PRAKHAR BHARDWAJ

ANDREW ID - prakharb

Question 1: Homography Theory

Suppose we have two cameras $\mathbf{C}_1$ and $\mathbf{C}_2$ looking at a common plane Π in 3D space. Any 3D point $\mathbf{P}$ on Π generates a projected 2D point located at $\mathbf{p} \equiv (u_1, v_1, 1)^T$ on the first camera $\mathbf{C}_1$ and $\mathbf{q} \equiv (u_2, v_2, 1)^T$ on the second camera $\mathbf{C}_2$. Since $\mathbf{P}$ is confined to the plane Π, we expect that there is a relationship between $\mathbf{p}$ and $\mathbf{q}$. In particular, there exists a common 3 × 3 matrix $\mathbf{H}$, so that for any $\mathbf{p}$ and $\mathbf{q}$, the following conditions holds:

$$\mathbf{p} \equiv \mathbf{Hq}$$

We call this relationship **planar homography**. Recall that both $\mathbf{p}$ and $\mathbf{q}$ are in homogeneous coordinates and the equality $\equiv$ means $\mathbf{p}$ is proportional to $\mathbf{Hq}$ (recall homogeneous coordinates). It turns out this relationship is also true for cameras that are related by pure rotation without the planar constraint.

1.1 Homography (5 points)

Prove that there exists an $\mathbf{H}$ that satisfies $\mathbf{p} \equiv \mathbf{Hq}$ given two 3×4 camera projection matrices $\mathbf{M_1}$ and $\mathbf{M_2}$ corresponding to cameras $\mathbf{C_1}$, $\mathbf{C_2}$ and a plane Π. Do not produce an actual algebraic expression for $\mathbf{H}$. All we are asking for is a proof of the existence of $\mathbf{H}$.

Note: A degenerate case may happen when the plane Π contains both cameras' centers, in which case there are infinite choices of $\mathbf{H}$ satisfying $\mathbf{p} \equiv \mathbf{Hq}$. You can ignore this case in your answer.

Consider the projection of 3D homogeneous point P on the plane Π to the two camera planes as p and q

$\mathbf{p} \equiv \mathbf{M_1 P}$

$\mathbf{q} \equiv \mathbf{M_2 P}$

Since p and q are related to the same planar point P by a system of linear equations, there exists a system of linear equations that relates p and q as linear transformationsare closed under composition.

Therefore, there exists $\mathbf{H}$ that satisfy $\mathbf{p} \equiv \mathbf{Hq}$

1.2 Homography under rotation (5 points)

1.2 Homography under rotation (5 points)

Prove that there exists a homography $\mathbf{H}$ that satisfies $\mathbf{p_1} \equiv \mathbf{Hp_2}$, given two cameras separated by a pure rotation. That is, for camera 1, $\mathbf{p_1} = \mathbf{K_1} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \mathbf{P}$ and for camera 2, $\mathbf{p_2} = \mathbf{K_2} \begin{bmatrix} \mathbf{R} & \mathbf{0} \end{bmatrix} \mathbf{P}$. Note that $\mathbf{K_1}$ and $\mathbf{K_2}$ are the 3×3 intrinsic matrices of the two cameras and are different. $\mathbf{I}$ is 3×3 identity matrix, $\mathbf{0}$ is a 3×1 zero vector and $\mathbf{P}$ is the homogeneous coordinate of a point in 3D space. $\mathbf{R}$ is the 3×3 rotation matrix of the camera.

Writing homogeneous coordinates $\mathbf{P} \equiv \begin{bmatrix} \mathbf{X} \\ \mathbf{1} \end{bmatrix}$, where $\mathbf{X}$ are the 3D world coordinates

$$\mathbf{P_1} \equiv \mathbf{K_1} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ \mathbf{1} \end{bmatrix} \equiv \mathbf{K_1 X}$$

$$\mathbf{P_2} \equiv \mathbf{K_2} \begin{bmatrix} \mathbf{R} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ \mathbf{1} \end{bmatrix} \equiv \mathbf{K_2 R X}$$

For the second equation, $\mathbf{X} \equiv (\mathbf{K_2 R})^{-1} \mathbf{p_2}$

Substituting in the first equation, $\mathbf{p_1} \equiv (\mathbf{K_1 X}) = (\mathbf{K_1})(\mathbf{K_2 R})^{-1} \mathbf{p_2}$

Therefore,

$$\mathbf{p_1} \equiv \mathbf{K_1 R^{-1} K_2^{-1} p_2}$$

where $\mathbf{K_1 R^{-1} K_2^{-1}}$ is a 3x3 matrix

Hence can be seen thatthere exists a 3X3 homography $\mathbf{H} = \mathbf{K_1 R^{-1} K_2^{-1}}$ from $\mathbf{p_2}$ to $\mathbf{p_1}$

1.3 Correspondences (10 points)

Let $\mathbf{x_1}$ be a set of points in an image and $\mathbf{x_2}$ be the set of corresponding points in an image taken by another camera. Suppose there exists a homography $\mathbf{H}$ such that:
$\mathbf{x_1^i} \equiv \mathbf{H x_2^i} \quad (i \in \{1 \ldots N\})$ where $\mathbf{x_1^i} = \begin{bmatrix} x_1^i & y_1^i & 1 \end{bmatrix}^T$ are in homogenous coordinates, $\mathbf{x_1^i} \in \mathbf{x_1}$ and $\mathbf{H}$ is a 3×3 matrix. For each point pair, this relation can be rewritten as

$$\mathbf{A_i h} = 0$$

where $\mathbf{h}$ is a column vector reshaped from $\mathbf{H}$, and $\mathbf{A_i}$ is a matrix with elements derived from the points $\mathbf{x_1^i}$ and $\mathbf{x_2^i}$. This can help calculate **H** from the given point correspondences.

- How many degrees of freedom does $\mathbf{h}$ have? (3 points)
- How many point pairs are required to solve $\mathbf{h}$? (2 points)
- Derive $\mathbf{A_i}$. (5 points)

Here, **h** has 8 degree of freedom. Thus, the number of point pairs to solve for **h** is 4.

