In [1]:

```
# File:       PlaneWaveData.py
# Author:     Dongwoon Hyun (dongwoon.hyun@stanford.edu)
# Created on: 2020-04-03
import numpy as np
import h5py
from scipy.signal import hilbert


class PlaneWaveData:
    """ A template class that contains the plane wave data.

    PlaneWaveData is a container or dataclass that holds all of the informati
    describing a plane wave acquisition. Users should create a subclass that
    __init__() according to how their data is stored.

    The required information is:
    idata       In-phase (real) data with shape (nangles, nchans, nsamps)
    qdata       Quadrature (imag) data with shape (nangles, nchans, nsamps)
    angles      List of angles [radians]
    ele_pos     Element positions with shape (N,3) [m]
    fc          Center frequency [Hz]
    fs          Sampling frequency [Hz]
    fdemod      Demodulation frequency, if data is demodulated [Hz]
    c           Speed of sound [m/s]
    time_zero   List of time zeroes for each acquisition [s]

    Correct implementation can be checked by using the validate() method. See
    PICMUSData class for a fully implemented example.
    """

    def __init__(self):
        """ Users must re-implement this function to load their own data. """
        # Do not actually use PlaneWaveData.__init__() as is.
        raise NotImplementedError

        # We provide the following as a visual example for a __init__() metho
        nangles, nchans, nsamps = 2, 3, 4
        # Initialize the parameters that *must* be populated by child classes
        self.idata = np.zeros((nangles, nchans, nsamps), dtype="float32")
        self.qdata = np.zeros((nangles, nchans, nsamps), dtype="float32")
        self.angles = np.zeros((nangles,), dtype="float32")
        self.ele_pos = np.zeros((nchans, 3), dtype="float32")
        self.fc = 5e6
        self.fs = 20e6
        self.fdemod = 0
        self.c = 1540
        self.time_zero = np.zeros((nangles,), dtype="float32")

    def validate(self):
        """ Check to make sure that all information is loaded and valid. """
        # Check size of idata, qdata, angles, ele_pos
        assert self.idata.shape == self.qdata.shape
        assert self.idata.ndim == self.qdata.ndim == 3
        nangles, nchans, nsamps = self.idata.shape
        assert self.angles.ndim == 1 and self.angles.size == nangles
        assert self.ele_pos.ndim == 2 and self.ele_pos.shape == (nchans, 3)
```

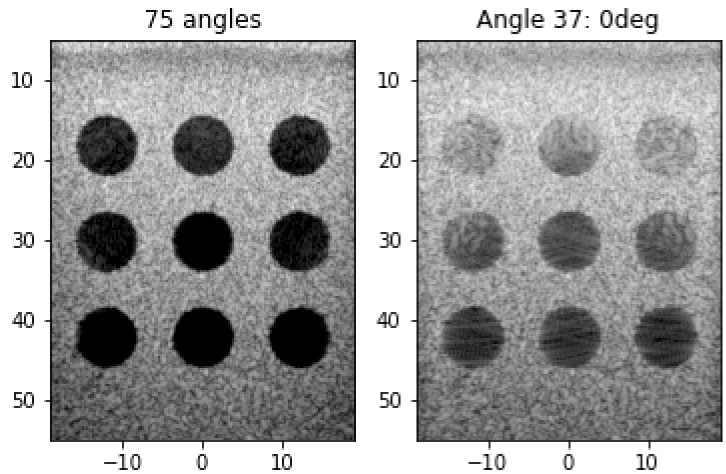
```
        # Check frequencies (expecting more than 0.1 MHz)
        assert self.fc > 1e5
        assert self.fs > 1e5
        assert self.fdemod > 1e5 or self.fdemod == 0
        # Check speed of sound (should be between 1000-2000 for medical imagi
        assert 1000 <= self.c <= 2000
        # Check that a separate time zero is provided for each transmit
        assert self.time_zero.ndim == 1 and self.time_zero.size == nangles


class PICMUSData(PlaneWaveData):
    """ PICMUSData - Demonstration of how to use PlaneWaveData to load PICMUS

    PICMUSData is a subclass of PlaneWaveData that loads the data from the PI
    challenge from 2016 (https://www.creatis.insa-lyon.fr/Challenge/IEEE_IUS_
    PICMUSData re-implements the __init__() function of PlaneWaveData.
