y_i') \leftrightarrow (x_i, y_i)$$
$$(3.0, 2.0) \leftrightarrow (0, 0)$$
$$(3.67, 2.0) \leftrightarrow (1, 0)$$
$$(3.5, 2.5) \leftrightarrow (1, 1)$$
$$(3.0, 3.0) \leftrightarrow (0, 1)$$

Creating Q matrix using above:

$$Q = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -3 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -2 \\ 1 & 0 & 1 & 0 & 0 & 0 & -3.67 & 0 & -3.67 \\ 0 & 0 & 0 & 1 & 0 & 1 & -2.0 & 0 & -2.0 \\ 1 & 1 & 1 & 0 & 0 & 0 & -3.5 & -3.5 & -3.5 \\ 0 & 0 & 0 & 1 & 1 & 1 & -2.5 & -2.5 & -2.5 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & -3.0 & -3.0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & -3.0 & -3.0 \end{bmatrix}$$

Solving the above Eigen Value problem using Python code.

<u>*Python Code to Solve the Above Eigen Value Problem:*</u>

```python
def step_break(): # function for output clarity, draws lines between steps
  for i in range(70):
    print("=", end="")
  print()

print("Python code for solving Eigen Value problem [transpose(Q).Q.a =  λ.a]")
import numpy as np

step_break()
print("Step 1 - Initializing matrix Q:")
Q = np.array([[0, 0, 1, 0, 0, 0, 0, 0, -3], [0, 0, 0, 0, 0, 1, 0, 0, -2], [1, 0, 1, 0, 0, 0, -3.67, 0, -3.67], [0, 0, 0, 1, 0, 1, -2, 0, -2], [1, 1, 1, 0, 0, 0, -3.5, -3.5, -3.5], [0, 0, 0, 1, 1, 1, -2.5, -2.5, -2.5], [0, 1, 1, 0, 0, 0, 0, -3, -3], [0, 0, 0, 0, 1, 1, 0, -3, -3]])
print(Q)

step_break()
print("Step 2 - Computing matrix = transpose(Q).Q:")
matrix = np.matmul(np.transpose(Q), Q)
print(matrix)

step_break()
print("Step 3 - Computing eigen values of matrix:") #np.linalg.eigh returns eigen values as well as vectors, [0]: values, [1]: vectors (columns in the matrix)
eigen_values = np.linalg.eigh(matrix)[0]
print(eigen_values)

step_break()
print("Step 4 - Computing eigen vectors of matrix:") #np.linalg.eigh returns eigen values as well as vectors, [0]: values, [1]: vectors (columns in the matrix)
eigen_vectors = np.transpose(np.linalg.eigh(matrix)[1])
print(eigen_vectors)

step_break()
print("Step 5 - Smallest eigen value of matrix:")
min_eigen_value = min(eigen_values)
print(min_eigen_value)

step_break()
print("Step 6 - Index of smallest eigen value of the matrix:")
idx_min_eigen_value = eigen_values.tolist().index(min_eigen_value)
print(idx_min_eigen_value)

step_break()
print("Step 7 - Eigen vector corresponding to smallest eigen value:")
eigen_vector = np.linalg.eigh(matrix)[1][:, idx_min_eigen_value]
print(eigen_vector)

step_break()
print("Step 8 - Magnitude of eigen vector in Step 6:")
magnitude = sum([x**2 for x in eigen_vector])
print(magnitude)
```

```
print("Approximately equal to 1")

step_break()
print("Step 9 - Solution to our eigen value problem:")
print("2D Homography matrix H:")
h_string = "h00,h01,h02,h10,h11,h12,h20,h21,h22"
SUB = str.maketrans("012", "₀₁₂")
h_parameters = h_string.translate(SUB).split(",")
h = np.array([h_parameters[0:3],h_parameters[3:6],h_parameters[6:9]])
print(h)

step_break()
print("Step 10 - Arranging values from eigen vector in H matrix:")
print("H =")
H = np.array([eigen_vector[0:3],eigen_vector[3:6], eigen_vector[6:9]])
print(H)

step_break()
print("Step 11 - Normalizing H - Dividing all parameters of H by {}:".format(h_parameters[-1]))
normalized_H = H/H[2][2]
print(normalized_H)
```

