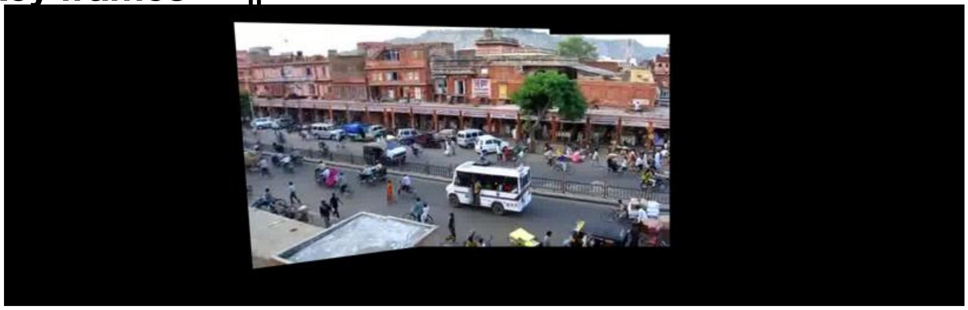
Programming Project #5: Video Stitching and Processing

CS445: Computational Photography - Fall 2019

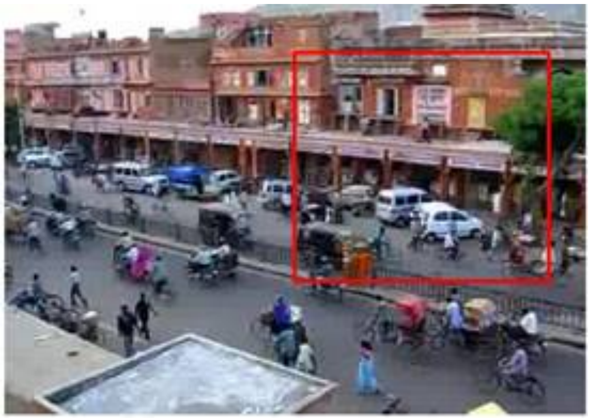
Part I: Stitch two key frames ¶



This involves:

1. compute homography H between two frames;
2. project each frame onto the same surface;
3. blend the surfaces.

Check that your homography is correct by plotting four points that form a square in frame 270 and their projections in each image, like this:



In [166]:
```
import cv2
import numpy as np
import utils
from numpy.linalg import svd, inv

%matplotlib inline
from matplotlib import pyplot as plt
import matplotlib.patches as patches
```

In [255]:
```
# images location
im1 = './images/input/frames2/f0360.jpg'
im2 = './images/input/frames2/f0499.jpg'

# Load an color image in grayscale
im1 = cv2.imread(im1)
#im1 = cv2.cvtColor(im1, cv2.COLOR_BGR2RGB)
im2 = cv2.imread(im2)
#im2 = cv2.cvtColor(im2, cv2.COLOR_BGR2RGB)
```