Given, $\mathbf{x_1^i} \equiv \mathbf{Hx_2^i}$

$$\begin{bmatrix} x_1^i \\ y_1^i \\ 1 \end{bmatrix} \equiv \begin{bmatrix} h11 & h12 & h13 \\ h21 & h22 & h23 \\ h31 & h32 & h33 \end{bmatrix} \begin{bmatrix} x_2^i \\ y_2^i \\ 1 \end{bmatrix}$$

Simplifying the RHS term

$$\begin{bmatrix} h11 & h12 & h13 \\ h21 & h22 & h23 \\ h31 & h32 & h33 \end{bmatrix} \begin{bmatrix} x_2^i \\ y_2^i \\ 1 \end{bmatrix} = \begin{bmatrix} h11 * x_2^i + h12 * y_2^i + h13 \\ h21 * x_2^i + h22 * y_2^i + h23 \\ h31 * x_2^i + h32 * y_2^i + h33 \end{bmatrix}$$

Make the last element 1

$$\begin{bmatrix} x_1^i \\ y_1^i \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{h11 * x_2^i + h12 * y_2^i + h13}{h31 * x_2^i + h32 * y_2^i + h33} \\ \frac{h21 * x_2^i + h22 * y_2^i + h23}{h31 * x_2^i + h32 * y_2^i + h33} \\ 1 \end{bmatrix}$$

$$\therefore x_1^i = \frac{h11 * x_2^i + h12 * y_2^i + h13}{h31 * x_2^i + h32 * y_2^i + h33} \therefore y_1^i = \frac{h21 * x_2^i + h22 * y_2^i + h23}{h31 * x_2^i + h32 * y_2^i + h33}$$

$$\therefore (h31 * x_2^i + h32 * y_2^i + h33)x_1^i = h11 * x_2^i + h12 * y_2^i + h13$$
$$\therefore (h31 * x_2^i + h32 * y_2^i + h33)y_1^i = h21 * x_2^i + h22 * y_2^i + h23$$

$$\therefore -h11 * x_2^i - h12 * y_2^i - h13 + x_1^i x_2^i * h31 + x_1^i y_2^i * h32 + x_1^i * h33 = 0$$
$$\therefore -h21 * x_2^i - h22 * y_2^i - h23 + y_1^i x_2^i * h31 + y_1^i y_2^i * h32 + y_1^i * h33 = 0$$

Writing this in matrix form,

$$\begin{bmatrix} -x_2^i & -y_2^i & -1 & 0 & 0 & 0 & x_1^i x_2^i & x_1^i y_2^i & x_1^i \\ 0 & 0 & 0 & -x_2^i & -y_2^i & -1 & y_1^i x_2^i & y_1^i y_2^i & y_1^i \end{bmatrix} \begin{bmatrix} h11 \\ h12 \\ h13 \\ h21 \\ h22 \\ h23 \\ h31 \\ h32 \\ h33 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Hence, $A_i = \begin{bmatrix} -x_2^i & -y_2^i & -1 & 0 & 0 & 0 & x_1^i x_2^i & x_1^i y_2^i & x_1^i \\ 0 & 0 & 0 & -x_2^i & -y_2^i & -1 & y_1^i x_2^i & y_1^i y_2^i & y_1^i \end{bmatrix}$

1.4 Understanding homographies under rotation (5 points)

Suppose that a camera is rotating about its center $\mathbf{C}$, keeping the intrinsic parameters $\mathbf{K}$ constant. Let $\mathbf{H}$ be the homography that maps the view from one camera orientation to the view at a second orientation. Let θ be the angle of rotation between the two. Show that $\mathbf{H^2}$ is the homography corresponding to a rotation of 2θ. Please limit your answer within a couple of lines. A lengthy proof indicates that you're doing something too complicated (or wrong).

If the camera is only being rotated around its center, $\mathbf{H}$ would be of the form

$$\mathbf{H} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Computing, $\mathbf{H}^2$

$$\mathbf{H}^2 = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos^2(\theta) - \sin^2(\theta) & -(2\sin(\theta) \\ 2\sin(\theta)\cos(\theta)) & \cos^2(\theta) - \\ 0 & 0 \end{bmatrix}$$

$\mathbf{H}^2$ is the homography corresponding to a rotation of 2θ.

1.5 Limitations of the planar homography (2 points)

Why is the planar homography not completely sufficient to map any arbitrary scene image to another viewpoint? State your answer concisely in one or two sentences.

When there is a substantial depth involved, this assumes that scene points will be on a plane, which isn't always possible. Also, translation could result in an occluded viewpoint where the scene's or object's full visibility is lost.

1.6 Behavior of lines under perspective projections (3 points)

We stated in class that perspective projection preserves lines (a line in 3D is projected to a line in 2D). Verify algebraically that this is the case, i.e., verify that the projection $\mathbf{P}$ in $\mathbf{x} = \mathbf{PX}$ preserves lines.

Lets say that the 3D line is given by $(x_0, y_0, z_0) + t(a, b, c)$ and (a, b, c) represent the direction vector, (x_0, y_0, z_0) represents one point lying on the line and $t \in \mathbb{R}$ is the stepping variable that helps satisfy all points lying on this line.

$$\therefore \mathbf{X} = \begin{bmatrix} t * a + x_0 \\ t * b + y_0 \\ t * c + z_0 \end{bmatrix}$$

The projection matrix can be broken down into intrinsic and extrinsic parameter matrices. Assuming that there's no rotation or translation,

$$\mathbf{x} \equiv \alpha \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} t * a + x_0 \\ t * b + y_0 \\ t * c + z_0 \end{bmatrix}$$

α is the scaling factor

$$\mathbf{x} \equiv \alpha \begin{bmatrix} f_x * (t * a + x_0) + o_x * (t * c + z_0) \\ f_y * (t * b + y_0) + o_y * (t * c + z_0) \\ t * c + z_0 \end{bmatrix}$$

Making them homogeneous :

$$\mathbf{x} = \alpha \begin{bmatrix} \frac{f_x * (t * a + x_0) + o_x * (t * c + z_0)}{t * c + z_0} \\ \frac{f_y * (t * b + y_0) + o_y * (t * c + z_0)}{t * c + z_0} \\ 1 \end{bmatrix}$$

$$\therefore x = \alpha * f_x * \frac{t * a + x_0}{t * c + z_0} + \alpha * o_x \therefore y = \alpha * f_y * \frac{t * b + y_0}{t * c + z_0} + \alpha * o_y$$

Consider 2 3-D points lying on the 3-D line - $(x_0, y_0, z_0) + t_1(a, b, c)$ and $(x_0, y_0, z_0) + t_2(a, b, c)$

The corresponding 2D coordinates would be given by

$$\therefore x_1 = \alpha * f_x * \frac{t_1 * a + x_0}{t_1 * c + z_0} + \alpha * o_x \therefore y_1 = \alpha * f_y * \frac{t_1 * b + y_0}{t_1 * c + z_0} + \alpha * o_y$$

and

$\therefore x_2 = \alpha * f_x * \frac{t_2 * a + x_0}{t_2 * c + z_0} + \alpha * o_x \therefore y_2 = \alpha * f_y * \frac{t_2 * b + y_0}{t_2 * c + z_0} + \alpha * o_y$