Output:

```
Python code for solving Eigen Value problem [transpose(Q).Q.a =  λ.a]
======================================================================
Step 1 - Initializing matrix Q:
[[ 0.    0.    1.    0.    0.    0.    0.    0.   -3.  ]
 [ 0.    0.    0.    0.    0.    1.    0.    0.   -2.  ]
 [ 1.    0.    1.    0.    0.    0.   -3.67  0.   -3.67]
 [ 0.    0.    0.    1.    0.    1.   -2.    0.   -2.  ]
 [ 1.    1.    1.    0.    0.    0.   -3.5  -3.5  -3.5 ]
 [ 0.    0.    0.    1.    1.    1.   -2.5  -2.5  -2.5 ]
 [ 0.    1.    1.    0.    0.    0.    0.   -3.   -3.  ]
 [ 0.    0.    0.    0.    1.    1.    0.   -3.   -3.  ]]
======================================================================
Step 2 - Computing matrix = transpose(Q).Q:
[[  2.        1.        2.        0.        0.        0.       -7.17     -3.5
   -7.17   ]
 [  1.        2.        2.        0.        0.        0.       -3.5      -6.5
   -6.5    ]
 [  2.        2.        4.        0.        0.        0.       -7.17     -6.5
  -13.17   ]
 [  0.        0.        0.        2.        1.        2.       -4.5      -2.5
   -4.5    ]
 [  0.        0.        0.        1.        2.        2.       -2.5      -5.5
   -5.5    ]
 [  0.        0.        0.        2.        2.        4.       -4.5      -5.5
   -9.5    ]
 [ -7.17     -3.5      -7.17     -4.5      -2.5      -4.5      35.9689   18.5
   35.9689]
 [ -3.5      -6.5      -6.5      -2.5      -5.5      -5.5      18.5      36.5
   36.5    ]
 [ -7.17     -6.5     -13.17     -4.5      -5.5      -9.5      35.9689   36.5
   66.9689]]
======================================================================
Step 3 - Computing eigen values of matrix:
[1.44492540e-14 3.23775075e-03 1.17328773e-02 6.28086068e-01
 1.05035523e+00 6.29874446e+00 9.88257262e+00 1.87961201e+01
 1.18766951e+02]
======================================================================
Step 4 - Computing eigen vectors of matrix:
```

```
[[ 7.18057522e-01  2.73290549e-01  2.65252592e-01  3.59028761e-01
   3.61708080e-01  1.76835061e-01  1.79514380e-01  9.10968498e-02
   8.84175307e-02]
 [-4.68653807e-03  5.00438751e-01 -6.54552704e-01  6.56928604e-02
   2.58398623e-01 -4.28997984e-01  2.20950428e-02  1.48279603e-01
  -2.08566893e-01]
 [ 4.07730520e-01 -4.84488207e-01 -3.24353129e-01  3.88046463e-01
  -4.58755182e-01 -2.79928161e-01  1.40707551e-01 -1.47206781e-01
  -1.13830779e-01]
 [-3.32909483e-01 -2.89584122e-01  3.54788984e-01  4.34083050e-01
   4.52219902e-01 -5.35246885e-01 -3.12081578e-03  9.97485289e-04
  -2.48873938e-03]
 [ 3.35554163e-01 -4.88329187e-01 -1.56639859e-01 -6.01849029e-01
   4.96304364e-01 -1.16856122e-01 -2.01472277e-02 -1.92615327e-02
  -3.78076235e-02]
 [-2.36105681e-01 -2.98819894e-01 -4.45424089e-01  3.89196755e-01
   3.42505169e-01  6.18675813e-01 -4.51864574e-02  5.42285401e-02
  -5.65175407e-03]
 [ 9.04568314e-02  1.01654722e-01 -1.60267366e-01  7.64971115e-02
   6.45653836e-02 -1.01836981e-01 -5.12604777e-01 -5.18284718e-01
   6.35629040e-01]
 [ 1.50909611e-01 -1.06411560e-01  4.05220697e-02  6.99782104e-02
  -1.22604415e-01 -5.09702230e-02 -6.90550855e-01  6.81816058e-01
   6.73648288e-04]
 [-8.94421224e-02 -8.28837628e-02 -1.40908882e-01 -5.77019885e-02
  -6.78963620e-02 -1.02260644e-01  4.53217579e-01  4.59608273e-01
   7.28170627e-01]]
======================================================================
Step 5 - Smallest eigen value of matrix:
1.444925396309616e-14
======================================================================
Step 6 - Index of smallest eigen value of the matrix:
0
======================================================================
Step 7 - Eigen vector corresponding to smallest eigen value:
[0.71805752 0.27329055 0.26525259 0.35902876 0.36170808 0.17683506
 0.17951438 0.09109685 0.08841753]
======================================================================
Step 8 - Magnitude of eigen vector in Step 6:
0.9999999999999999
Approximately equal to 1
======================================================================
Step 9 - Solution to our eigen value problem:
2D Homography matrix H:
[['h_00' 'h_01' 'h_02']
 ['h_10' 'h_11' 'h_12']
 ['h_20' 'h_21' 'h_22']]
======================================================================
Step 10 - Arranging values from eigen vector in H matrix:
H =
[[0.71805752 0.27329055 0.26525259]
 [0.35902876 0.36170808 0.17683506]
 [0.17951438 0.09109685 0.08841753]]
======================================================================
Step 11 - Normalizing H - Dividing all parameters of H by h_22:
[[8.12121212 3.09090909 3.        ]
 [4.06060606 4.09090909 2.        ]
 [2.03030303 1.03030303 1.        ]]
```