    """

    def __init__(self, database_path, acq, target, dtype):
        """ Load PICMUS dataset as a PlaneWaveData object. """
        # Make sure the selected dataset is valid
        assert any([acq == a for a in ["simulation", "experiments"]])
        assert any([target == t for t in ["contrast_speckle", "resolution_dis
        assert any([dtype == d for d in ["rf", "iq"]])

        # Load PICMUS dataset
        fname = "%s/%s/%s/%s_%s_dataset_%s.hdf5" % (
            database_path,
            acq,
            target,
            target,
            acq[:4],
            dtype,
        )
        f = h5py.File(fname, "r")["US"]["US_DATASET0000"]
        self.idata = np.array(f["data"]["real"], dtype="float32")
        self.qdata = np.array(f["data"]["imag"], dtype="float32")
        self.angles = np.array(f["angles"])
        self.fc = 5208000.0  # np.array(f["modulation_frequency"]).item()
        self.fs = np.array(f["sampling_frequency"]).item()
        self.c = np.array(f["sound_speed"]).item()
        self.time_zero = np.array(f["initial_time"])
        self.ele_pos = np.array(f["probe_geometry"]).T
        self.fdemod = self.fc if dtype == "iq" else 0

        # If the data is RF, use the Hilbert transform to get the imag. compo
        if dtype == "rf":
            iqdata = hilbert(self.idata, axis=-1)
            self.qdata = np.imag(iqdata)

        # Make sure that time_zero is an array of size [nangles]
        if self.time_zero.size == 1:
            self.time_zero = np.ones_like(self.angles) * self.time_zero

        # Validate that all information is properly included
        super().validate()
```

In [4]:

```
# File:       example_picmus_tf.py
# Author:     Dongwoon Hyun (dongwoon.hyun@stanford.edu)
# Created on: 2020-04-27
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from das_tf import DAS_PW
from PlaneWaveData import PICMUSData
from PixelGrid import make_pixel_grid

# Load PICMUS dataset
database_path = "C:/Users/Block-03-EE/Desktop/cubdl-master"
acq = "simulation"
target = "contrast_speckle"
dtype = "iq"
P = PICMUSData(database_path, acq, target, dtype)

# Define pixel grid limits (assume y == 0)
xlims = [P.ele_pos[0, 0], P.ele_pos[-1, 0]]
zlims = [5e-3, 55e-3]
wvln = P.c / P.fc
dx = wvln / 3
dz = dx  # Use square pixels
grid = make_pixel_grid(xlims, zlims, dx, dz)
fnum = 1

# Create a DAS_PW neural network for all angles, for 1 angle
dasN = DAS_PW(P, grid)
idx = len(P.angles) // 2  # Choose center angle for 1-angle DAS
das1 = DAS_PW(P, grid, idx)

# Stack the I and Q data in the innermost dimension
with tf.device("/gpu:0"):
    iqdata = np.stack((P.idata, P.qdata), axis=-1)

# Make 75-angle image
idasN, qdasN = dasN(iqdata)
idasN, qdasN = np.array(idasN), np.array(qdasN)
iqN = idasN + 1j * qdasN  # Tranpose for display purposes
bimgN = 20 * np.log10(np.abs(iqN))  # Log-compress
bimgN -= np.amax(bimgN)  # Normalize by max value

# Make 1-angle image
idas1, qdas1 = das1(iqdata)
idas1, qdas1 = np.array(idas1), np.array(qdas1)
iq1 = idas1 + 1j * qdas1  # Transpose for display purposes
bimg1 = 20 * np.log10(np.abs(iq1))  # Log-compress
bimg1 -= np.amax(bimg1)  # Normalize by max value

# Display images via matplotlib
extent = [xlims[0] * 1e3, xlims[1] * 1e3, zlims[1] * 1e3, zlims[0] * 1e3]
plt.subplot(121)
plt.imshow(bimgN, vmin=-60, cmap="gray", extent=extent, origin="upper")
plt.title("%d angles" % len(P.angles))
plt.subplot(122)
plt.imshow(bimg1, vmin=-60, cmap="gray", extent=extent, origin="upper")
```

```
plt.title("Angle %d: %ddeg" % (idx, P.angles[idx] * 180 / np.pi))
plt.show()
```

```
100%|██████████| 75/75 [02:25<00:00,  1.94s/it]
100%|██████████| 1/1 [00:01<00:00,  1.25s/it]
```




In [10]:

```
# File:       PixelGrid.py
# Author:     Dongwoon Hyun (dongwoon.hyun@stanford.edu)
# Created on: 2020-04-03
import numpy as np

eps = 1e-10


def make_pixel_grid(xlims, zlims, dx, dz):
    """ Generate a pixel grid based on input parameters. """
    x = np.arange(xlims[0], xlims[1] + eps, dx)
    z = np.arange(zlims[0], zlims[1] + eps, dz)
    xx, zz = np.meshgrid(x, z, indexing="xy")
    yy = 0 * xx
    grid = np.stack((xx, yy, zz), axis=-1)
    return grid


def make_foctx_grid(rlims, dr, oris, dirs):
    """ Generate a pixel grid based on input parameters. """
    # Get focusing positions in rho-theta coordinates
    r = np.arange(rlims[0], rlims[1] + eps, dr)  # Depth rho
    t = dirs[:, 0]  # Use azimuthal angle theta (ignore elevation angle phi)
    rr, tt = np.meshgrid(r, t, indexing="xy")

    # Convert the focusing grid to Cartesian coordinates
    xx = rr * np.sin(tt) + oris[:, [0]]
    zz = rr * np.cos(tt) + oris[:, [2]]
    yy = 0 * xx
    grid = np.stack((xx, yy, zz), axis=-1)
    return grid
```

In [11]:

```python
# File:       FocusedTxData.py
# Author:     Dongwoon Hyun (dongwoon.hyun@stanford.edu)
# Created on: 2020-04-18
import numpy as np


class FocusedTxData:
    """ A template class that contains the focused transmit data.

    FocusedTxData is a container or dataclass that holds all of the informati
    a focused transmit acquisition. Users should create a subclass that reimp
    __init__() according to how their data is stored.

    The required information is:
    idata       In-phase (real) data with shape (nxmits, nchans, nsamps)
    qdata       Quadrature (imag) data with shape (nxmits, nchans, nsamps)
    tx_ori      List of transmit origins with shape (N,3) [m]
    tx_dir      List of transmit directions with shape (N,2) [radians]
    ele_pos     Element positions with shape (N,3) [m]
    fc          Center frequency [Hz]
    fs          Sampling frequency [Hz]
    fdemod      Demodulation frequency, if data is demodulated [Hz]
    c           Speed of sound [m/s]
    time_zero   List of time zeroes for each acquisition [s]

    Correct implementation can be checked by using the validate() method.
    """

    def __init__(self):
        """ Users must re-implement this function to load their own data. """