We Find the slope of line between these points

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\left[\alpha * f_y * \frac{t_2 * b + y_0}{t_2 * c + z_0} + \alpha * o_y\right] - \left[\alpha * f_y * \frac{t_1 * b + y_0}{t_1 * c + z_0} + \alpha * o_y\right]}{\left[\alpha * f_x * \frac{t_2 * a + x_0}{t_2 * c + z_0} + \alpha * o_x\right] - \left[\alpha * f_x * \frac{t_1 * a + x_0}{t_1 * c + z_0} + \alpha * o_x\right]}$$

Simplifying,

$$m = \frac{\left[f_y * (t_2 * b + y_0)(t_1 * c + z_0)\right] - \left[f_y * (t_1 * b + y_0)(t_2 * c + z_0)\right]}{\left[f_x * (t_2 * a + x_0)(t_1 * c + z_0)\right] - \left[f_x * (t_1 * a + x_0)(t_2 * c + z_0)\right]} = \frac{f_y}{f_x}\left[\frac{t_1 t_2 bc + t_1 c y_0 + t_2 b z_0 + y_0 z_0 - t_1 t_2 bc - t_2 c y_0 - t_1 b z_0 - y_0 z_0}{t_1 t_2 ac + t_1 c x_0 + t_2 a z_0 + x_0 z_0 - t_1 t_2 ac - t_2 c x_0 - t_1 a z_0 - x_0 z_0}\right.$$

$$m = \frac{f_y}{f_x}\left[\frac{c y_0 (t_1 - t_2) - b z_0 (t_1 - t_2)}{c x_0 (t_1 - t_2) - a z_0 (t_1 - t_2)}\right] = \frac{f_y}{f_x} \frac{(t1 - t2)}{(t1 - t2)} \left[\frac{c y_0 - b z_0}{c x_0 - a z_0}\right]$$

$$\therefore m = \frac{f_y}{f_x}\left[\frac{c y_0 - b z_0}{c x_0 - a z_0}\right]$$

The slope, m, is independent of the variable t. Hence, for any point which lies on the 3D line would be projected to this 2D line with slope m using the projection matrix, $\mathbf{P}$.

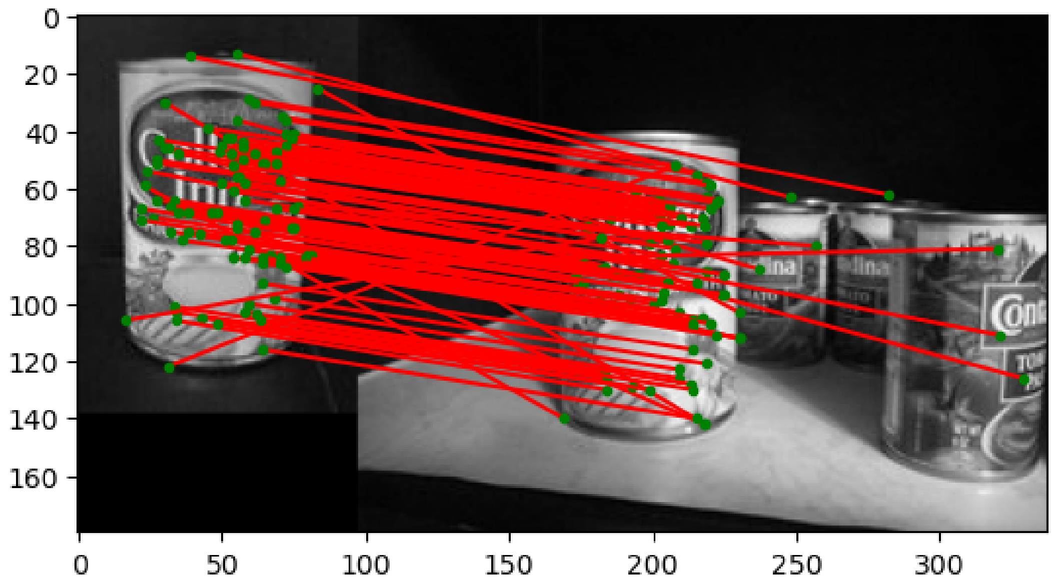
2.4 Check Point: Descriptor Matching (5 pts)

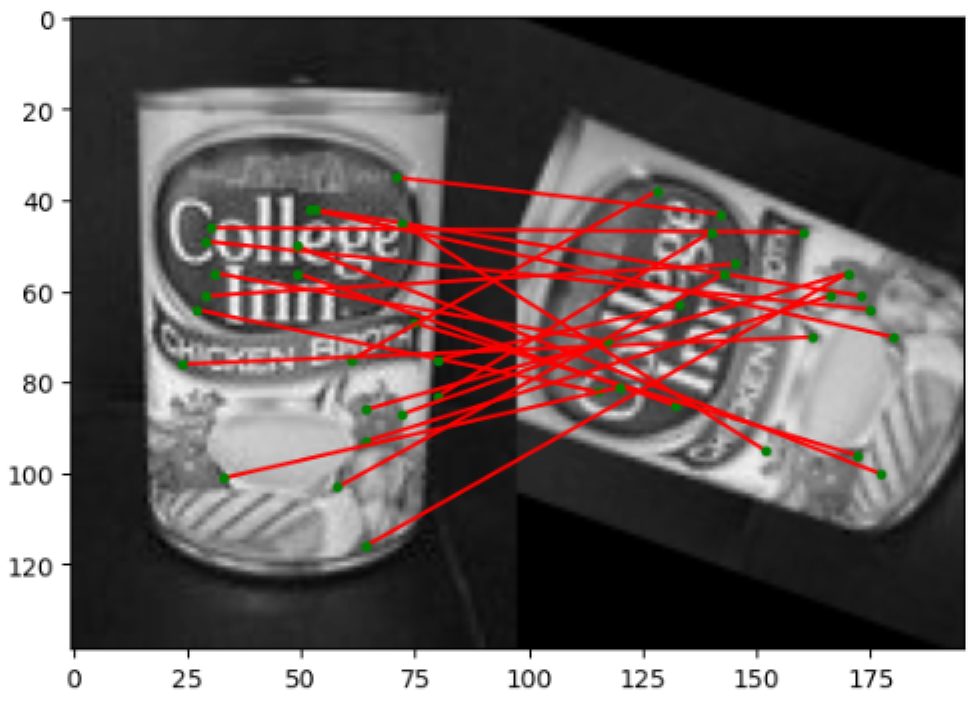
Save the resulting figure and submit it in your PDF. Briefly discuss any cases that perform worse or better.