Problem 2) Solution:

2D Affine Transformation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = A \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \text{ where } A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix}$$

2a) Let A be an invertible matrix, let $\vec{b}$ be a vector, and let $f: \mathbb{R}^2 \Leftrightarrow \mathbb{R}^2$ be defined as an Affine Transformation via $f(\vec{x}) \mapsto A\vec{x} + \vec{b}$.

$$f(\vec{x}) = \vec{y}$$
$$A\vec{x} + \vec{b} = \vec{y}$$
$$A\vec{x} = \vec{y} - \vec{b}$$
$$A^{-1}(A\vec{x}) = A^{-1}(\vec{y} - \vec{b}) \quad \text{Multiplying } A^{-1} \text{ on both sides}$$
$$I_2\vec{x} = A^{-1}\vec{y} - A^{-1}\vec{b}$$
$$\vec{x} = A^{-1}(\vec{y} - \vec{b})$$

We can conclude that f^{-1} exists and can be given by $f^{-1}(\vec{x}) = A^{-1}(\vec{x} - \vec{b})$

This is also an Affine transformation.

For every vector $\vec{x}$: $\boxed{(f^{-1} \circ f)(\vec{x}) = (f \circ f^{-1})(\vec{x}) = \vec{x}}$

$\therefore$ Both f & f^{-1} are bijections on $\mathbb{R}^2$.

2b) Computing transformed coordinates:

$(x, y) \Leftrightarrow (0, 0)$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{02} \\ a_{12} \\ 1 \end{bmatrix}$$

$(x', y') \Rightarrow (a_{02}, a_{12})$

$(x, y) \Leftrightarrow (1, 0)$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} + a_{02} \\ a_{10} + a_{12} \\ 1 \end{bmatrix}$$

$(x', y') \Rightarrow (a_{00} + a_{02}, a_{10} + a_{12})$

$(x, y) \Leftrightarrow (0, 1)$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{01} + a_{02} \\ a_{11} + a_{12} \\ 1 \end{bmatrix}$$

$(x', y') \Rightarrow (a_{01} + a_{02}, a_{11} + a_{12})$

2c) A & B are both 3X3 affine transformation matrices

Let $f(\vec{x}) = A\vec{x} + \vec{a}$ represent transformation using matrix A

$g(\vec{x}) = B\vec{x} + \vec{b}$ represent transformation using matrix B

Then, $(g \circ f)(\vec{x}) = g(f(\vec{x}))$

$= B(A\vec{x} + \vec{a}) + \vec{b}$

$= (BA)\vec{x} + (B\vec{a} + \vec{b})$

Since A & B are invertible, (BA) is invertible as well.
∴ (g∘f) or BA is also an affine transformation.

We can show this through matrix multiplication as well:

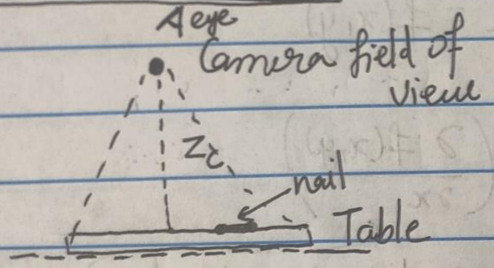
$$BA = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix}$$

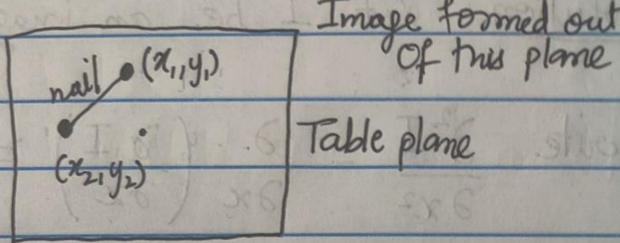
$$BA = \begin{bmatrix} b_{00}a_{00} + b_{01}a_{10} & b_{00}a_{01} + b_{01}a_{11} & b_{00}a_{02} + b_{01}a_{12} + b_{02} \\ b_{10}a_{00} + b_{11}a_{10} & b_{10}a_{01} + b_{11}a_{11} & b_{10}a_{02} + b_{11}a_{12} + b_{12} \\ 0 & 0 & 1 \end{bmatrix}$$

$$BA = \begin{bmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ 0 & 0 & 1 \end{bmatrix}$$ where c represents a constant.

∴ BA is also an affine transformation

Problem 3) Solution:





Length of the nail (distance formula) $= l = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Considering ideal perspective projection, we can compute image positions of (x_1, y_1) as (x_1', y_1') & (x_2, y_2) as (x_2', y_2').

$$\begin{bmatrix} x_1' \\ y_1' \end{bmatrix} = \frac{f}{Z_c} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \quad \& \quad \begin{bmatrix} x_2' \\ y_2' \end{bmatrix} = \frac{f}{Z_c} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

Length of nail in the image $= l' = \sqrt{\left(\frac{f}{Z_c}\right)^2 (x_1 - x_2)^2 + \left(\frac{f}{Z_c}\right)^2 (y_1 - y_2)^2}$

$$l' = \sqrt{\left(\frac{f}{Z_c}\right)^2 \left[(x_1 - x_2)^2 + (y_1 - y_2)^2\right]}$$

$$\boxed{l' = \frac{f}{Z_c} \times l}$$

$\therefore$ l' is only dependent on f, Z_c & l, and not dependent on nail's position or orientation.

$\therefore$ l' will remain constant irrespective of changes in position of nail or orientation in case of ideal perspective projection.

Problem 4) Solution: Let I be an image. $I = f(x,y)$

We can write $\dfrac{\partial^2 I}{\partial x^2} = \dfrac{\partial}{\partial x}\left(\dfrac{\partial I}{\partial x}\right) = \dfrac{\partial}{\partial x}\left(\dfrac{\partial f(x,y)}{\partial x}\right)$

Using Backward Diff. Formula of derivative in x-direction

$$\frac{\partial^2 I}{\partial x^2} = \frac{\partial}{\partial x}\left(\lim_{h \to 0} \frac{f(x,y) - f(x-h,y)}{h}\right)$$

For images that have discrete pixel values, we have to consider the smallest difference as $h=1$

$$\frac{\partial^2 I}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{f(x,y) - f(x-1,y)}{1}\right)$$

$$\frac{\partial^2 I}{\partial x^2} = \frac{f'(x,y) - f'(x-1,y)}{1}$$

$\left[f'(x,y) = \dfrac{\partial}{\partial x}(f(x,y)) \right]$

$\left[f'(x-1,y) = \dfrac{\partial}{\partial x}(f(x-1,y)) \right]$

Using forward difference formula above:

$$\frac{\partial^2 I}{\partial x^2} = \left[\frac{f(x+1,y) - f(x,y)}{1}\right] - \left[\frac{f(x,y) - f(x-1,y)}{1}\right]$$

$$\frac{\partial^2 I}{\partial x^2} = f(x+1,y) - 2f(x,y) + f(x-1,y)$$

We can take the co-efficients of eqn. above to create a convolutional filter in x-direction:

$$\Rightarrow [\underset{f(x-1,y)}{1}, \underset{f(x,y)}{-2}, \underset{f(x+1,y)}{1}]$$

$\therefore \dfrac{\partial^2 I}{\partial x^2}$ is approximated using $\boxed{1 \mid -2 \mid 1}$ kernel