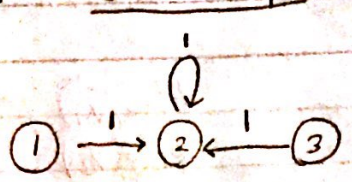
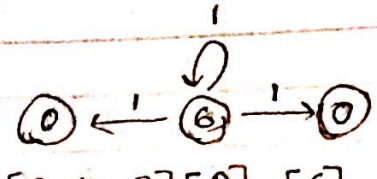
Homework #2 EE16A

(#1) a) $\begin{bmatrix} 0 & 0 & 0 \\ b & a & 0 \\ c & 0 & 0 \end{bmatrix}$ original $\quad \begin{bmatrix} 0 & b & c \\ 0 & a & 0 \\ 0 & 0 & 0 \end{bmatrix}$ reverse

No

Counterexample:





$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 6 \\ 0 \end{bmatrix} = \begin{bmatrix} 6 \\ 6 \\ 6 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \neq \begin{bmatrix} 6 \\ 6 \\ 6 \end{bmatrix}$$

b) There's no leak in this system. No one leaves or enters.

c) $$A = \begin{bmatrix} 0 & 0 & 0 \\ 0.5 & 0.4 & 0.2 \\ 0. & 0.6 & 0.65 \end{bmatrix}$$

Since column vectors don't all sum to 1, there is a leak in the system. Website 1 has $\emptyset$ people after the first time step because Row 1 is all zeroes. Site 3 will slowly lose all people because column 3 doesn't add to 1, meaning it has a net loss each step.

d) $$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & a_{ij} & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}\begin{bmatrix} x \\ \vdots \\ x \end{bmatrix} = \begin{bmatrix} x \\ \vdots \\ x \end{bmatrix}$$

row form

$$= \begin{bmatrix} a_1^T \\ \vdots \\ a_n^T \end{bmatrix}\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \longrightarrow b_1 = a_1^T \underline{x} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \cdots a_{1n} \end{bmatrix}\begin{bmatrix} x \\ x \\ \vdots \\ x \end{bmatrix}$$

$$= a_{11}x + a_{12}x + a_{13}x + a_{1n}x$$

$$= x\underbrace{(a_{11} + a_{12} + a_{13} \cdots a_{1n})}_{\text{must sum to 1}}$$

$$b_1 = x$$

This applies to each row, so $\underline{b} = \underline{x}$.

(#2) $\{\vec{v}_1, \vec{v}_2, \cdots, \vec{v}_k\}$ = set of k linearly dependent vectors in R^n

A = any $n \times n$ matrix

$\{a_1 \cdots a_k\}$ = scalars, atleast 1 is non-zero

Linear Dependence → $\alpha_1\vec{v}_1 + \alpha_2\vec{v}_2 + \cdots \alpha_k\vec{v}_k = \vec{0}$

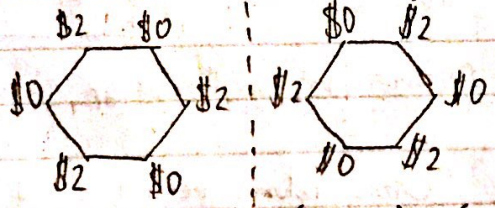
$\times A$

$A(\alpha_1\vec{v}_1) + A(\alpha_2\vec{v}_2) + \cdots + A(\alpha_k\vec{v}_k) = \vec{0}$

$\alpha_1(A\vec{v}_1) + \alpha_2(A\vec{v}_2) + \cdots + \alpha_k(A\vec{v}_k) = \vec{0}$

These $\{\alpha_1 \cdots \alpha_k\}$ scalars prove that $\{A\vec{v}_1 \cdots A\vec{v}_k\}$ are linearly dependent.