In [ ]:

```
def briefMatch(desc1, desc2, ratio=0.8):

    D = cdist(np.float32(desc1), np.float32(desc2), metric='hamming')
    ix2 = np.argmin(D, axis=1)
    d1 = D.min(1)
    d12 = np.partition(D, 2, axis=1)[:,0:2]
    d2 = d12.max(1)
    r = d1/(d2+1e-10)
    is_discr = r<ratio
    ix2 = ix2[is_discr]
    ix1 = np.arange(D.shape[0])[is_discr]
    matches = np.stack((ix1,ix2), axis=-1)
    return matches

def plotMatches(im1, im2, matches, locs1, locs2):
    fig = plt.figure()
    # draw two images side by side
    imH = max(im1.shape[0], im2.shape[0])
    im = np.zeros((imH, im1.shape[1]+im2.shape[1]), dtype='uint8')
    im[0:im1.shape[0], 0:im1.shape[1]] = cv2.cvtColor(im1, cv2.COLOR_BGR2GRAY)
    im[0:im2.shape[0], im1.shape[1]:] = cv2.cvtColor(im2, cv2.COLOR_BGR2GRAY)
    plt.imshow(im, cmap='gray')
    for i in range(matches.shape[0]):
        pt1 = locs1[matches[i,0], 0:2]
        pt2 = locs2[matches[i,1], 0:2].copy()
        pt2[0] += im1.shape[1]
        x = np.asarray([pt1[0], pt2[0]])
        y = np.asarray([pt1[1], pt2[1]])
        plt.plot(x,y,'r')
        plt.plot(x,y,'g.')
    plt.show()

img1 = cv2.imread('data/model_chickenbroth.jpg')
# img1 = cv2.imread('data/chickenbroth_01.jpg')
locs1, desc1 = briefLite(img1)

img2 = cv2.imread('data/chickenbroth_01.jpg')
locs2, desc2 = briefLite(img2)

matches = briefMatch(desc1, desc2)
plotMatches(img1, img2, matches, locs1, locs2)
```

BRIEF descriptors are not rotational and scale invariant. Therefore, the descriptors developed between these two photos wouldn't match if the same image were rotated or when there is a scale change.





The above image is rotated by the angle of 70 degrees and it does not performs well because of the general fact that the BRIEF descriptor is not rotational and scale invarient.

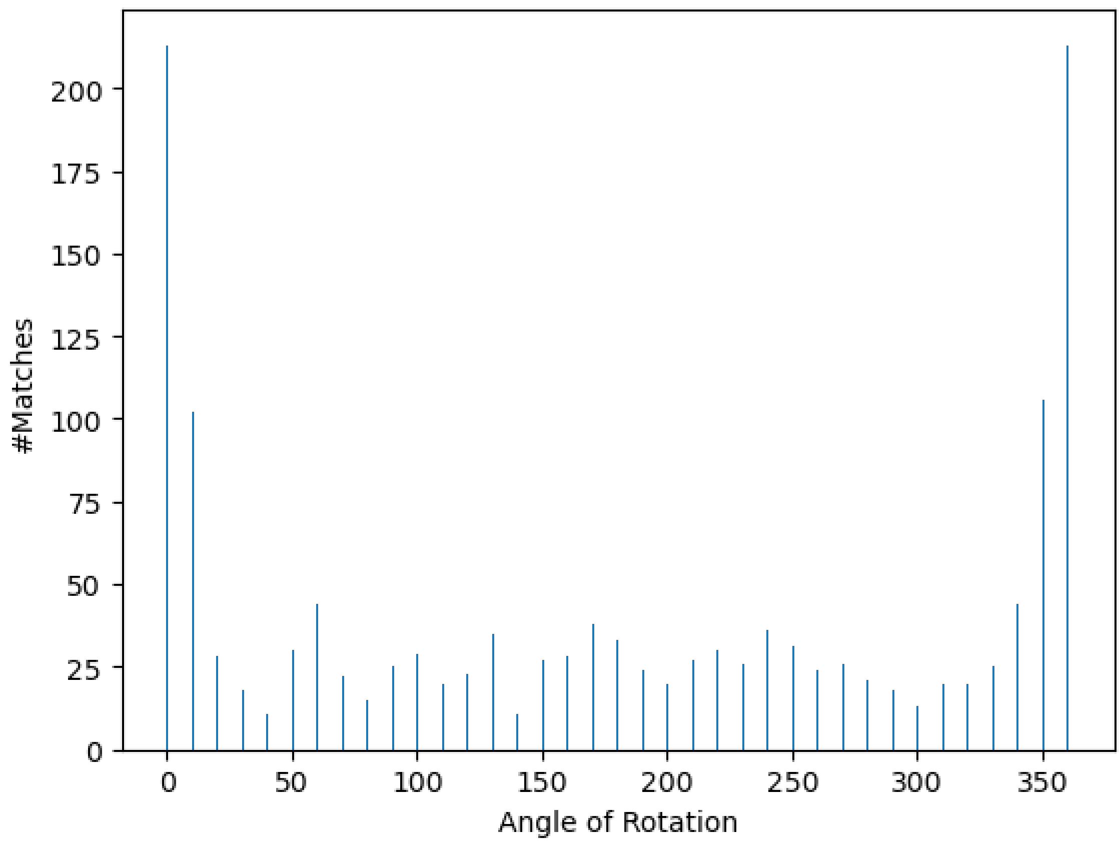
2.5 BRIEF and rotations (5 pts)

Include your code and the historgram figure in your PDF, and explain why you think the descriptor behaves this way.

In [ ]:

```
img1 = cv2.imread('data/model_chickenbroth.jpg')
locs1, desc1 = briefLite(img1)

img2 = cv2.imread('data/model_chickenbroth.jpg')
h, w, _ = img2.shape

counts = []

for i in range(0, 370, 10):


    M = cv2.getRotationMatrix2D((w / 2, h / 2), i, 1.0)
    rotated = cv2.warpAffine(img2, M, (w, h))
    locs2, desc2 = briefLite(rotated)
    matches = briefMatch(desc1, desc2)
    if i==70:
        plotMatches(img1, rotated, matches, locs1, locs2)
    counts.append(len(matches))

print(len(list(range(0, 370, 10))), len(counts))
plt.bar(list(range(0, 370, 10)), counts, align='edge')
plt.xlabel('Angle of Rotation')
plt.ylabel('#Matches')
plt.show()
```

The descriptor analyzes n pairs of pixels in a patch surrounding a keypoint to function. The pixel placements within the patch have changed locally as a result of rotation. As a result, the pixels that the n pairings used to correspond to in the unrotated image are no longer the same.