In [250]:
```
def auto_homography(Ia,Ib, homography_func=None, normalization_func=None)
    '''
    Computes a homography that maps points from Ia to Ib

    Input: Ia and Ib are images
    Output: H is the homography

    '''
    if Ia.dtype == 'float32' and Ib.dtype == 'float32':
        Ia = (Ia*255).astype(np.uint8)
        Ib = (Ib*255).astype(np.uint8)

    Ia_gray = cv2.cvtColor(Ia,cv2.COLOR_BGR2GRAY)
    Ib_gray = cv2.cvtColor(Ib,cv2.COLOR_BGR2GRAY)

    # Initiate SIFT detector
    sift = cv2.xfeatures2d.SIFT_create()
```

```
# find the keypoints and descriptors with SIFT
kp_a, des_a = sift.detectAndCompute(Ia_gray,None)
kp_b, des_b = sift.detectAndCompute(Ib_gray,None)

# BFMatcher with default params
bf = cv2.BFMatcher()
matches = bf.knnMatch(des_a,des_b, k=2)

# Apply ratio test
good = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        good.append(m)

numMatches = int(len(good))

matches = good

# Xa and Xb are 3xN matrices that contain homogeneous coordinates for
# matching points for each image
Xa = np.ones((3,numMatches))
Xb = np.ones((3,numMatches))

for idx, match_i in enumerate(matches):
    Xa[:,idx][0:2] = kp_a[match_i.queryIdx].pt
    Xb[:,idx][0:2] = kp_b[match_i.trainIdx].pt

## RANSAC
niter = 1000
best_score = 0

for t in range(niter):
    # estimate homography
    subset = np.random.choice(numMatches, 4, replace=False)
    pts1 = Xa[:,subset]
    pts2 = Xb[:,subset]

    H_t = homography_func(pts1, pts2) # edit helper code below (compu

    # score homography
    Xb_ = np.dot(H_t, Xa) # project points from first image to second
    du = Xb_[0,:]/Xb_[2,:] - Xb[0,:]/Xb[2,:]
    dv = Xb_[1,:]/Xb_[2,:] - Xb[1,:]/Xb[2,:]

    ok_t = np.sqrt(du**2 + dv**2) < 1  # you may need to play with th
    score_t = sum(ok_t)

    if score_t > best_score:
        best_score = score_t
        H = H_t
```

```
            in_idx = ok_t

    #print('best score: {:02f}'.format(best_score))

    # Optionally, you may want to re-estimate H based on inliers

    return H
```

In [251]:

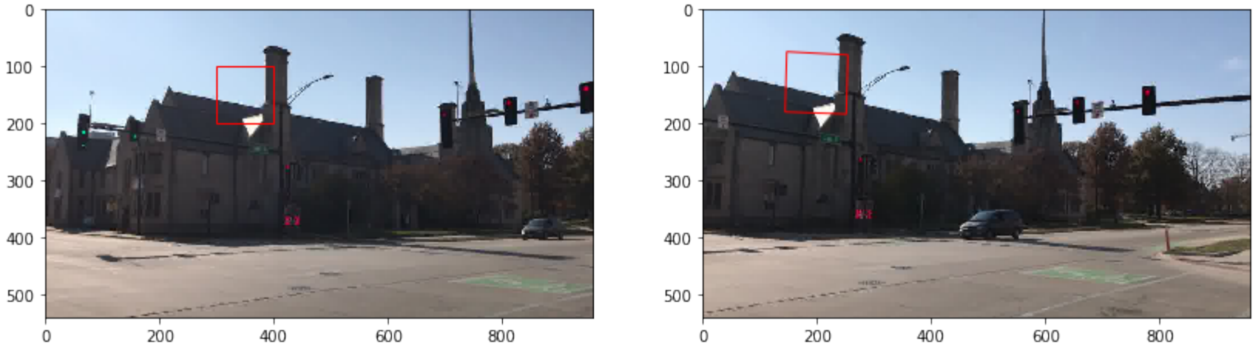
```
def computeHomography(pts1, pts2):
    '''
    Compute homography that maps from pts1 to pts2 using least squares so

    Input: pts1 and pts2 are 3xN matrices for N points in homogeneous
    coordinates.

    Output: H is a 3x3 matrix, such that pts2~=H*pts1
    '''
    (_, n) = pts1.shape
    A = np.zeros((2*n, 9))

    for i in range(n):
        A[2*i, :3] = -pts1[:, i]
        A[2*i, 6:9] = pts1[:, i]*(pts2[0, i]/pts2[2,i])
        A[2*i+1, 3:6] = -pts1[:, i]
        A[2*i+1, 6:9] = pts1[:, i]*(pts2[1, i]/pts2[2,i])


    u, s, vh = np.linalg.svd(A)
    H = vh.T[:, -1]
    H /= H[-1]


    return H.reshape((3,3))
```