(#3) a) No you cannot. Both of these arrangements would reveal identical $\{P_1 \cdots P_6\}$:



$\{P_1 \cdots P_6\} = \{1 \cdots 1\}$ $\{P_1 \cdots P_6\} = \{1 \cdots 1\}$

b)

$$\begin{bmatrix} 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 \\ 1/2 & 0 & 0 & 0 & 1/2 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \end{bmatrix}$$

$R_5 = R_5 - R_1 + R_2 - R_3 + R_4$

$$\begin{bmatrix} 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 - P_1 - P_3 + P_2 + P_4 \end{bmatrix}$$

b) continued

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} = \begin{bmatrix} P_1 - P_2 + P_3 - P_4 + P_5 \\ P_2 - P_3 + P_4 - P_5 + P_1 \\ P_3 - P_4 + P_5 - P_1 + P_2 \\ P_4 - P_5 - P_2 + P_1 + P_3 \\ P_5 - P_1 - P_3 + P_2 + P_4 \end{bmatrix}$$

so... unique solution for tips. ✓

c) We can find tips for any odd n. For even n, there are always 2 combos of tips to get the same plates.

#4 a)

$$\begin{bmatrix} q_x \\ q_y \end{bmatrix} = \begin{bmatrix} R_{xx} & R_{xy} \\ R_{yx} & R_{yy} \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

$$\begin{bmatrix} q_x \\ q_y \end{bmatrix} = \begin{bmatrix} R_{xx}p_x + R_{xy}p_y \\ R_{yx}p_x + R_{yy}p_y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix} \Rightarrow$$

$$\begin{aligned} q_x &= R_{xx}p_x + R_{xy}p_y + T_x \\ q_y &= R_{yx}p_x + R_{yy}p_y + T_y \end{aligned}$$

Known values: q_x, q_y, p_x, p_y (4)

Unknown values: $R_{xx}, R_{xy}, R_{yx}, R_{yy}, T_x, T_y$ (6)

We'll need 6 equations & 3 pairs of common points

b) (q_1, p_1) (q_2, p_2) (q_2, p_3)

$$\begin{bmatrix} q_{1x} \\ q_{1y} \\ q_{2x} \\ q_{2y} \\ q_{3x} \\ q_{3y} \end{bmatrix} = \begin{bmatrix} p_{1x} & p_{1y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{1x} & p_{1y} & 0 & 1 \\ p_{2x} & p_{2y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{2x} & p_{2y} & 0 & 1 \\ p_{3x} & p_{3y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{3x} & p_{3y} & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{xx} \\ R_{xy} \\ R_{yx} \\ R_{yy} \\ T_x \\ T_y \end{bmatrix} = \begin{bmatrix} q_{1x} \\ q_{1y} \\ q_{2x} \\ q_{2y} \\ q_{3x} \\ q_{3y} \end{bmatrix}$$

c) $R = \begin{bmatrix} 1.195 & 0.105 \\ -0.109 & 1.176 \end{bmatrix}$ $T = \begin{bmatrix} -150.0 \\ -235.8 \end{bmatrix}$

d)

$$\begin{bmatrix} p_{1x} & p_{1y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{1x} & p_{1y} & 0 & 1 \\ p_{2x} & p_{2y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{2x} & p_{2y} & 0 & 1 \\ p_{3x} & p_{3y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{3x} & p_{3y} & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{xx} \\ R_{xy} \\ R_{yx} \\ R_{yy} \\ T_x \\ T_y \end{bmatrix} = \begin{bmatrix} q_{1x} \\ q_{1y} \\ q_{2x} \\ q_{2y} \\ q_{3x} \\ q_{3y} \end{bmatrix}$$

$R4 = R4 - R1 \quad R5 = R5 - R1$

$$\begin{bmatrix} p_{1x} & p_{1y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{1x} & p_{1y} & 0 & 1 \\ p_{2x} & p_{2y} & 0 & 0 & 1 & 0 \\ -p_{1x} & -p_{1y} & p_{2x} & p_{2y} & -1 & 1 \\ p_{3x}-p_{1x} & p_{3y}-p_{1y} & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{3x} & p_{3y} & 0 & 1 \end{bmatrix} \begin{bmatrix} \\ \\ \\ \\ \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \\ \\ \\ \end{bmatrix}$$

$R4 = R4 - R2 \quad R6 = R6 - R2$

$$\begin{bmatrix} p_{1x} & p_{1y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{1x} & p_{1y} & 0 & 1 \\ p_{2x}-p_{1x} & p_{2y}-p_{1y} & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{2x}-p_{1x} & p_{2y}-p_{1y} & 0 & 0 \\ p_{3x}-p_{1x} & p_{3y}-p_{1y} & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{3x}-p_{1x} & p_{3y}-p_{1y} & 0 & 0 \end{bmatrix} \begin{bmatrix} \\ \\ \\ \\ \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \\ \\ \\ \end{bmatrix}$$

By linearity, $p_{2x} - p_{1x} = m(p_{3x} - p_{1x})$

(m ↑ slope)

So 5 is just a scaled version of 3 and 6 is of 4.