2.6 Improving Performance - (Extra Credit, 10 pts)

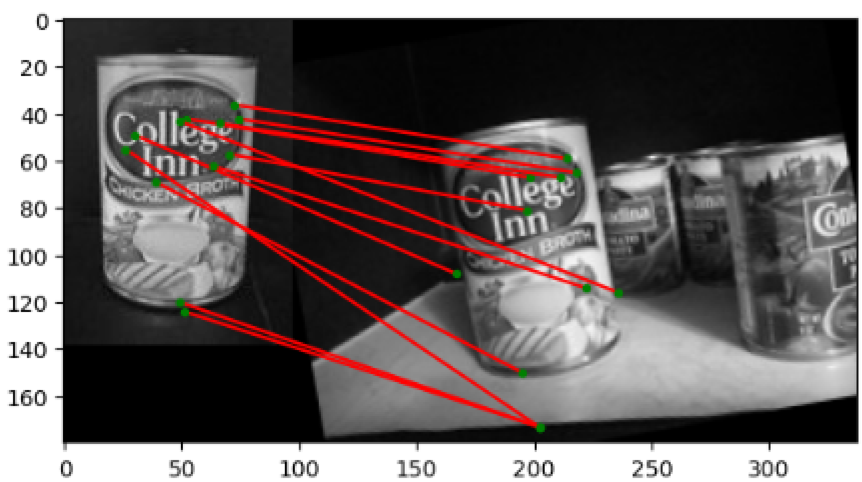
The extra credit opportunities described below are optional and provide an avenue to explore computer vision and improve the performance of the techniques developed above.

1. (**5 pts**) As we have seen, BRIEF is not rotation invariant. Design a simple fix to solve this problem using the tools you have developed so far (think back to edge detection and/or Harris corner's covariance matrix). Include yout code in your PDF, and explain your design decisions and how you selected any parameters that you use. Demonstrate the effectiveness of your algorithm on image pairs related by large rotation.
2. (**5 pts**) This implementation of BRIEF has some scale invariance, but there are limits. What happens when you match a picture to the same picture at half the size? Look to section 3 of [Lowe2004 (https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf)](https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf) for a technique that will make your detector more robust to changes in scale. Implement it and demonstrate it in action with several test images. Include yout code and the test images in your PDF. You may simply rescale some of the test images we have given you.

In [ ]:

```
def computeRBrief(gaussPyramid, locsDoG, compareX, compareY, patch_width=9):

    h, w, _ = gaussPyramid.shape
    left_limit = patch_width // 2
    right_limit_h = gaussPyramid.shape[0] - patch_width // 2
    right_limit_w = gaussPyramid.shape[1] - patch_width // 2

    valid_locsDoG = locsDoG[np.logical_and(np.logical_and(locsDoG[:, 0] > left_
                                                          locsDoG[:, 0] < right
                                           np.logical_and(locsDoG[:, 1] > left_
                                                          locsDoG[:, 1] < right

    desc = []

    for i in range(valid_locsDoG.shape[0]):
        patch = gaussPyramid[valid_locsDoG[i, 0] - patch_width // 2: valid_locs
                             valid_locsDoG[i, 1] - patch_width // 2: valid_locs
                             valid_locsDoG[i, 2]]

        m10 = np.sum(np.repeat(np.arange(valid_locsDoG[i, 0] - patch_width // 2
                                         valid_locsDoG[i, 0] + patch_width // 2
                               patch_width, axis=0) * patch)
        m01 = np.sum(np.repeat(np.arange(valid_locsDoG[i, 1] - patch_width // 2
                                         valid_locsDoG[i, 1] + patch_width // 2
                               patch_width, axis=0) * patch)
        m00 = np.sum(patch)
        theta = - np.arctan2(m01, m10)
        R = np.array([[np.cos(theta), - np.sin(theta)],
                      [np.sin(theta), np.cos(theta)]])

        compareMat = np.int32(R @ np.stack((compareX.flatten(), compareY.flatte

        tau = gaussPyramid[(int(valid_locsDoG[i, 0] - patch_width // 2) + compa
                           (int(valid_locsDoG[i, 1] - patch_width // 2) + compa
                           valid_locsDoG[i, 2]] < \
              gaussPyramid[(int(valid_locsDoG[i, 0] - patch_width // 2) + compa
                           (int(valid_locsDoG[i, 1] - patch_width // 2) + compa
                           valid_locsDoG[i, 2]]
        desc.append(list(map(lambda j: str(int(tau[j])), range(len(tau)))))

    newlocs = valid_locsDoG[:, -2: -4: -1]
    desc = np.stack(desc)

    return newlocs, desc

def rBriefLite(im):

    locsDoG, gauss_pyramid = DoGdetector(im)
```

```
    [compareX, compareY] = np.load('data/testPattern.npy')

    locs, desc = computeRBrief(gauss_pyramid, locsDoG, compareX, compareY)

    return locs, desc

img1 = cv2.imread('data/model_chickenbroth.jpg')
locs1, desc1 = rBriefLite(img1)

img2 = cv2.imread('data/chickenbroth_01.jpg')

h, w, _ = img2.shape

M = cv2.getRotationMatrix2D((w / 2, h / 2), 10, 1.0)
rotated = cv2.warpAffine(img2, M, (w, h))
locs2, desc2 = rBriefLite(rotated)

matches = briefMatch(desc1, desc2)
plotMatches(img1, rotated, matches, locs1, locs2)
```



Similar to what the ORB descriptor does, the concept of solving for rotation invariance was developed. We Take the patch, discover its rotation, and, by rotating it at the same angle in the other direction, discover the new points to compare. It would assist in resolving the patch rotation problem.

Using a guassian pyramid would be a solution for scale invariance. A scale-invariant system would result from using the gaussian pyramid of one image and comparing it to combinations of the other image and various pyramid levels.