In [252]:

```
H = auto_homography(im1,im2, computeHomography)
```

In [253]:

```
fig, ax = plt.subplots(nrows=1,ncols=2)
fig.set_size_inches(14,6)

pts = np.array([[300,100,1], [400, 100, 1], [400, 200, 1], [300, 200, 1]]
pts2 = H@pts
for i in range(4):
    pts2[:,i] /= pts2[2,i]

ax[0].imshow(im1)
ax[1].imshow(im2)
poly1 = patches.Polygon(pts[:-1,:].T, fill=False, edgecolor='r')
ax[0].add_patch(poly1)
poly2 = patches.Polygon(pts2[:-1,:].T, fill=False, edgecolor='r')
ax[1].add_patch(poly2)

plt.savefig('match_square.jpg')
```

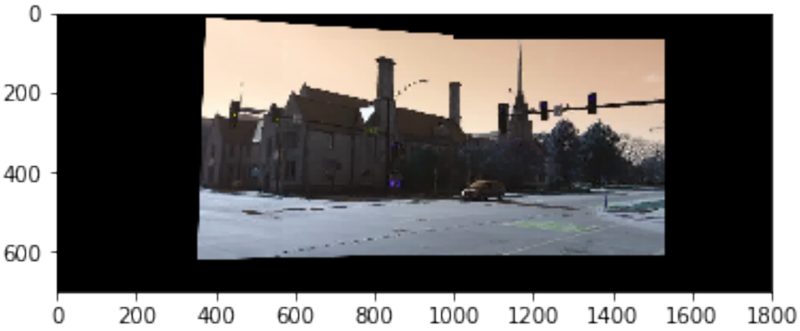


In [256]:

```
H_t = np.array([[1, 0, 570], [0, 1, 70], [0, 0, 1]])
img_warped = cv2.warpPerspective(im1, H_t.dot(H), (1000, 700))
plt.imshow(img_warped)

blend = np.zeros((700, 1800, 3))
blend[:700, :1000, :] = img_warped

blend[70:70+540,570:570+960,:] = im2
blend = blend.astype(np.uint8)
plt.imshow(blend)

cv2.imwrite('match.jpg', blend)
```

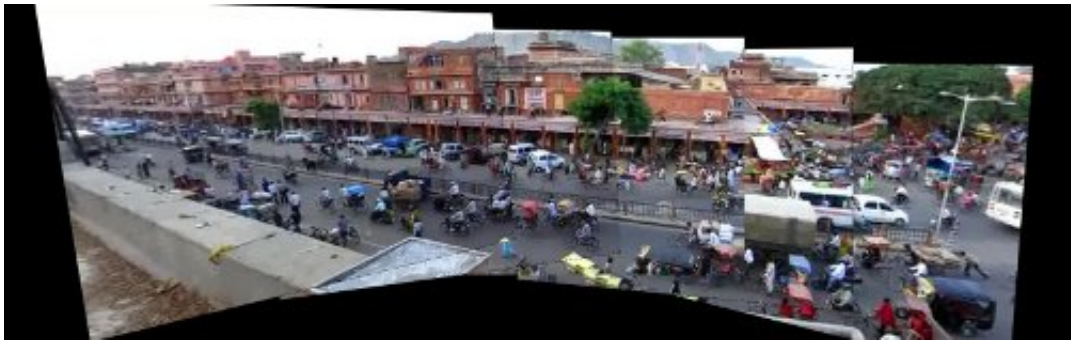
Out[256]:

```
True
```



Part II: Panorama using five key frames

In this part you will produce a panorama using five key frames. Let's determine frames [90, 270, 450, 630, 810] as key frames. The goal is to map all the five frames onto the plane corresponding to frame 450 (that we also call the *reference frame*). For the frames 270 and 630 you can follow the instructions in part 1.



Mapping frame 90 to frame 450 is difficult because they share very little area. Therefore you need to perform a two stage mapping by using frame 270 as a guide. Compute one projection from 90 to 270 and one from 270 to 450 and multiply the two matrices. This produces a projection from 90 to 450 even though these frames have very little area in common

In [125]:

```
import cv2
import numpy as np
```

In [126]:

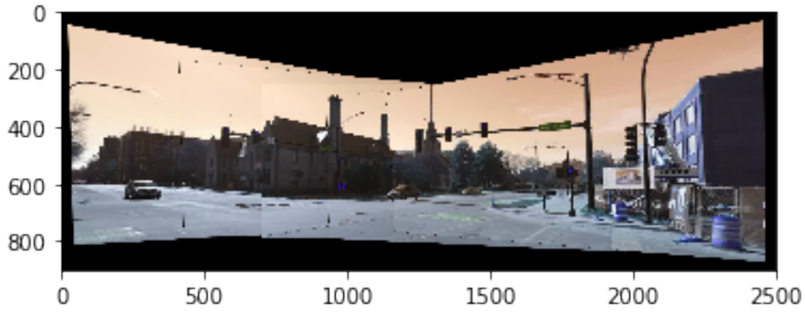
```
def naive_blend(im1, im2):
    mask = im2 != 0
    im1 = im1*(1-mask)
    return im1+im2
```

In [266]:

```
master_frames =[100, 300, 500, 700, 900]
#reference_frame = 205
#reference_idx = master_frames.index(reference_frame)
#im1 = './images/input/frames1/f0270.jpg'
frames = []

frames.append(cv2.imread('./images/input/frames2/f0100.jpg'))
for i in range(1, len(master_frames)):
    frames.append(cv2.imread('./images/input/frames2/f0'+str(master_frame

H12 = auto_homography(frames[0],frames[1], computeHomography)
H23 = auto_homography(frames[1],frames[2], computeHomography)
H34 = auto_homography(frames[3],frames[2], computeHomography)
H45 = auto_homography(frames[4],frames[3], computeHomography)
```

In [269]:
```
H_t = np.array([[1, 0, 700], [0, 1, 250], [0, 0, 1]])

img_warped = cv2.warpPerspective(frames[0], H_t.dot(H23@H12), (2500, 900)
img_warped1 = cv2.warpPerspective(frames[1], H_t.dot(H23), (2500, 900))
img_warped2 = cv2.warpPerspective(frames[3], H_t.dot(H34), (2500, 900))
img_warped3 = cv2.warpPerspective(frames[4], H_t.dot(H45@H34), (2500, 900

blended = naive_blend(img_warped, img_warped1)

blended[250:250+540, 700:700+960, :] = frames[2]
blended = naive_blend(blended, img_warped2)
blended = naive_blend(blended, img_warped3)
blended = blended.astype(np.uint8)

plt.imshow(blended)
cv2.imwrite('panorama_5.jpg', blended)
```

Out[269]:
```
True
```



Part 3: Map the video to the reference plane

In [270]:
```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from math import floor

import utils
```

In [271]:
```
dir_frames = 'images/input/frames2'
filenames = []
filesinfo = os.scandir(dir_frames)
```

In [272]:
```
filenames = [f.path for f in filesinfo if f.name.endswith(".jpg")]
filenames.sort(key=lambda f: int(''.join(filter(str.isdigit, f))))
```

In [273]:

```
frameCount = len(filenames)
frameHeight, frameWidth, frameChannels = cv2.imread(filenames[0]).shape
frames = np.zeros((frameCount, frameHeight, frameWidth, frameChannels),dt
```

In [274]:

```
for idx, file_i in enumerate(filenames):
    frames[idx] = cv2.imread(file_i) / 255.0
    #frames[idx] = cv2.cvtColor(cv2.imread(file_i), cv2.COLOR_BGR2RGB) /
```

In [275]:

```
pastHomographies = np.zeros((len(filenames),len(filenames), 3, 3),dtype=n
originTranslations = np.zeros((len(filenames), 2), dtype=np.float32)
```

In [276]:

```
for i in range(1000):
    projectedReference, pastHomographies, originTranslations = utils.proj
                                                                pastHomogr
                                                                auto_H_fun
                                                                yrange=100
                                                                numKeyfram
    cv2.imwrite('aligned_frames/a{:04d}.jpg'.format(i+1), projectedRefere
```

```
/Users/kathie/anaconda3/lib/python3.5/site-packages/ipykernel_launcher
.py:60: RuntimeWarning: divide by zero encountered in true_divide
/Users/kathie/anaconda3/lib/python3.5/site-packages/ipykernel_launcher
.py:61: RuntimeWarning: divide by zero encountered in true_divide
```

In [277]:

```
utils.imageFolder2mpeg('aligned_frames', fps=30)
```

In [219]:

```
H = auto_homography(frames[145],frames[205], computeHomography)
projectedReference = cv2.warpPerspective(frames[145], H_t.dot(H), (2500,
pastHomographies[145] = H
cv2.imwrite('aligned_frames/a{:04d}.jpg'.format(145+1), projectedReferenc
```

Out[219]:

```
True
```

In [18]:

```
import importlib
importlib.reload(utils)
```

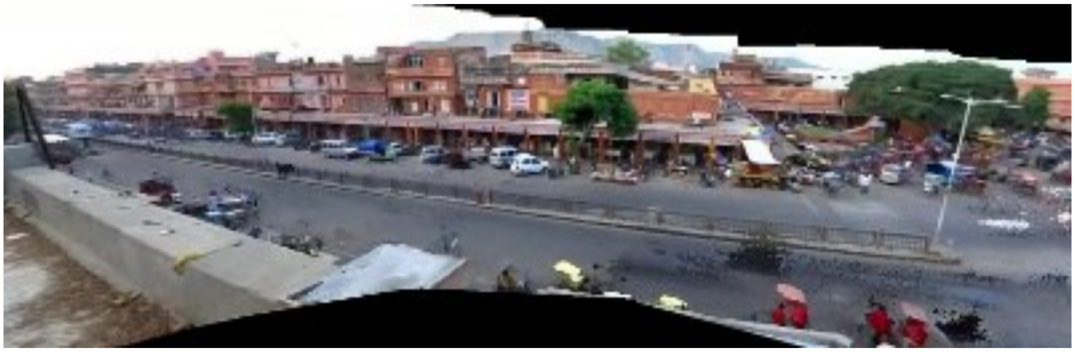
Out[18]:

```
<module 'utils' from '/Users/kathie/Dropbox/cs445/proj5/project 5 - st
arter package/utils.py'>
```

Part 4: Create background panorama

In this part you will remove moving objects from the video and create a background panorama that should incorporate pixels from all the frames.

In the video you produced in **part 3** each pixel appears in several frames. You need to estimate which of the many colors correspond to the background. We take advantage of the fact that the background color is fixed while the foreground color changes frequently (because foreground moves).



For each pixel in the sequence of **part 3**, determine all valid colors (colors that come from all frames that overlap that pixel). You can experiment with different methods for determining the background color of each pixel, as discussed in class. Perform the same procedure for all pixels and generate output. The output should be a completed panorama showing only pixels of background or non-moving objects.

In [290]:
```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [293]:
```
dir_frames = 'aligned_frames'
filenames = []
filesinfo = os.scandir(dir_frames)
```

In [294]:
```
filenames = [f.path for f in filesinfo if f.name.endswith(".jpg")]
filenames.sort(key=lambda f: int(''.join(filter(str.isdigit, f))))
```

In [295]:
```
frameCount = len(filenames)
projected_frames = np.zeros((frameCount, 1000, 3000, frameChannels),dtype
```

In [297]:
```
for idx, file_i in enumerate(filenames):
    projected_frames[idx] = cv2.imread(file_i) / 255.0
    #projected_frames[idx] = cv2.cvtColor(cv2.imread(file_i), cv2.COLOR_E
```

In [298]:
```
background = np.zeros((1000, 3000, 3))
```

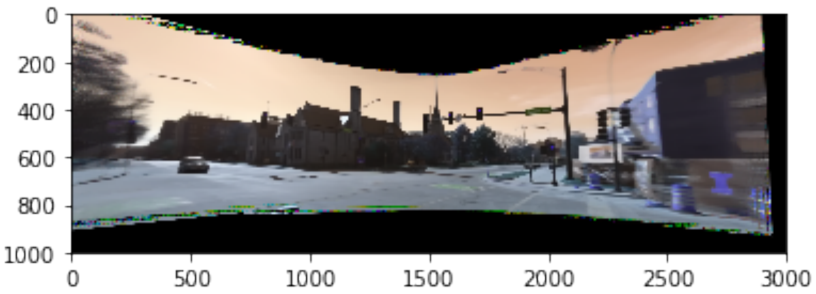
In [299]:
```
for i in range(1000):
    for j in range(3000):
        for c in range(3):
            nonzero = np.nonzero(projected_frames[:, i, j, c])
            if len(nonzero) != 0:
                #print(np.median(projected_frames[nonzero, i, j, c]))
                background[i][j][c] = np.median(projected_frames[nonzero,
```

```
/Users/kathie/anaconda3/lib/python3.5/site-packages/numpy/core/fromnum
eric.py:2920: RuntimeWarning: Mean of empty slice.
  out=out, **kwargs)
/Users/kathie/anaconda3/lib/python3.5/site-packages/numpy/core/_method
s.py:85: RuntimeWarning: invalid value encountered in true_divide
  ret = ret.dtype.type(ret / rcount)
```

In [300]:
```
plt.imshow(background)
```

Out[300]:
```
<matplotlib.image.AxesImage at 0x1255f2470>
```

```
/Users/kathie/anaconda3/lib/python3.5/site-packages/numpy/core/_method
s.py:32: RuntimeWarning: invalid value encountered in reduce
  return umr_minimum(a, axis, None, out, keepdims, initial)
```



In [301]:
```
cv2.imwrite('background_panorama.jpg', background*255)
```

Out[301]:
```
True
```

Part 5: Create background movie

Map the background panorama to the movie coordinates. For each frame of the movie, say frame 1, you need to estimate a projection from the panorama to frame 1. Note, you should be able to re-use the homographies that you estimated in **Part 3**. Perform this for all frames and generate a movie that looks like the input movie but shows only background pixels. All moving objects that belong to the foreground must be removed.

In [302]:

```
import os
import cv2
import numpy as np
```

In [303]:

```
x_min, y_min = originTranslations[0]
t = [-x_min, -y_min]
H_t = np.array([[1, 0, t[0]], [0, 1, t[1]], [0, 0, 1]], dtype=np.float32)
```

In [304]:

```
for i in range(1000):
    projected_background = cv2.warpPerspective(background, np.linalg.pinv
    cv2.imwrite('background/a{:04d}.jpg'.format(i+1), projected_backgroun
```

In [305]:

```
utils.imageFolder2mpeg('background', fps=30)
```

Part 6: Create foreground movie

In the background video, moving objects are removed. In each frame, those pixels that are different enough than the background color are considered foreground. For each frame determine foreground pixels and generate a movie that only includes foreground pixels.

In [306]:

```
import os
import cv2
import numpy as np
```

```
In [311]: for i in range(1000):
              projected_background = cv2.warpPerspective(background, np.linalg.pinv
              projected_image = cv2.warpPerspective(projected_frames[i], np.linalg.
              foreground = np.zeros((540, 960, 3))
              diff  = (projected_image - projected_background)
              diff = np.sqrt(diff[:,:,0]**2 + diff[:,:,1]**2 + diff[:,:,2]**2)#dist
              indexes = np.argwhere(diff > 0.2)

              mask = np.zeros((540, 960,3))
              mask[indexes[:,0], indexes[:,1], :] = 1
              mask = cv2.GaussianBlur(mask, (3, 3), cv2.BORDER_DEFAULT)

              foreground += projected_image*mask
              cv2.imwrite('foreground/a{:04d}.jpg'.format(i+1), foreground*255)
```

```
/Users/kathie/anaconda3/lib/python3.5/site-packages/ipykernel_launcher
.py:7: RuntimeWarning: invalid value encountered in greater
  import sys
```

```
In [312]: utils.imageFolder2mpeg('foreground', fps=30)
```

Bells and whistles

background wide

```
In [246]: for i in range(260):
              projected_background = cv2.warpPerspective(background, np.linalg.pinv
              cv2.imwrite('background_wide/a{:04d}.jpg'.format(i+1), projected_back
```

```
In [247]: utils.imageFolder2mpeg('background_wide', fps=30)
```

```
In [ ]:
```