$\Rightarrow$ Undetermined.

(#5) Dhruv Kathuria #3031790620
Divya Vijayan #3032175290

We collaborated on the tough parts of questions, such as part d of #5

Problem 4 Image Stitching

This section of the notebook continues the image stiching problem. Be sure to have a `figures` folder in the same directory as the notebook. The `figures` folder should contain the files:

```
Berkeley_banner_1.jpg
Berkeley_banner_2.jpg
stacked_pieces.jpg
lefthalfpic.jpg
righthalfpic.jpg
```

Note: This structure is present in the provided HW2 zip file.

Run the next block of code before proceeding

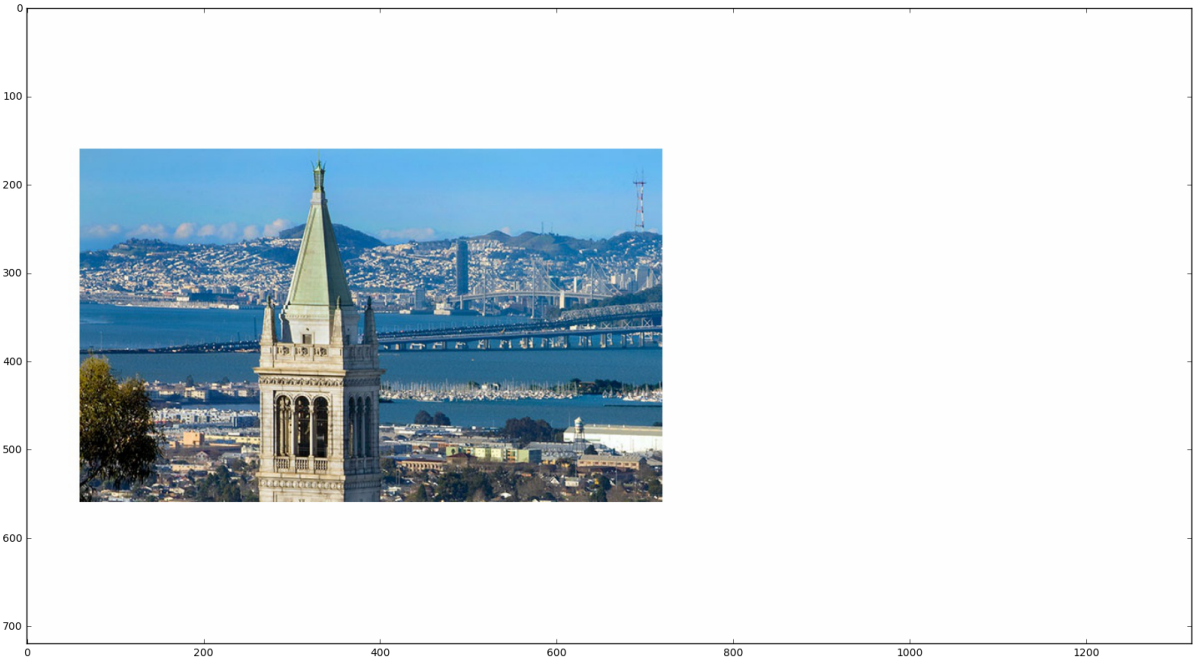
In [2]:
```
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from numpy import pi, cos, exp, sin
import matplotlib.image as mpimg
import matplotlib.transforms as mtransforms


%matplotlib inline

#loading images
image1=mpimg.imread('figures/Berkeley_banner_1.jpg')
image1=image1/255.0
image2=mpimg.imread('figures/Berkeley_banner_2.jpg')
image2=image2/255.0
image_stack=mpimg.imread('figures/stacked_pieces.jpg')
image_stack=image_stack/255.0


image1_marked=mpimg.imread('figures/lefthalfpic.jpg')
image1_marked=image1_marked/255.0
image2_marked=mpimg.imread('figures/righthalfpic.jpg')
image2_marked=image2_marked/255.0

def
euclidean_transform_2to1(transform_mat,translation,image,position,LL,UL):

    new_position=np.round(transform_mat.dot(position)+translation)
    new_position=new_position.astype(int)


    if (new_position>=LL).all() and (new_position<UL).all():
        values=image[new_position[0][0],new_position[1][0],:]
    else:
        values=np.array([2.0,2.0,2.0])

    return values

def
euclidean_transform_1to2(transform_mat,translation,image,position,LL,UL):

    transform_mat_inv=np.linalg.inv(transform_mat)
    new_position=np.round(transform_mat_inv.dot(position-translation))
    new_position=new_position.astype(int)

    if (new_position>=LL).all() and (new_position<UL).all():
        values=image[new_position[0][0],new_position[1][0],:]
    else:
        values=np.array([2.0,2.0,2.0])

    return values
```