3.3 Automated Homography Estimation/Warping for Augmented Reality (10 points)

Reality (10 points)

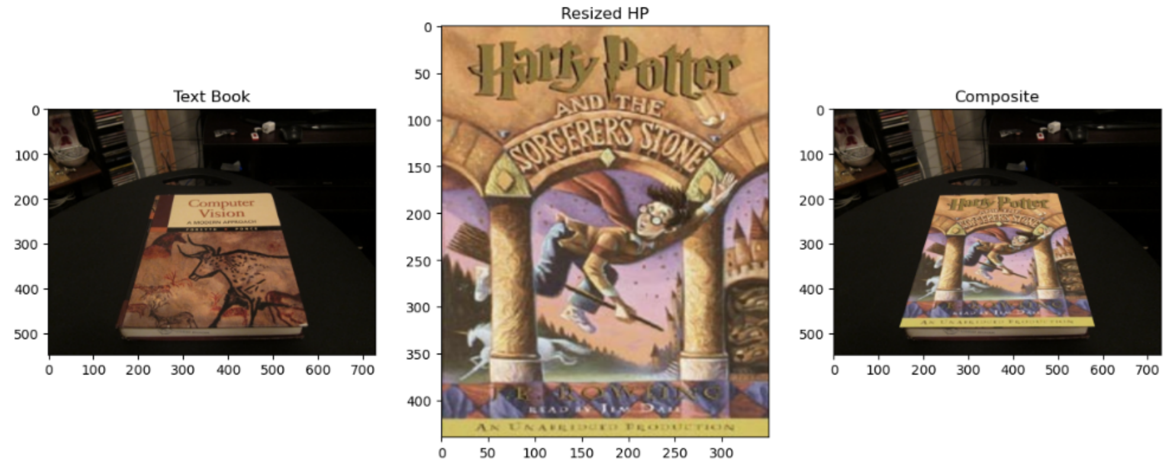
Implement the following steps:

1. Reads `cv-cover.jpg`, `cv-desk.png`, and `hp-cover.jpg`.
2. Computes a homography automatically using `computeH-ransac`.
3. Warps `hp-cover.jpg` to the dimensions of the `cv-desk.png` image using the OpenCV `warpPerspective` function.
4. At this point you should notice that although the image is being warped to the correct location, it is not filling up the same space as the book. Why do you think this is happening? How would you modify `hp-cover.jpg` to fix this issue?
5. Implement the function:
   `function [ composite-img ] = compositeH( H2to1, template, img)` to now compose this warped image with the desk image as in the following figures.
6. Include your resulting image in your write-up. Please also print the final H matrix in your writeup (normalized so the bottom right value is 1)

In [ ]:

```
def compositeH(H, template, img):

    warped_img = cv2.warpPerspective(template, np.linalg.inv(H), img.shape[-2:
    composite_img = np.uint8(warped_img == 0) * img + warped_img

    return composite_img.astype(np.uint8)

im1 = cv2.cvtColor(cv2.imread("figure/cv_cover.jpg"), cv2.COLOR_BGR2RGB)
im2 = cv2.cvtColor(cv2.imread("figure/cv_desk.jpg"), cv2.COLOR_BGR2RGB)

locs1, desc1 = briefLite(im1)
locs2, desc2 = briefLite(im2)
matches = briefMatch(desc1, desc2)

H, inliers = computeH_ransac(matches, locs1, locs2)

print('H: {0}'.format(H / H[2, 2]))

template = cv2.cvtColor(cv2.imread("figure/hp_cover.jpg"), cv2.COLOR_BGR2RGB)

resized_template = cv2.resize(template, im1.shape[-2: -4: -1])

composite_img = compositeH(H, resized_template, im2)

fig, axes = plt.subplots(1, 3, figsize=(15, 15))

axes[0].imshow(im2)
axes[0].set_title('Text Book')
axes[1].imshow(resized_template)
axes[1].set_title('Resized HP')
axes[2].imshow(composite_img)
axes[2].set_title('Composite')
plt.show()
```

```
H: [[ 2.48545989e+00  7.60759698e-01 -7.34723204e+02]
 [ 7.94604405e-02  4.57290255e+00 -9.01915954e+02]
 [ 1.53968563e-04  4.13264430e-03  1.00000000e+00]]
```



4.1 Image Stitching (5 pts)

Visualize the warped image. Please include the image and your H2to1 matrix (with the bottom right index as 1) in your writeup PDF, along with stating which image pair you used.

In [ ]:

```
def imageStitching(im1, im2, H2to1, pad=((0, 0), (0, 500), (0, 0))):
    '''
    Returns a panorama of im1 and im2 using the given
    homography matrix

    INPUT
        Warps img2 into img1 reference frame using the provided warpH() functio
        H2to1 - a 3 x 3 matrix encoding the homography that best matches the li
                 equation.
    OUTPUT
        img_pano - the panorama image.
    '''

    padded_im1 = np.pad(im1, pad)
    warped_img = cv2.warpPerspective(im2, H2to1, padded_im1.shape[-2: -4: -1])

    w1 = distance_transform_edt(np.pad(np.ones_like(im1[1: -1, 1: -1, 0]), 1))
    w2 = distance_transform_edt(np.pad(np.ones_like(im2[1: -1, 1: -1, 0]), 1))

    padded_w1 = np.pad(w1, pad[0: 2])
    padded_w1 = padded_w1.reshape(padded_w1.shape[0], -1, 1)

    warped_w2 = cv2.warpPerspective(w2, H2to1, padded_im1.shape[-2: -4: -1])
    warped_w2 = warped_w2.reshape(warped_w2.shape[0], -1, 1)

    img_pano = np.uint8((padded_im1 * padded_w1 + warped_img * warped_w2) / (pa

    return img_pano

im1 = cv2.cvtColor(cv2.imread('data/incline_L.png'), cv2.COLOR_BGR2RGB)
im2 = cv2.cvtColor(cv2.imread('data/incline_R.png'), cv2.COLOR_BGR2RGB)

locs1, desc1 = briefLite(im1)
locs2, desc2 = briefLite(im2)
matches = briefMatch(desc1, desc2)

H2to1, inliers = computeH_ransac(matches, locs1, locs2)

print('H2to1: {0}'.format(H2to1 / HH2to1[2, 2]))

img_pano = imageStitching(im1, im2, H2to1)

plt.rcParams['figure.figsize'] = (15, 15)
plt.imshow(img_pano)
plt.show()
```

Image pair used are **incline_L.png** and **incline_R.png**

```
H2to1: [[ 6.60853848e-01 -2.57012412e-02  3.60687370e+02]
 [-7.84187162e-02  8.84975759e-01 -1.87915675e+01]
 [-3.54057040e-04  4.80171932e-07  1.00000000e+00]]
```

```
C:\Users\CH\AppData\Local\Temp\ipykernel_9540\814953079.py:30: RuntimeWarning: invalid value encountered in true_divide
  img_pano = np.uint8((padded_im1 * padded_w1 + warped_img * warped_w2) / (padded_w1 + warped_w2))
```

4.2 Image Stitching with No Clip (3 pts)

Visualize the warped image. Please include the image in your writeup PDF, along with stating which image pair you used.

In [ ]:

```
def imageStitching_noClip(im1, im2, H2to1, pad=500, isPadWidth=True, M=np.array

    if isPadWidth:
        w, h = im1.shape[1] + pad, (im1.shape[0] * (im1.shape[1] + pad)) // im1
    else:
        w, h = (im1.shape[1] * (im1.shape[0] + pad)) // im1.shape[0], im1.shape

    out_size = (w, h)

    warp_im1 = cv2.warpPerspective(im1, M, out_size)
    warp_im2 = cv2.warpPerspective(im2, M @ H2to1, out_size)

    plt.imshow(warp_im1)
    w1 = distance_transform_edt(np.pad(np.ones_like(im1[1: -1, 1: -1, 0]), 1))
    w2 = distance_transform_edt(np.pad(np.ones_like(im2[1: -1, 1: -1, 0]), 1))
    warp_w1 = cv2.warpPerspective(w1, M, out_size)
    warp_w1 = warp_w1.reshape(warp_w1.shape[0], -1, 1)
    warp_w2 = cv2.warpPerspective(w2, M @ H2to1, out_size)
    warp_w2 = warp_w2.reshape(warp_w2.shape[0], -1, 1)

    img_pano = np.uint8((warp_im1 * warp_w1 + warp_im2 * warp_w2) / (warp_w1 +

    return img_pano

im1 = cv2.cvtColor(cv2.imread('data/incline_L.png'), cv2.COLOR_BGR2RGB)
im2 = cv2.cvtColor(cv2.imread('data/incline_R.png'), cv2.COLOR_BGR2RGB)

locs1, desc1 = briefLite(im1)
locs2, desc2 = briefLite(im2)
matches = briefMatch(desc1, desc2)

H2to1, inliers = computeH_ransac(matches, locs1, locs2)
print('H2to1: {0}'.format(H2to1 / H2to1[2, 2]))

img_pano = imageStitching_noClip(im1, im2, H2to1)

plt.rcParams['figure.figsize'] = (10, 10)
plt.imshow(img_pano)
plt.show()
```

Image pair used are **incline_L.png** and **incline_R.png**

```
H2to1: [[ 6.63694789e-01 -1.40226951e-02  3.60770282e+02]
 [-8.05621547e-02  9.02563001e-01 -2.17191196e+01]
 [-3.57863361e-04  2.65241068e-05  1.00000000e+00]]
```

```
C:\Users\CH\AppData\Local\Temp\ipykernel_9540\1080007857.py:42: RuntimeWarning: invalid value encountered in true_divide
  img_pano = np.uint8((warp_im1 * warp_w1 + warp_im2 * warp_w2) / (warp_w1 + warp_w2))
```

Image pair used are **incline_L.png** and **incline_R.png**

4.3 Generate Panorama (2 pts)

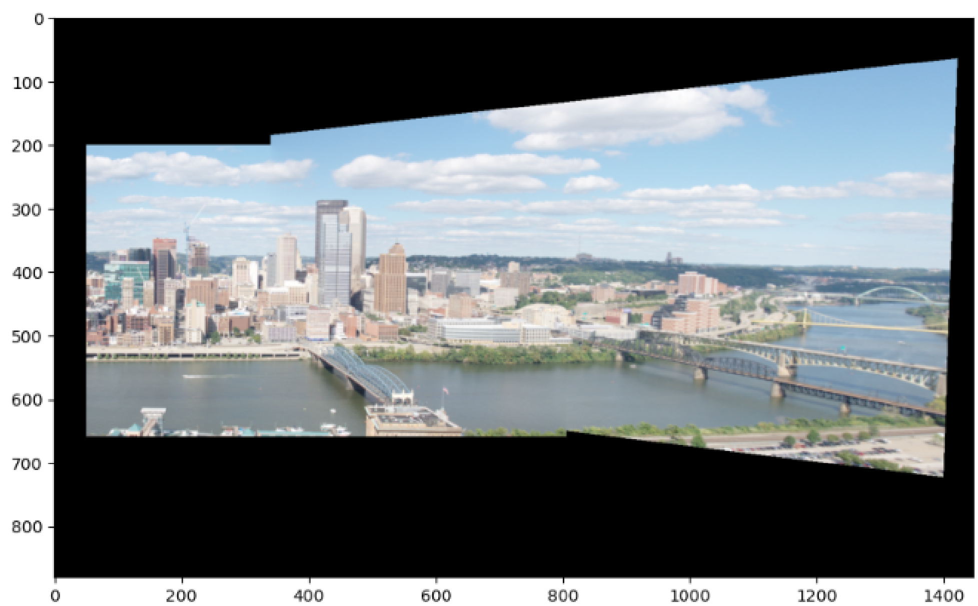
Save the resulting panorama on the full sized images and include the figure and computed homography matrix in your writeup.

In [ ]:

```
def generatePanorama(im1, im2):
    im1 = cv2.cvtColor(im1, cv2.COLOR_BGR2RGB)
    im2 = cv2.cvtColor(im2, cv2.COLOR_BGR2RGB)

    locs1, desc1 = briefLite(im1)
    locs2, desc2 = briefLite(im2)
    matches = briefMatch(desc1, desc2)
    H2to1, inliers = computeH_ransac(matches, locs1, locs2)
    print('H2to1: {0}'.format(H2to1 / H2to1[2, 2]))
    img_pano = imageStitching_noClip(im1, im2, H2to1)
    return img_pano

im1 = cv2.imread('data/incline_L.png')
im2 = cv2.imread('data/incline_R.png')

img_pano = generatePanorama(im1, im2)

plt.rcParams['figure.figsize'] = (10, 10)
plt.imshow(img_pano)
plt.show()
```

```
H2to1: [[ 6.53688849e-01 -3.17736591e-02  3.63486297e+02]
 [-8.11163187e-02  8.82552372e-01 -1.80974740e+01]
 [-3.59996314e-04 -1.49839079e-06  1.00000000e+00]]
```

```
C:\Users\CH\AppData\Local\Temp\ipykernel_9540\1080007857.py:42: RuntimeWarning: invalid value encountered in true_divide
  img_pano = np.uint8((warp_im1 * warp_w1 + warp_im2 * warp_w2) / (warp_w1 + warp_w2))
```

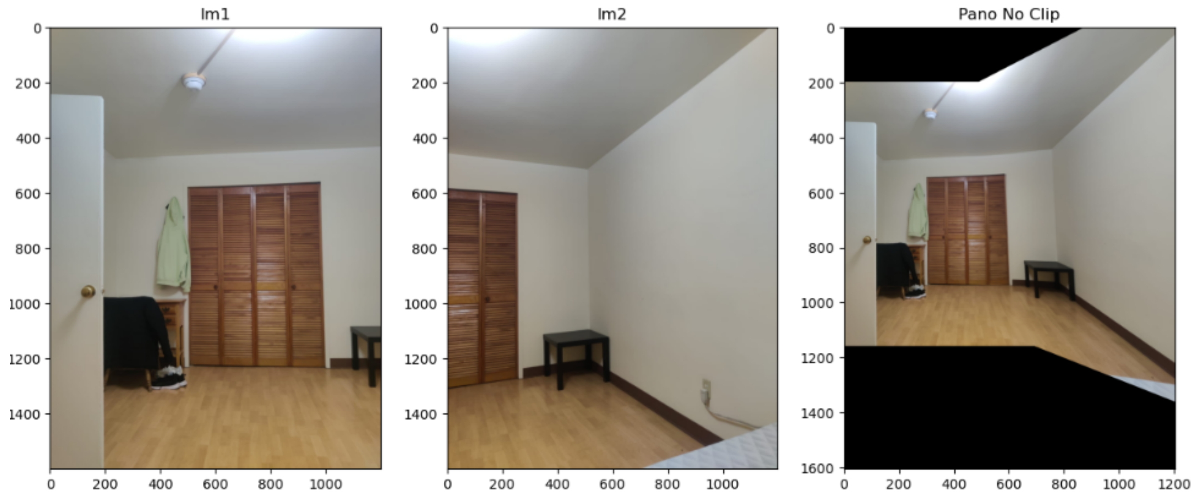


4.4 extra credits (3 pts)

Collect a pair of your own images (with your phone) and stitch them together using your code from the previous section. Include the pair of images and their result in the write-up.

In [ ]:

```
np.random.seed(16720)

im1 = cv2.imread('data/L.jpeg')
im2 = cv2.imread('data/R.jpeg')

im1 = cv2.cvtColor(im1, cv2.COLOR_BGR2RGB)
im2 = cv2.cvtColor(im2, cv2.COLOR_BGR2RGB)

locs1, desc1 = briefLite(im1)
locs2, desc2 = briefLite(im2)
matches = briefMatch(desc1, desc2)

H2to1, inliers = computeH_ransac(matches, locs1, locs2)
print('H2to1: {0}'.format(H2to1 / H2to1[2, 2]))

M = np.array([[0.6, 0, 0],
              [0, 0.6, 200],
              [0, 0, 1.]])

img_pano = imageStitching_noClip(im1, im2, H2to1, pad=5, isPadWidth=False, M=M)

fig, axes = plt.subplots(1, 3, figsize=(15, 15))

axes[0].imshow(im1)
axes[0].set_title('Im1')
axes[1].imshow(im2)
axes[1].set_title('Im2')
axes[2].imshow(img_pano)
axes[2].set_title('Pano No Clip')
plt.show()
```



4.5 extra credits (2 pts)

Collect at least 6 images and stitch them into a single noClip image. You can either collect your own, or use the [PNC Park images (http://www.cs.jhu.edu/~misha/Code/SMG/PNC3.zip)](http://www.cs.jhu.edu/~misha/Code/SMG/PNC3.zip) from Matt Uyttendaele. We used the PNC park images (subsmapled to 1/4 sized) and ORB keypoints and descriptors for our reference solution.

YOUR ANSWER HERE...

Question 5: Poisson Image Stitching (15 points)

Write a function called $\texttt{poisson-blend(background,foreground,mask)}$ which takes 3 equal sized images (background and foreground as RGB, mask as binary) and solves the Poisson equation, using gradients from foreground and boundary conditions from the background.

The problem will be manually graded. Please include results from both the $(\texttt{fg1,bg1,mask1})$ and $(\texttt{fg2,bg2,mask2})$ images in your write-up.

In [1]:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from scipy.sparse import linalg as linalg
from scipy.sparse import lil_matrix as lil_matrix


def image_edge(index, mask):
    if (mask[index] == 1) == False:
        return False
    for pt in surr_(index):
        if (mask[pt] == 1) == False: return True
    return False

def surr_(index):
    i,j = index
    return [(i+1,j),(i-1,j),(i,j+1),(i,j-1)]

def poisson_sparse_matrix(points):
    A = lil_matrix((len(points),len(points)))
    for i,index in enumerate(points):
        A[i,i] = 4
        for x in surr_(index):
            if x not in points: continue
            j = points.index(x)
            A[i,j] = -1
    return A

def poisson_blend(background, foreground, mask):
    nonzero = np.nonzero(mask)
    indicies = list(zip(nonzero[0], nonzero[1]))
    A = poisson_sparse_matrix(indicies)
    b = np.zeros(len(indicies))
    for i,index in enumerate(indicies):
        j,k = index
        b[i] = (4 * background[j,k]) - (1 * background[j+1, k]) - (1 * backgrou
        - (1 * background[j, k+1]) - (1 * background[j, k-1])
        if (image_edge(index,mask) == True) == 1:
            for pt in surr_(index):
                if (mask[pt] == 1) == False:
                    b[i] += foreground[pt]

    x = linalg.cg(A, b)
    composite = np.copy(foreground).astype(int)
    for i,index in enumerate(indicies):
        composite[index] = x[0][i]
    return composite
```

```
background_img = cv2.cvtColor(cv2.imread('data/fg1.png'), cv2.COLOR_BGR2RGB)
foreground_img = cv2.cvtColor(cv2.imread('data/bg1.png'), cv2.COLOR_BGR2RGB)
mask_img = cv2.cvtColor(cv2.imread('data/mask1.png'), cv2.COLOR_BGR2GRAY)

mask = np.atleast_3d(mask_img).astype(np.float) / 255.
mask[mask != 1] = 0
mask = mask[:,:,0]
channels = background_img.shape[-1]

stack_result = [poisson_blend(background_img[:,:,i], foreground_img[:,:,i], mas
result = cv2.merge(stack_result)
plt.rcParams['figure.figsize'] = (10, 10)
plt.imshow(result)
plt.show()
```

```
C:\Users\CH\AppData\Local\Temp\ipykernel_14660\3861378070.py:53:
DeprecationWarning: `np.float` is a deprecated alias for the bui
ltin `float`. To silence this warning, use `float` by itself. Do
ing this will not modify any behavior and is safe. If you specif
ically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: http
s://numpy.org/devdocs/release/1.20.0-notes.html#deprecations (ht
tps://numpy.org/devdocs/release/1.20.0-notes.html#deprecations)
  mask = np.atleast_3d(mask_img).astype(np.float) / 255.
Clipping input data to the valid range for imshow with RGB data
 ([0..1] for floats or [0..255] for integers).
```

In [ ]:

```

```