```
import generator as gen
import preprocessing as pp
import cv2
import numpy as np
import matplotlib.pyplot as plt, matplotlib.cm as cm, matplotlib.patches as mp
atches
import scipy as scipy

#find the center of mass for the two images
def find_com(im1, im2):
    c1, c2 = [0, 0], [0, 0]
    s1, s2 = 0,0
    res = len(im1)
    for x in range(res):
        for y in range(res):
            c1[0] += x*im1[y][x]
            c1[1] += y*im1[y][x]
            c2[0] += x*im2[y][x]
            c2[1] += y*im2[y][x]
            s1 += im1[y][x]
            s2 += im2[y][x]
    return [c/s1 for c in c1], [c/s2 for c in c2]

#Shift the images according to the provided values
def shift_images(im1, im2, cd):
    num_rows, num_cols = im1.shape[:2]

    translation_matrix = np.float32([[1, 0, -cd[0]/2], [0, 1, -cd[1]/2]])
    im1 = cv2.warpAffine(im1, translation_matrix, (num_cols, num_rows))
    translation_matrix = np.float32([[1, 0, cd[0]/2], [0, 1, cd[1]/2]])
    im2 = cv2.warpAffine(im2, translation_matrix, (num_cols, num_rows))
    return im1, im2

#returns the tip and tilt, and the images with tip tilt removed
def parse_tip_tilt(preim, postim, defocus):
    focallength = 0.6096
    c1, c2 = find_com(preim, postim)
    cd = np.array([e1 - e2 for e1, e2 in zip(c1, c2)])
    preim, posim = shift_images(preim, postim, cd)
    #calculate rate (in pix/m)
    dzdr = cd * defocus/(2*focallength*(focallength - defocus))
    #Convert to m/m
    dzdr *= 5.86e-6
    #Convert to waves/5cm
    dzdr *= 20/0.55e-6
    #Convert from ptp to rms
    dzdr /= 3

    return dzdr, preim, posim

#Will be used for later functionality
def get_signal(preim, postim, mask):
    S = preim - postim
```

```
    S /= preim + postim
    S[np.logical_not(mask)] = 0
    return S

if __name__ == "__main__":
    #Go through a range of defocuses and identify tip and tilt

    zern = [-0.2868445,
            0,
            0.08498557,
            -0.16551221,
            0,
            0.16330520,
            0.03002613,
            0,
            0.00034476,
            0,
            -0.0008443]
    plt.ion()
    plt.show()

    dfl = [0.1e-3, 0.2e-3, 0.4e-3, 0.8e-3, 1.6e-3, 3.2e-3, 6.4e-3]
    dfl = np.arange(0.4e-3, 1.0e-3, 0.05e-3)
    error = []

    for defocus in dfl:
        preim, postim = gen.generate_images(zern, defocus = defocus)
        preim, postim = pp.normalize(preim, postim)
        preim, postim = pp.blur_images(preim, postim)
        dzdr, preim, postim = parse_tip_tilt(preim, postim, defocus)
        error.append(abs(dzdr - zern[1:3])*100)

        diff = preim - postim
        plt.subplot(1,3,1)
        plt.title("Prefocal")
        plt.imshow(preim, cmap=cm.gray)
        plt.subplot(1,3,2)
        plt.title("Postfocal")
        plt.imshow(postim, cmap=cm.gray)
        plt.subplot(1,3,3)
        plt.title("Difference")
        plt.imshow(diff, cmap=cm.gray)
        plt.draw()
        plt.pause(0.001)
        #plt.savefig("plots/" + 'OurZernikies' + '_{}um'.format(int(defocus*1e
6)) + ".png")



    fig, ax1 = plt.subplots()
    ax1.set_xlabel('Defocus Distance (m)')
    ax1.plot(dfl, [e[0] for e in error], 'r')
    ax1.plot(dfl, [e[1] for e in error], 'b')
```

```
    ax1.plot(dfl, [np.sqrt(e[0]**2 + e[1]**2) for e in error], 'g')
    #ax1.set_ylim((0,100))
    ax1.set_ylabel('Error (%)', color='r')
    ax1.tick_params('y', colors='r')

    red_patch = mpatches.Patch(color='red', label='Z2')
    blue_patch = mpatches.Patch(color='blue', label='Z3')
    green_patch = mpatches.Patch(color='green', label='Combined')
    plt.legend(handles=[red_patch, blue_patch, green_patch])

    fig.tight_layout()
    plt.show()
    plt.pause(10)
    plt.savefig("plots/" + 'OurZernikiesError.png')

    #Commented out section will be used later for higher order aberrations

    """


    defocus = 720e-6

    input = np.arange(0.04, 0.40, 0.01)
    output = []
    error = []
    zern = [-0.2868445,
            0,
            0.08498557,
            -0.16551221,
            0,
            0.16330520,
            0.03002613,
            0,
            0.00034476,
            0,
            -0.0008443]

    preim, postim = gen.generate_images(zern, defocus = defocus)
    preim, postim = pp.baseline_process(preim, postim)
    dzdr, preim, postim = parse_tip_tilt(preim, postim, defocus)

    coef = dzdr[1]
    print(coef)

    """
    """

    for i in input:
        preim, postim = gen.generate_images([0,i], defocus = defocus)
        preim, postim = pp.baseline_process(preim, postim)
        cd, preim, postim = parse_tip_tilt(preim, postim)

        coef = dzdr[0]
```

```
        output.append(coef)
        error.append(abs((i-coef)/i)*100)

    #plt.plot(input.tolist(), output)
    #plt.plot(input.tolist(), error)
    #plt.show()

    fig, ax1 = plt.subplots()
    ax1.plot(input, output, 'b')
    ax1.set_xlabel('Z2 (lambda RMS)')
    # Make the y-axis label, ticks and tick labels match the line color.
    ax1.set_ylabel('Z2 (lambda RMS)', color='b')
    ax1.tick_params('y', colors='b')

    ax2 = ax1.twinx()
    ax2.plot(input, error, 'r.')
    ax2.set_ylim((0,100))
    ax2.set_ylabel('Error (%)', color='r')
    ax2.tick_params('y', colors='r')

    fig.tight_layout()
    plt.show()


    #shrink mask by 10 pixels
    #mask = scipy.ndimage.binary_erosion(mask, iterations = 5).astype(mask.dty
pe)
    """
```