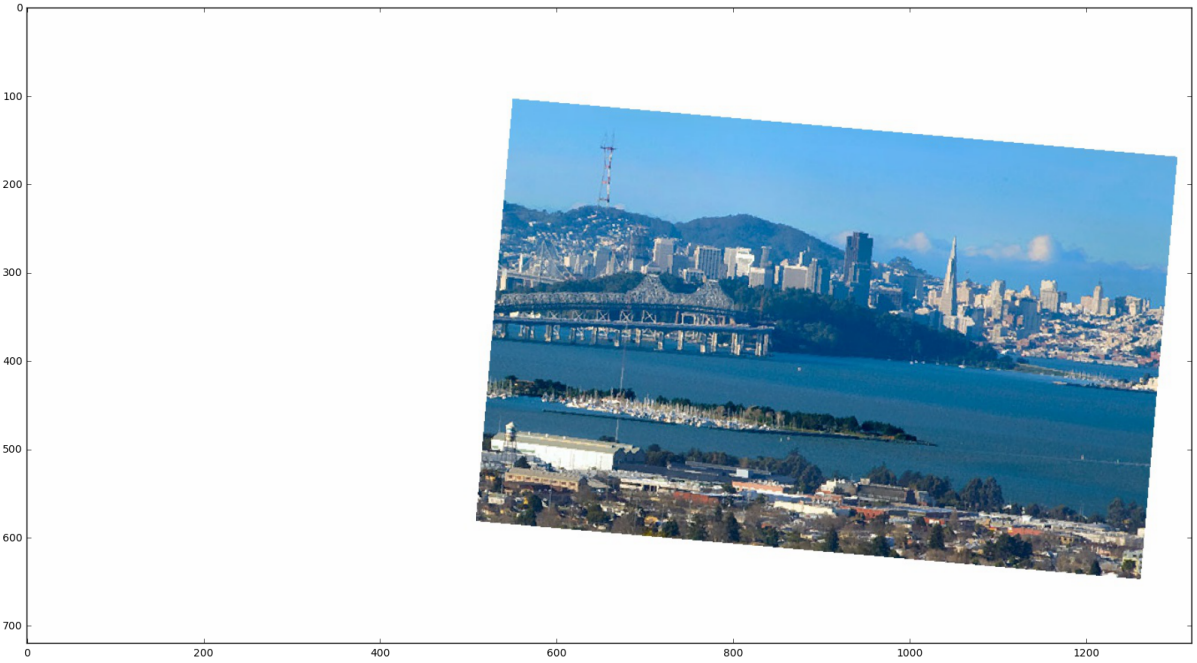
We will stick to a simple example and just consider stitching two images (if you can stitch two pictures, then you could conceivably stitch more by applying the same technique over and over again).

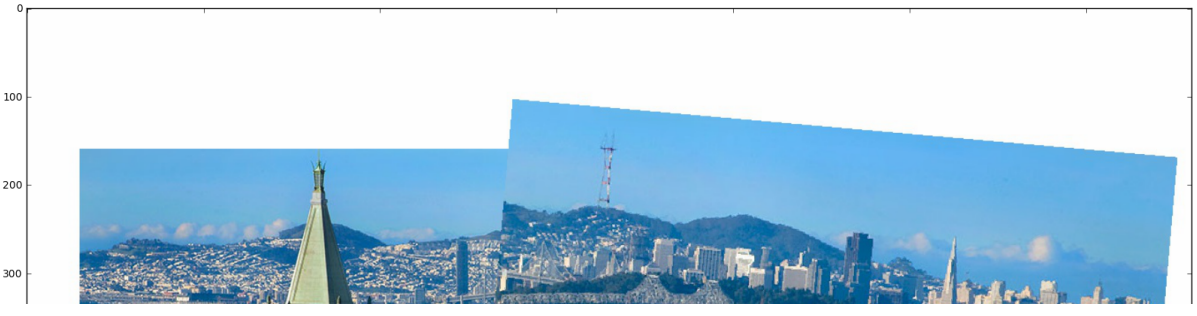
Professor Ayazifar decided to take an amazing picture of the Campanile overlooking the bay. Unfortunately, the field of view of his camera was not large enough to capture the entire scene, so he decided to take two pictues and stich them together.

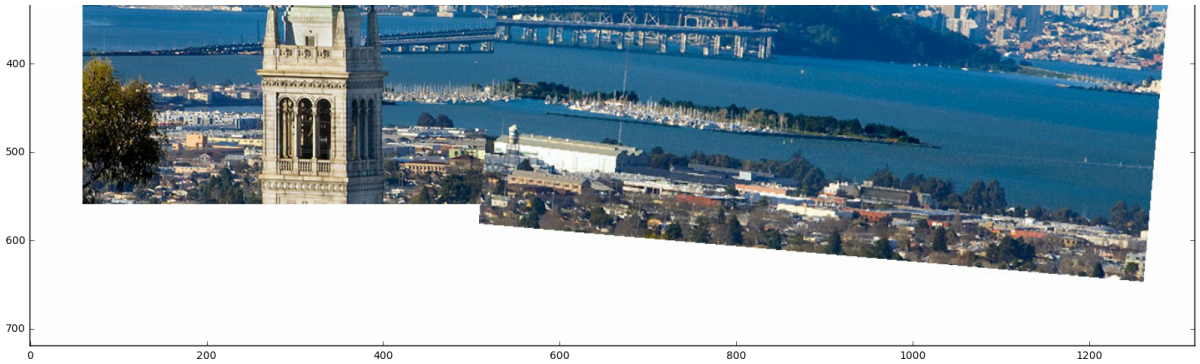
The next block will display the two images.

In [3]:
```
plt.figure(figsize=(20,40))

plt.subplot(311)
plt.imshow(image1)

plt.subplot(312)
plt.imshow(image2)

plt.subplot(313)
plt.imshow(image_stack)

plt.show()
```


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

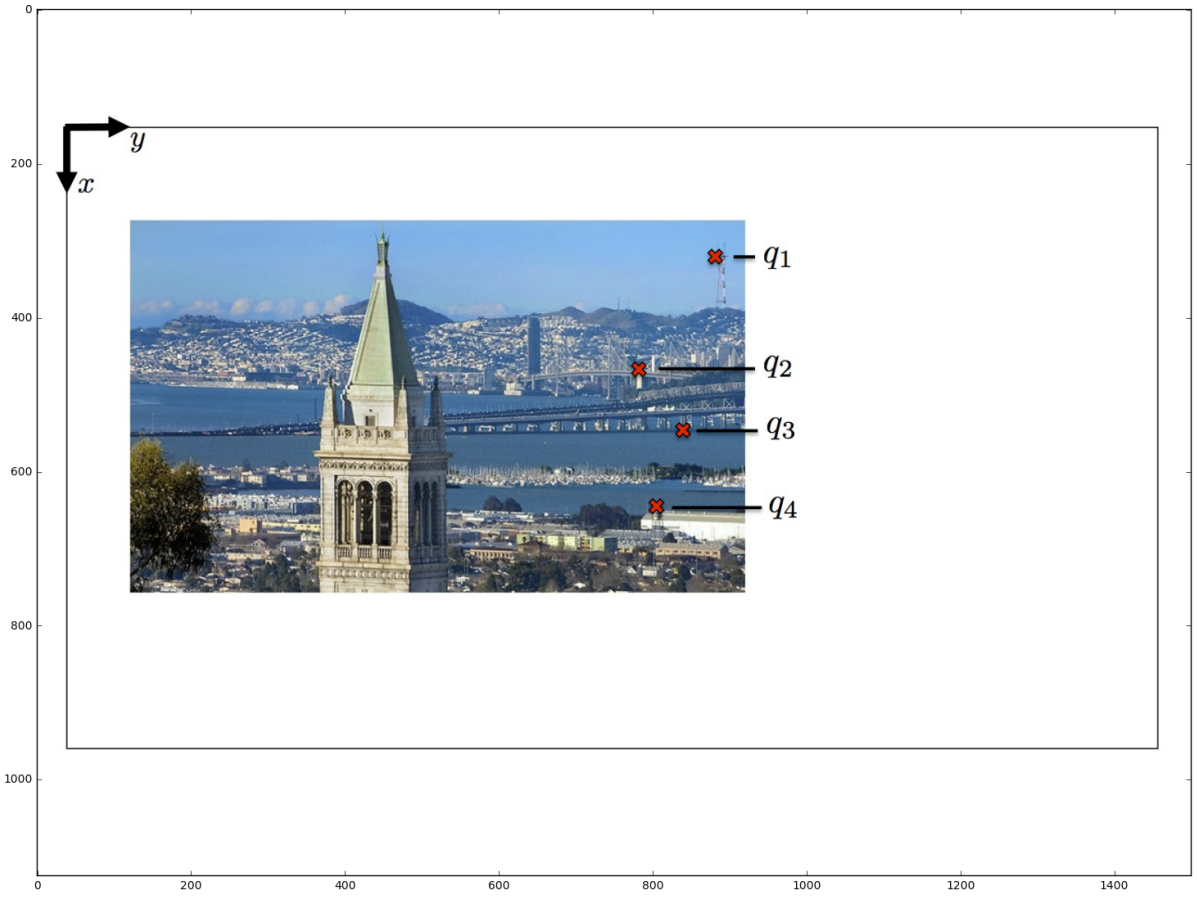

0
100
200
300

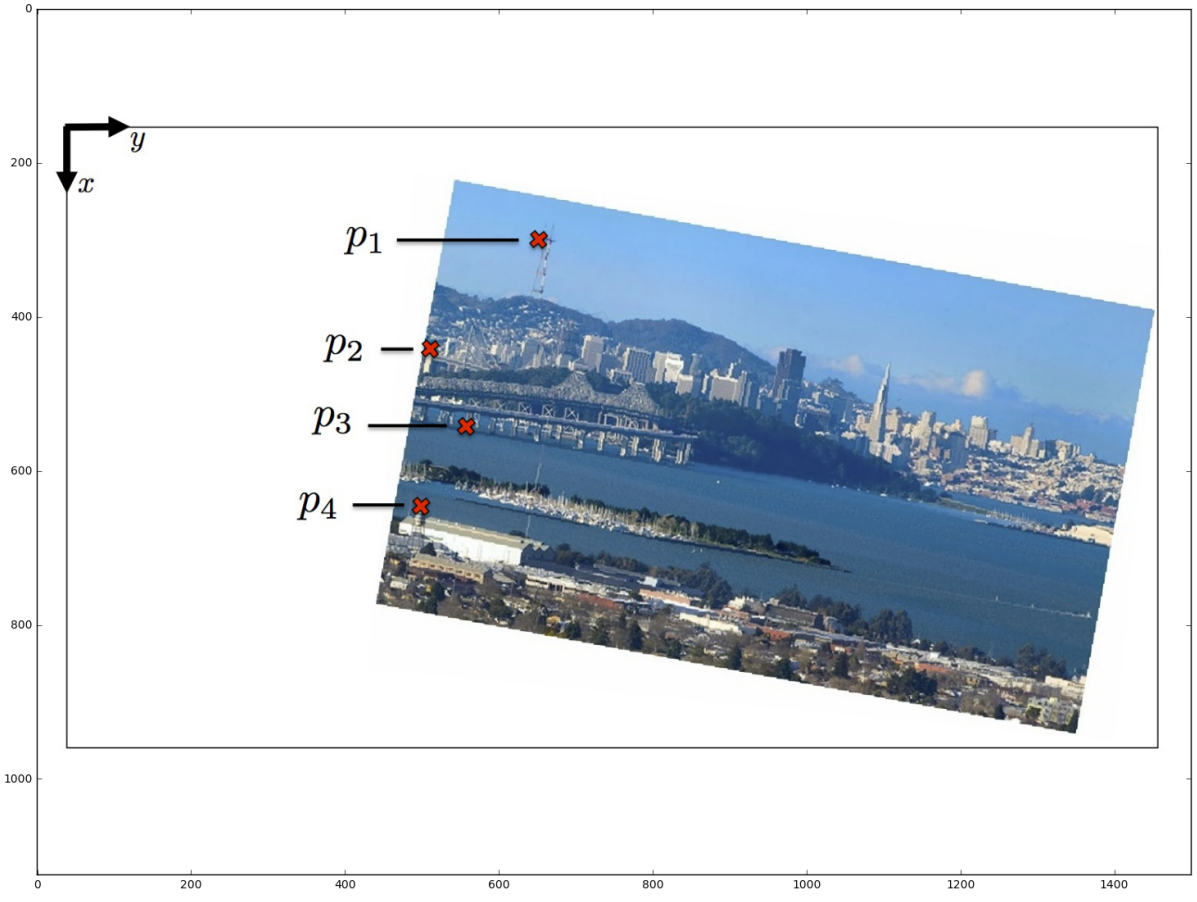


Once you apply Marcela's algorithm on the two images you get the following result (run the next block):

In [4]:
```
plt.figure(figsize=(20,30))

plt.subplot(211)
plt.imshow(image1_marked)

plt.subplot(212)
plt.imshow(image2_marked)
```

```
Out[4]: <matplotlib.image.AxesImage at 0x10bcc7c18>
```


y
x
q_1
q_2
q_3
q_4
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400


y
x
p_1
p_2
p_3
p_4
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400

As you can see Marcela's algorithm was able to find four common points between the two images. These points expressed in the coordinates of the first image and second image are

$$\vec{p_1} = \begin{bmatrix} 200 \\ 700 \end{bmatrix} \quad \vec{p_2} = \begin{bmatrix} 310 \\ 620 \end{bmatrix} \quad \vec{p_3} = \begin{bmatrix} 390 \\ 660 \end{bmatrix} \quad \vec{p_4} = \begin{bmatrix} 460 \\ 630 \end{bmatrix}$$

$$\vec{q_1} = \begin{bmatrix} 162.2976 \\ 565.8862 \end{bmatrix} \quad \vec{q_2} = \begin{bmatrix} 285.4283 \\ 458.7469 \end{bmatrix} \quad \vec{q_3} = \begin{bmatrix} 385.2465 \\ 498.1973 \end{bmatrix} \quad \vec{q_4} = \begin{bmatrix} 465.7892 \\ 455.0132 \end{bmatrix}$$

It should be noted that in relation to the image the positive x-axis is down and the positive y-axis is right. This will have no bearing as to how you solve the problem, however it helps in interpreting what the numbers mean relative to the image you are seeing.

Using the points determine the parameters $R_{11}, R_{12}, R_{21}, R_{22}, T_x, T_y$ that map the points from the first image to the points in the second image by solving an appropriate system of equations. Hint: you do not need all the points to recover the parameters.

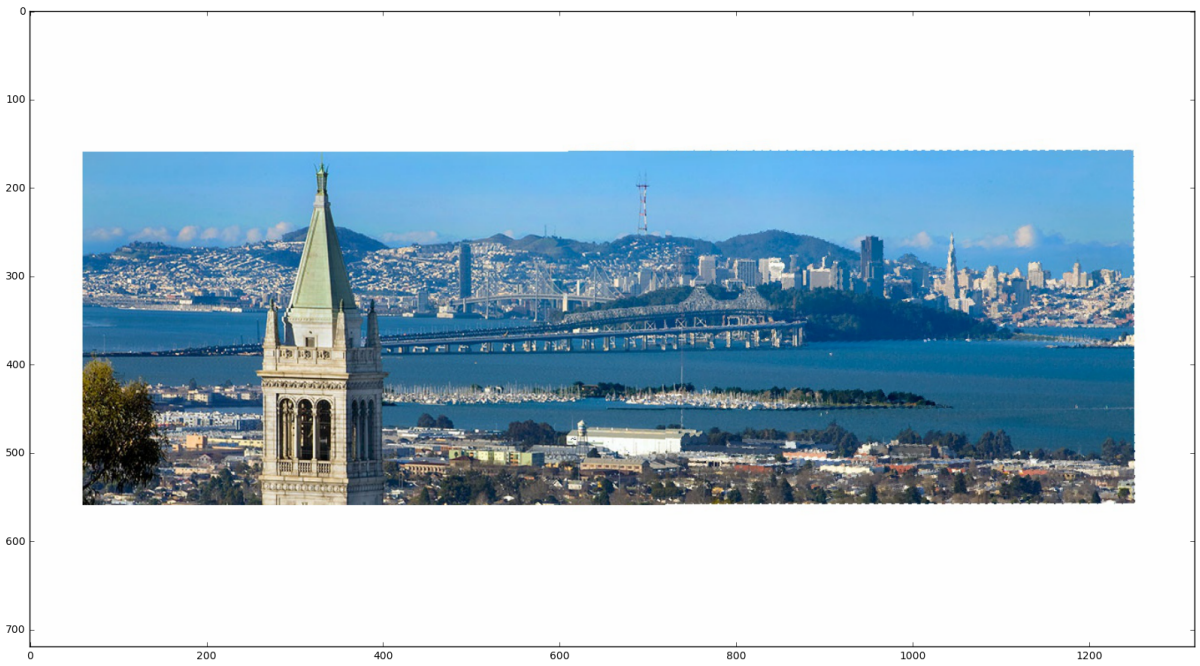
In [7]:
```
# Note that the following is a general template for solving for 6 unknow
ns from 6 equations represented as Az = b.
# You do not have to use the following code exactly.
# All you need to do is to find parameters R_11, R_12, R_21, R_22, T_x,
 T_y.
# If you prefer finding them another way it is fine.

# fill in the entries
A = np.array([[200,700,0,0,1,0],
              [0,0,200,700,0,1],
              [310,620,0,0,1,0],
              [0,0,320,620,0,1],
              [390,660,0,0,1,0],
              [0,0,390,660,0,1]])

# fill in the entries
b = np.array([[162.2976],[565.8862],[285.4283],[458.7469],[385.2465],[49
8.1973]])

A = A.astype(float)
b = b.astype(float)

# solve the linear system for the coefficiens
z = np.linalg.solve(A,b)

#Parameters for our transformation
R_11 = z[0,0]
R_12 = z[1,0]
R_21 = z[2,0]
R_22 = z[3,0]
T_x  = z[4,0]
T_y  = z[5,0]

z
```

Out[7]:
```
array([[  1.19543370e+00],
       [  1.04587593e-01],
       [ -1.08609615e-01],
       [  1.17632683e+00],
       [ -1.50000456e+02],
       [ -2.35820656e+02]])
```

Stitch the images using the transformation you found by running the code below.

Note that it takes about 40 seconds for the block to finish running on a modern laptop.

In [6]:
```
matrix_transform=np.array([[R_11,R_12],[R_21,R_22]])
translation=np.array([T_x,T_y])

#Creating image canvas (the image will be constructed on this)
num_row,num_col,blah=image1.shape
image_rec=1.0*np.ones((int(num_row),int(num_col),3))

#Reconstructing the original image

LL=np.array([[0],[0]]) #lower limit on image domain
UL=np.array([[num_row],[num_col]]) #upper limit on image domain

for row in range(0,int(num_row)):
    for col in range(0,int(num_col)):
        #notice that the position is in terms of x and y, so the c
        position=np.array([[row],[col]])
        if image1[row,col,0] > 0.995 and image1[row,col,1] > 0.995 and i
mage1[row,col,2] > 0.995:
            temp =
euclidean_transform_2to1(matrix_transform,translation,image2,position,LL,
)
            image_rec[row,col,:] = temp
        else:
            image_rec[row,col,:] = image1[row,col,:]


plt.figure(figsize=(20,20))
plt.imshow(image_rec)
plt.axis('on')
plt.show()
```



In [ ]: