To Calcualte ZERNIKE MOMENTs

For a discrete image, if P_{xy} is the current pixel then Eq^n # 1 becomes,

$$\boxed{A_{mn} = \frac{m+1}{\pi} \sum_x \sum_y P_{xy} \left[V_{mn}(x,y) \right]^*}$$

where $x^2+y^2 \leq 1$
$r \leq 1$

<u>To calculate the Zernike moments</u>: —

moment, the image (or ROI) is first mapped to a unit disc using Polar coord where the centre of image is at the origin of the unit disc. Those pixels falling outside the unit disc are not used in calculation. The coordinates are then described by the length of the vector form from the origin to the coordinate point, r, 'θ' is the angle measured from the +ve X axis anticlockwise positive.

Use $x = r\cos\theta$ $\quad$ $\therefore$ $\theta = \tan^{-1}(y/x)$ $\Rightarrow$ Note: $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$.

$y = r \cdot \sin\theta$ $\quad$ $r = \sqrt{x^2+y^2}$.

Translation and Scale invariance can be achieved by normalising the image using Cartesian coord prior to Calculation of the Zernike moments.

Translation and Scale invariance can be achieved by normalising the image using Cartesian coord prior to calculation of the Zernike moments.

TRANSLATION invariance is achieved by moving the origin to the images' COM, causing $m_{01} = m_{10} = 0$. Following this, SCALE invariance is produced by altering each object so that its area (or pixel count for a binary image) is $m_{00} = \beta$, where β is a pre-determined value.

Both invariance properties (for a binary image) can be achieved using:

$$\boxed{h(x, y) = f\left(\frac{x}{a} + \bar{x}, \frac{y}{a} + \bar{y}\right)} \qquad \text{where } a = \sqrt{\frac{\beta}{m_{00}}}$$

and $h(x, y)$ is the new translated and scaled function.

The Error involved in discrete transformation can be reduced by interpolation. If the coordinate calculated by $h(x,y)$ does not coincide with the actual grid location, then the pixel value associated with it is interpolated from the four surrounding pixels. As a result of normalisation, the Zernike moments $|A_{00}|$ & $|A_{11}|$ are set to known values. $|A_{11}|$ is set to zero, due to translatn of the shape to the centre of the co-ord system. $|A_{00}|$ is dependent on m_{00}, and thus on β. $\therefore |A_{00}| = \frac{\beta}{\pi}$.

Further, the absolute value of Zernike moment is rotn invariant as reflected in mapping of the image to the unit DISC.

Pg 4

The rotation of the shape around the unit disc is expressed as phase change, if "ϕ" is the angle of rotation then A^R_{mn} is the Zernike moment of the rotated image and A_{mn} is the Zernike moment of the original image, such that

$$\boxed{A^R_{mn} = A_{mn} \cdot \exp(-jn\phi)}$$

```
# -------------------------------------------------------------------
# Function to find the Zernike moments for an N x N binary ROI
#
# Z, A, Phi = Zernikmoment(src, n, m)
# where
#   src = input image
#   n = The order of Zernike moment (scalar)
#   m = The repetition number of Zernike moment (scalar)
# and
#   Z = Complex Zernike moment
#   A = Amplitude of the moment
#   Phi = phase (angle) of the mement (in degrees)
#
# Example:
#   1- calculate the Zernike moment (n,m) for an oval shape,
#   2- rotate the oval shape around its centeroid,
#   3- calculate the Zernike moment (n,m) again,
#   4- the amplitude of the moment (A) should be the same for both images
#   5- the phase (Phi) should be equal to the angle of rotation
```

STEP 1 : Check if the IMAGE source is in float32 format if NOT then convert it into flaot32

STEP 2: ZERNIKE moments needs the imnage to be in GRAY format. ENSURE this

STEP =3 : We need the Image to be SQUARE in shape i.e. no of ROWS = no. of Columns

STEP = 4 : It is better if the Image is an odd by odd GRID. i.e. N= ODD, ENSURE this, else

STEP = 5 : Assign N to x and y both and then CREATE a MESHGRID of N X N

STEP = 6 :

```
def Zernikemoment(src, n, m):
    if src.dtype != np.float32:                         # STEP 1
        src = np.where(src > 0, 0, 1).astype(np.float32)
    if len(src.shape) == 3:                             #STEP 2
        print 'the input image src should be in gray'
        return

    H, W = src.shape                                    # STEP 3
    if H > W:
        src = src[(H - W) / 2: (H + W) / 2, :]
    elif H < W:
        src = src[:, (W - H) / 2: (H + W) / 2]

    N = src.shape[0]                                    #STEP 4
    if N % 2:
        src = src[:-1, :-1]
        N -= 1

    x = range(N)                                        # STEP 5
    y = x
    X, Y = np.meshgrid(x, y)


    R = np.sqrt((2 * X - N + 1) ** 2 + (2 * Y - N + 1) ** 2) / N   # STEP 6
    Theta = np.arctan2(N - 1 - 2 * Y, 2 * X - N + 1)
    R = np.where(R <= 1, 1, 0) * R

    Rad = radialpoly(R, n, m)     # get the radial polynomial

    Product = src * Rad * np.exp(-1j * m * Theta)

    Z = Product.sum()            # calculate the moments

    cnt = np.count_nonzero(R) + 1     # count the number of pixels inside the unit circle


    Z = (n + 1) * Z / cnt             # normalize the amplitude of moments

    A = abs(Z)                   # calculate the amplitude of the moment

    Phi = np.angle(Z) * 180 / np.pi   # calculate the phase of the mement (in degrees)

    return Z, A, Phi
```