In [14]:

```
import numpy as np
import skimage as sk
import skimage.io as skio
from skimage import filters
import cv2 as cv
```

In [15]:

```
def trim_borders(img, percentage=0.08):
    # trim borders from the image to reduce noise for alignment
    # get dimensions
    height, width = img.shape[:2]
    # crop by a percentage
    border_height = int(height * percentage)
    border_width = int(width * percentage)
    # crop
    new_img = img[border_height:height-border_height, border_width
    return new_img
```

In [26]:

```
def edge_detect_trim(img):
    height, width = img.shape[:2]

    # convert to 8-bit
    img8 = cv.convertScaleAbs(img)

    # increase contrast
    clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    img8 = clahe.apply(img8)

    # apply gaussian blur to reduce noise
    img_blur = cv.GaussianBlur(img8, (3, 3), 0)

    # generate edge map
    edges = filters.sobel(img)

    # conver to 8-bit
    edges = cv.convertScaleAbs(edges)

    # find contours
    contours, _ = cv.findContours(edges, cv.RETR_EXTERNAL, cv.CHAI

    if not contours:
        print("no contours")
        return img

    # get the bounding box of the largest contour
    x, y, w, h = cv.boundingRect(np.concatenate(contours))

    # ensure bounding box is not too small
    if x > 0.08 * width:
        x = 0

    if y > 0.08 * height:
        y = 0

    if w < 0.92 * width:
        w = width

    if y < 0.92 * height:
        h = height

    # crop the image using the bounding box
    cropped_img = img[y:y+h, x:x+w]

    height, width = cropped_img.shape[:2]

    return cropped_img
```

In [27]:

```
def L2(image1, image2):
    l2 = np.sqrt(np.sum((image1-image2) ** 2))
    return l2
```

In [28]:

```
def L2_linalg(image1, image2):
    l2 = np.linalg.norm(image1-image2, ord=2)
    return l2
```

In [29]:

```
def NCC(image1, image2):
    # dot product between two normalized vectors:
    # (image1./||image1|| and image2./||image2||)
    A = image1 / np.linalg.norm(image1)
    B = image2 / np.linalg.norm(image2)
    d = np.sum(A * B)
#     print('d', d)
    return d
```

In [30]:

```
def SSD(image1, image2):
    return np.sum((image1 - image2) ** 2)
```

In [31]:

```
def align_channel(channel1, channel2, drange=15):
    """
    align channel2 with channel1
    exhaustively search over a window of possible displacements
    score each using image matching metric (eg, L2 norm, NCC)
    take displacement with best score
    """

#     channel1_copy = trim_borders(channel1)
#     channel2_copy = trim_borders(channel2)

    best_offset = (0, 0)
    min_score = float('inf')
    best_shifted = channel2

    # search over window of possible displacements
    for x in range(-drange, drange + 1):
        for y in range(-drange, drange + 1):
            shifted = np.roll(channel2, shift=(x, y), axis=(0, 1))
            score = SSD(channel1, shifted)

            if score < min_score:
                min_score = score
                best_offset = (x, y)
                best_shifted = shifted

#     print('naive final offset', best_offset)
    return best_offset
```

In [32]:

```
def pyramid_align(channel1, channel2, levels=10, drange=15):
    channel1_copy = channel1[:]
    channel2_copy = channel2[:]

#     channel1_copy = trim_borders(channel1)
#     channel2_copy = trim_borders(channel2)

    pyr_channel1 = [channel1_copy]
    pyr_channel2 = [channel2_copy]

    # build image pyramid
    real_levels = 0
    for l in range(levels):
        if pyr_channel1[-1].size < 32:
            break
        real_levels += 1
        pyr_channel1.append(cv.resize(pyr_channel1[-1], (0, 0), fx
        pyr_channel2.append(cv.resize(pyr_channel2[-1], (0, 0), fx

    # default offset is 0
    offset = (0, 0)

    # iterate from coarsest to finest
    for level in range(real_levels - 1, -1, -1):
        pc1 = pyr_channel1[level]
        pc2 = pyr_channel2[level]

        offset = (2 * offset[0], 2 * offset[1])

        # shift by previous offset
        pc2 = np.roll(pc2, shift=offset, axis=(0,1))

        # get new offset
        new_offset = align_channel(pc1, pc2, drange)

        offset = (offset[0] + new_offset[0], offset[1] + new_offse

#     print('pyramid final offset', offset)
#     final_shifted = np.roll(channel2_copy, shift=offset, axis=(0
    return offset
```

In [37]:

```
def edge_align(channel1, channel2, levels=10, drange=15):
#     channel1_copy = trim_borders(channel1)
#     channel2_copy = trim_borders(channel2)

    channel1_copy = channel1[:]
    channel2_copy = channel2[:]

    edges1 = filters.sobel(channel1_copy)
    edges2 = filters.sobel(channel2_copy)

    offset = pyramid_align(edges1, edges2, levels, drange)
    print('edge final offset', offset)

    final_shifted = np.roll(channel2_copy, shift=offset, axis=(0,1

    return final_shifted
```

In [34]:

```
def color_image(imname="data/cathedral.jpg"):
    print('imname', imname)

    # read in the image
    im = skio.imread(imname)

    # convert to double (might want to do this later on to save me
    im = sk.img_as_float(im)

    # compute the height of each part (just 1/3 of total)
    height = np.floor(im.shape[0] / 3.0).astype(int)

    # separate color channels
    b = im[:height]
    g = im[height: 2*height]
    r = im[2*height: 3*height]

    # trim before aligning
    b = edge_detect_trim(b)
    g = edge_detect_trim(g)
    r = edge_detect_trim(r)

    height = min(b.shape[0], g.shape[0], r.shape[0])
    width = min(b.shape[1], g.shape[1], r.shape[1])

    b = b[:height, :width]
    g = g[:height, :width]
    r = r[:height, :width]

    # align
    ag = edge_align(b, g)
    ar = edge_align(b, r)
#     b = trim_borders(b)

    # create a color image
    im_out = np.dstack([ar, ag, b])

    # save the image
    fname = './out_path/edge_align_edge_crop/out_{}.jpg'.format(im
    skio.imsave(fname, im_out)

    # display the image
#     skio.imshow(im_out)
#     skio.show()
```

In [35]:

```python
def color_all():
    import os

    # Get the list of all files and directories
    path = "data/"
    dir_list = os.listdir(path)
    dir_list = [path + d for d in dir_list]

    dir_list.remove('data/.DS_Store')

    for f in dir_list:
        color_image(f)
```

```
In [39]: color_all()
```

```
imname data/emir.tif
edge final offset (-9356, -10591)
edge final offset (-9298, -10574)

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.

imname data/monastery.jpg
edge final offset (-344, -379)

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.

edge final offset (-338, -380)
imname data/church.tif
edge final offset (-9473, -10561)
edge final offset (-9440, -10603)

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.

imname data/three_generations.tif
edge final offset (-8603, -10479)
edge final offset (-8745, -10480)

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.

imname data/melons.tif
edge final offset (-9325, -10626)
edge final offset (-9293, -7084)

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.

imname data/onion_church.tif
no contours
no contours
no contours
edge final offset (-9593, -7538)
edge final offset (-9538, -7527)

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.

imname data/train.tif
edge final offset (-9521, -10854)
edge final offset (-9477, -10757)

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.

imname data/tobolsk.jpg
edge final offset (-338, -383)
```

```
Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.

edge final offset (-335, -384)
imname data/icon.tif
edge final offset (-9069, -10612)
edge final offset (-8814, -10446)

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.

imname data/cathedral.jpg
edge final offset (-336, -373)

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.

edge final offset (-329, -373)
imname data/self_portrait.tif
edge final offset (-9238, -10588)
edge final offset (-9079, -10578)

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.

imname data/harvesters.tif
no contours
no contours
no contours
edge final offset (-9583, -11055)
edge final offset (-9468, -11046)

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.

imname data/sculpture.tif
edge final offset (-9436, -10959)
edge final offset (-9390, -10977)

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.

imname data/lady.tif
edge final offset (-9303, -10710)
edge final offset (-9237, -10734)

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.
```

```
In [38]: color_image()
```

```
imname data/cathedral.jpg
edge final offset (-336, -373)

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image t
o uint8 prior to saving to suppress this warning.

edge final offset (-329, -373)
```

In [ ]:

```
```