        # Do not actually use FocusedTxData.__init__() as is.
        raise NotImplementedError

        # We provide the following as a visual example for a __init__() metho
        nxmits, nchans, nsamps = 2, 3, 4
        # Initialize the parameters that *must* be populated by child classes
        self.idata = np.zeros((nxmits, nchans, nsamps), dtype="float32")
        self.qdata = np.zeros((nxmits, nchans, nsamps), dtype="float32")
        self.tx_ori = np.zeros((nxmits, 3), dtype="float32")
        self.tx_dir = np.zeros((nxmits, 2), dtype="float32")
        self.ele_pos = np.zeros((nchans, 3), dtype="float32")
        self.fc = 5e6
        self.fs = 20e6
        self.fdemod = 0
        self.c = 1540
        self.time_zero = np.zeros((nxmits,), dtype="float32")

    def validate(self):
        """ Check to make sure that all information is loaded and valid. """
        # Check size of idata, qdata, tx_ori, tx_dir, ele_pos
        assert self.idata.shape == self.qdata.shape
        assert self.idata.ndim == self.qdata.ndim == 3
        nxmits, nchans, nsamps = self.idata.shape
        assert self.tx_ori.shape == (nxmits, 3)
        assert self.tx_dir.shape == (nxmits, 2)
        assert self.ele_pos.shape == (nchans, 3)
```

```
        # Check frequencies (expecting more than 0.1 MHz)
        assert self.fc > 1e5
        assert self.fs > 1e5
        assert self.fdemod > 1e5 or self.fdemod == 0
        # Check speed of sound (should be between 1000-2000 for medical imagi
        assert 1000 <= self.c <= 2000
        # Check that a separate time zero is provided for each transmit
        assert self.time_zero.ndim == 1 and self.time_zero.size == nxmits
```

In [12]:

```
# File:       metrics.py
# Author:     Dongwoon Hyun (dongwoon.hyun@stanford.edu)
# Created on: 2020-04-20
import numpy as np


# Compute contrast ratio
def contrast(img1, img2):
    return img1.mean() / img2.mean()


# Compute contrast-to-noise ratio
def cnr(img1, img2):
    return (img1.mean() - img2.mean()) / np.sqrt(img1.var() + img2.var())


# Compute the generalized contrast-to-noise ratio
def gcnr(img1, img2):
    _, bins = np.histogram(np.stack((img1, img2)), bins=256)
    f, _ = np.histogram(img1, bins=bins, density=True)
    g, _ = np.histogram(img2, bins=bins, density=True)
    f /= f.sum()
    g /= g.sum()
    return 1 - np.sum(np.minimum(f, g))


def res_FWHM(img):
    # TODO: Write FWHM code
    raise NotImplementedError


def speckle_res(img):
    # TODO: Write speckle edge-spread function resolution code
    raise NotImplementedError


def snr(img):
    return img.mean() / img.std()


## Compute L1 error
def l1loss(img1, img2):
    return np.abs(img1 - img2).mean()


## Compute L2 error
def l2loss(img1, img2):
    return np.sqrt(((img1 - img2) ** 2).mean())


def psnr(img1, img2):
    dynamic_range = max(img1.max(), img2.max()) - min(img1.min(), img2.min())
    return 20 * np.log10(dynamic_range / l2loss(img1, img2))


def ncc(img1, img2):
```

```
    return (img1 * img2).sum() / np.sqrt((img1 ** 2).sum() * (img2 ** 2).sum(


if __name__ == "__main__":
    img1 = np.random.rayleigh(2, (80, 50))
    img2 = np.random.rayleigh(1, (80, 50))
    print("Contrast [dB]:  %f" % (20 * np.log10(contrast(img1, img2))))
    print("CNR:            %f" % cnr(img1, img2))
    print("SNR:            %f" % snr(img1))
    print("GCNR:           %f" % gcnr(img1, img2))
    print("L1 Loss:        %f" % l1loss(img1, img2))
    print("L2 Loss:        %f" % l2loss(img1, img2))
    print("PSNR [dB]:      %f" % psnr(img1, img2))
    print("NCC:            %f" % ncc(img1, img2))
```

```
Contrast [dB]:  5.768481
CNR:            0.818935
SNR:            1.888015
GCNR:           0.453500
L1 Loss:        1.492590
L2 Loss:        1.900649
PSNR [dB]:      12.366249
NCC:            0.780980
```

In [ ]: