1. a) $\begin{matrix}1\\2\\3\end{matrix}$

$\begin{bmatrix}0\\b\\c\end{bmatrix}\begin{bmatrix}0\\a\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\end{bmatrix}$ $\quad\quad$ $\begin{matrix}1&2&3\end{matrix}$

$\begin{bmatrix}0\\0\\0\end{bmatrix}\begin{bmatrix}b\\a\\0\end{bmatrix}\begin{bmatrix}c\\0\\0\end{bmatrix}$

b) the total number of the people in the system stays the same

c) $\begin{bmatrix}0 & 0 & 0\\ 0.5 & 0.4 & 0.2\\ 0 & 0.6 & 0.65\end{bmatrix}$ People are leaving the system

d) $\begin{bmatrix}a_1 & a_2 & a_3\\ b_1 & b_2 & b_3\\ c_1 & c_2 & c_3\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} = \begin{bmatrix}a_1x_1 + a_2x_2 + a_3x_3\\ b_1x_1 + b_2x_2 + b_3x_3\\ c_1x_1 + c_2x_2 + c_3x_3\end{bmatrix}$

$a_1 + a_2 + a_3 = 1$

$b_1 + b_2 + b_3 = 1$ $= \begin{bmatrix}(a_1+a_2+a_3)x_1\\ (b_1+b_2+b_3)x_2\\ (c_1+c_2+c_3)x_3\end{bmatrix}$

$c_1 + c_2 + c_3 = 1$

$x_1 = x_2 = x_3$ $= \begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}$

Scanned by CamScanner

2. if v_n is dependent, then

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = 0$$

for some constants $a_1 \ldots a_n$

if we multiply v_n by A, we get

$$\begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n \end{bmatrix}$$

and since we know that $c_1x_1 + c_2x_2 + \cdots + c_nx_n = 0$ if we multiply each element

the linear combs of this vector can be written as

$$(a_{11} + \ldots + a_{n1})x_1 + (a_{12} \cdots + a_{n2})x_2 + \cdots (a_{1n} + \cdots + a_{nn})x_n$$

and since we know that $c_1x_1 + c_2x_2 + \cdots c_nx_n = 0$ from the original vector,

$$\frac{c_1(a_{11} + \ldots + a_{n1})}{(a_{11} + \cdots + a_{n1})}x_1 + \frac{c_2(a_{12} + \ldots + a_{n2})}{(a_{12} + \cdots + a_{n2})}x_2 + \cdots \frac{c_n(a_{1n} + \cdots + a_{nn})}{(a_{1n} + \cdots + a_{nn})}x_n = 0$$

therefore, v_n multiplied by any matrix A will be linearly dependent

Scanned by CamScanner

3. a) $P_1 = \frac{1}{2}T_1 + \frac{1}{2}T_2$

$P_2 = \frac{1}{2}T_2 + \frac{1}{2}T_3$

$P_3 = \frac{1}{2}T_3 + \frac{1}{2}T_4$

$P_4 = \frac{1}{2}T_4 + \frac{1}{2}T_5$

$P_5 = \frac{1}{2}T_5 + \frac{1}{2}T_6$

$P_6 = \frac{1}{2}T_6 + \frac{1}{2}T_1$

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \end{bmatrix}$$

$$\begin{bmatrix} \frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & -\frac{1}{2} \\ -\frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

we have a row of zeros, so this system doesn't have a unique solution

$T_1 = 2, T_2 = 4, T_3 = 4, T_4 = 4, T_5 = 4, T_6 = 4$

$T_1 = 4, T_2 = 2, T_3 = 6, T_4 = 2, T_5 = 6, T_6 = 2$

b) $$\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$ we can, because when we row reduce the coefficient matrix we get an identity matrix, meaning that $A\vec{x} = \vec{b}$ has a unique solution $\vec{x}$ since A is linearly independent

c) when n is odd, you can figure out the top

Scanned by CamScanner

4. a) $q_x = R_{xx} P_x + R_{xy} P_y + T_x$

$q_y = R_{yx} P_x + R_{yy} P_y + T_y$

there are six unknowns

you need six equations

you need three pairs of common points

b) $q_{1x} = R_{1xx} P_{1x} + R_{1xy} P_{1y} + T_{1x}$

$q_{1y} = R_{1yx} P_{1x} + R_{1yy} P_{1y} + T_{1y}$

$q_{2x} = R_{2xx} P_{2x} + R_{2xy} P_{2y} + T_{2x}$

$q_{2y} = R_{2yx} P_{2x} + R_{2yy} P_{2y} + T_{2y}$

$q_{3x} = R_{3xx} P_{3x} + R_{3xy} P_{3y} + T_{3x}$

$q_{3y} = R_{3yx} P_{3x} + R_{3yy} P_{3y} + T_{3y}$

c) $\begin{bmatrix} & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \end{bmatrix} \begin{bmatrix} R_{1xx} \\ R_{xy} \\ R_{yx} \\ R_{yy} \\ T_x \\ T_y \end{bmatrix}$

d) If $\vec{P}_1, \vec{P}_2, \vec{P}_3$ is colinear, then the ~~system~~ columns of the matrix will be linearly dependent and there won't be a unique solution. It makes sense geometrically, because you can't define a ~~3d space~~ 2d plane with less than 3 unique points that lie ~~on~~ on diffent lines.

Scanned by CamScanner

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

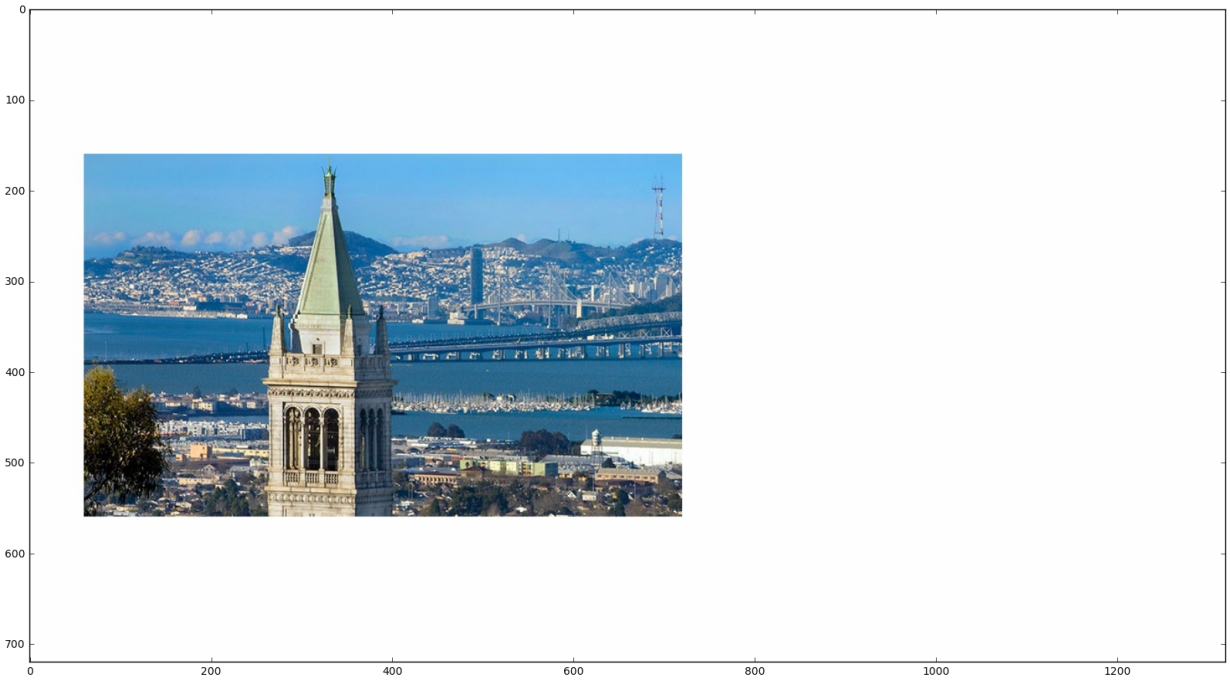
In [1]:
```
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from numpy import pi, cos, exp, sin
import matplotlib.image as mpimg
import matplotlib.transforms as mtransforms


%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
```

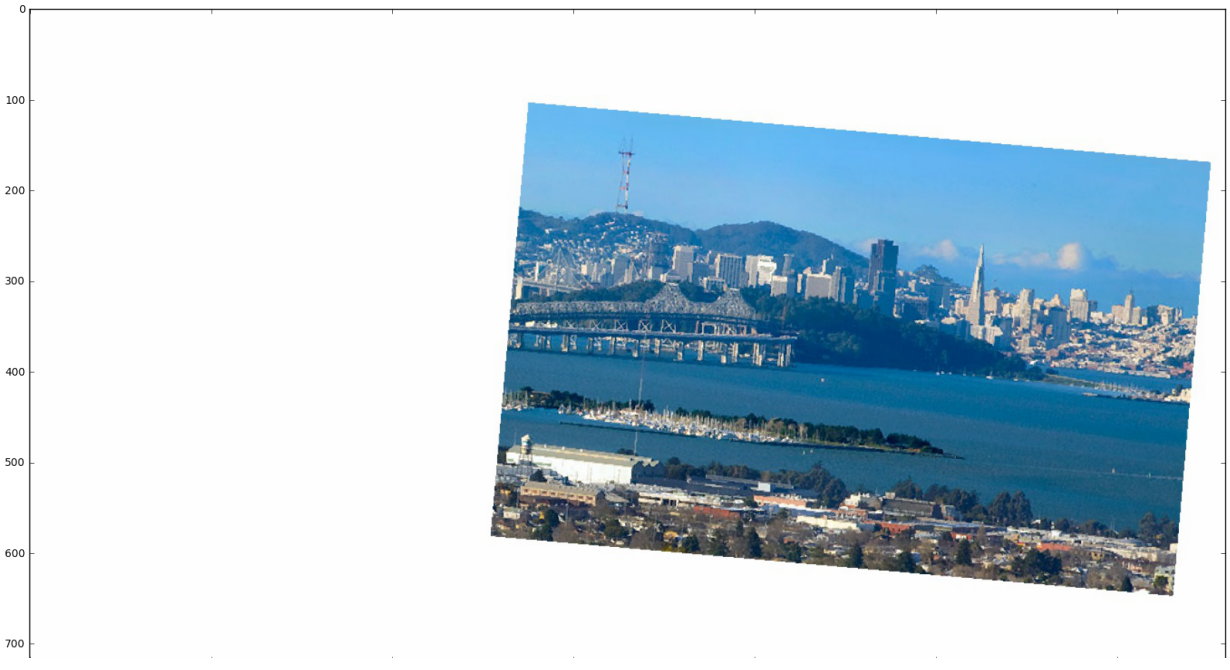
We will stick to a simple example and just consider stitching two images (if you can stitch two pictures, then you could conceivably stitch more by applying the same technique over and over again).

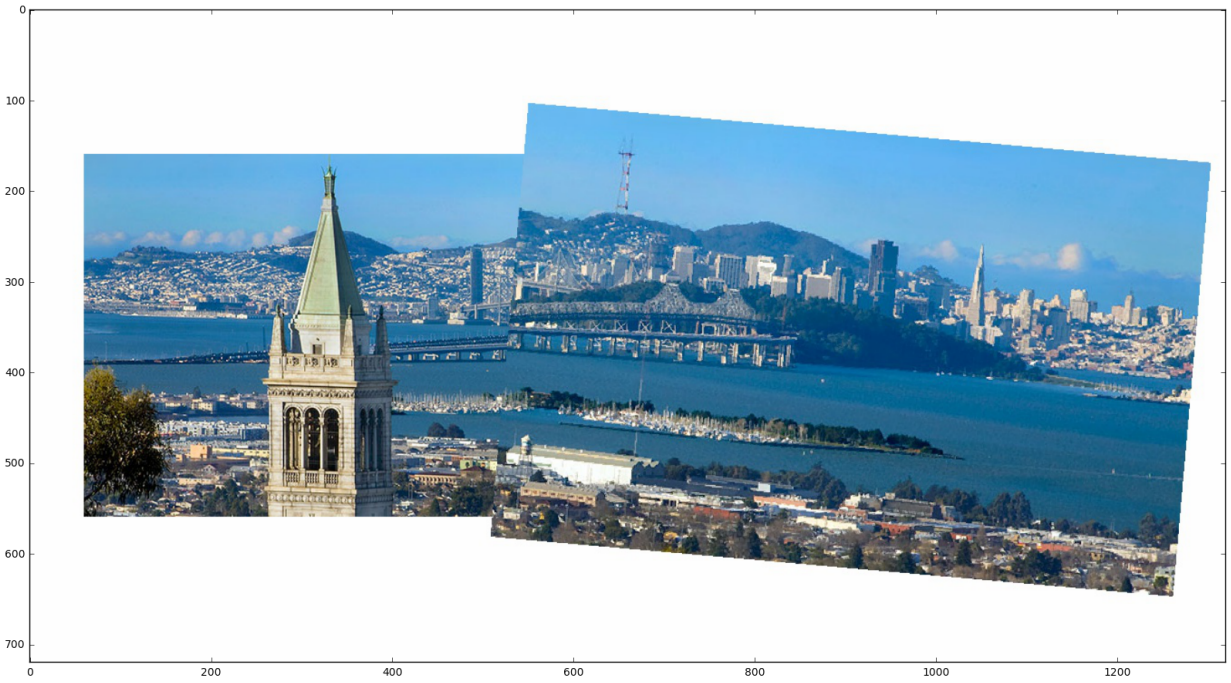
Professor Ayazifar decided to take an amazing picture of the Campanile overlooking the bay. Unfortunately, the field of view of his camera was not large enough to capture the entire scene, so he decided to take two pictues and stich them together.

The next block will display the two images.

In [2]:
```
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()
```



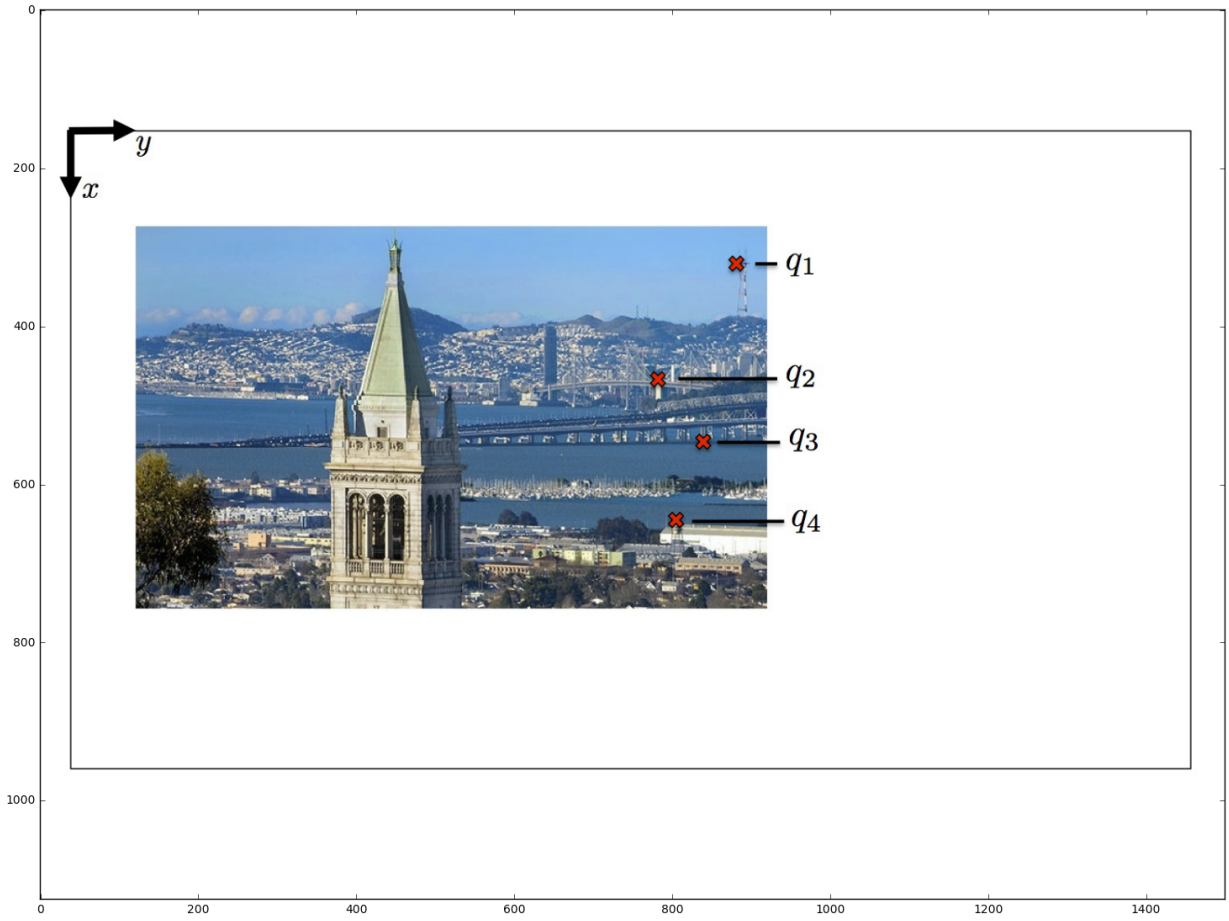


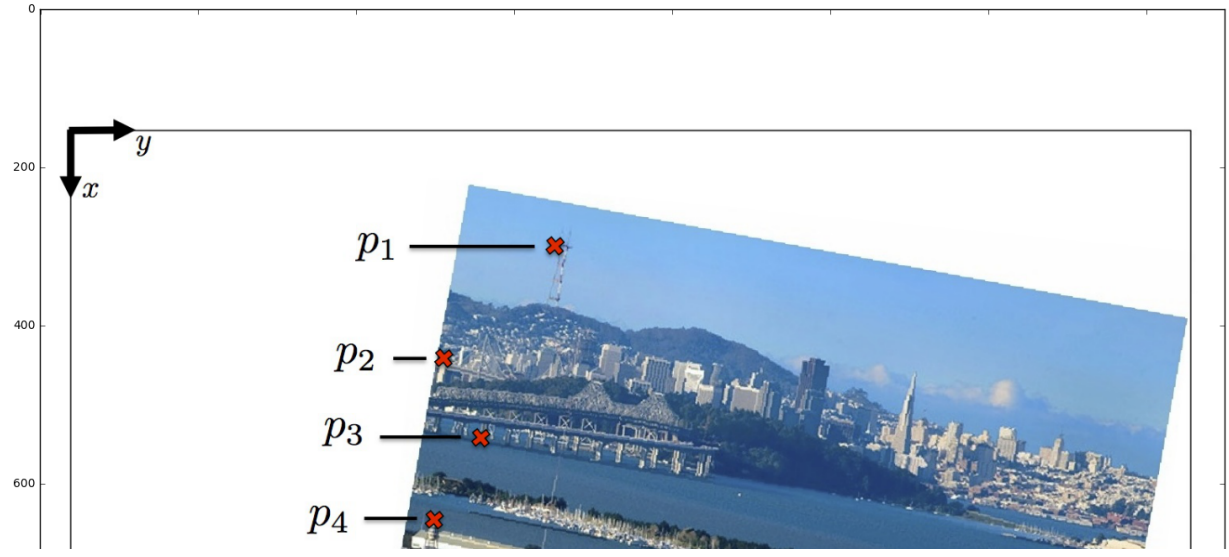


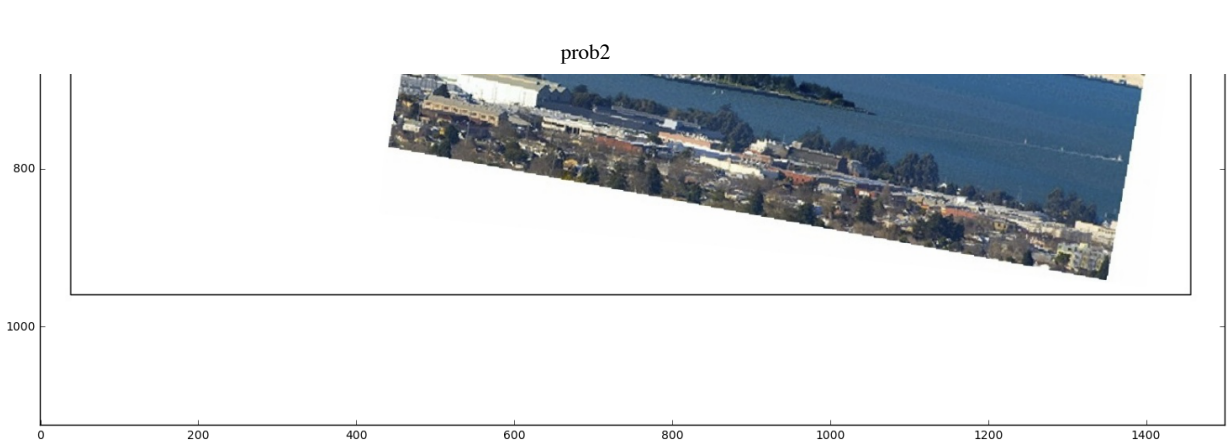
Once you apply Marcela's algorithm on the two images you get the following result (run the next block):

In [3]:
```
plt.figure(figsize=(20,30))

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

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

Out[3]: <matplotlib.image.AxesImage at 0x10bcd7e10>







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.

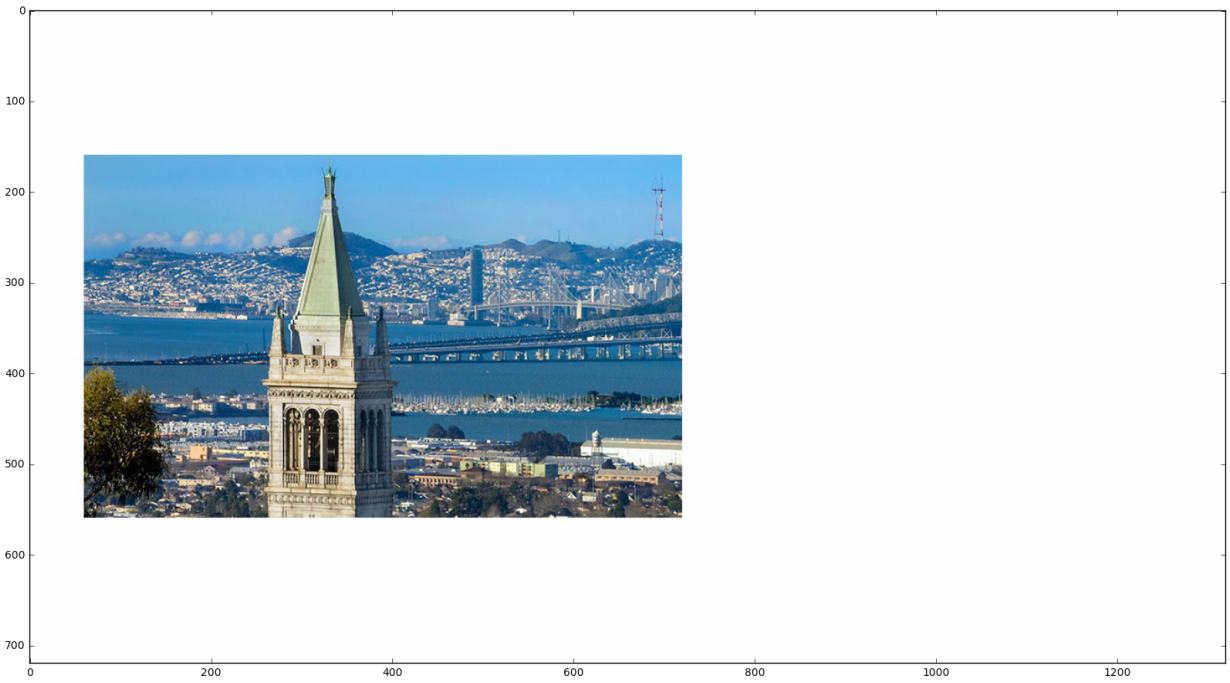
Using the points determine the parameters $R_{11}, R_{12}, R_{21}, R_{22}, T_x, T_y$ that map the points from the first image to the points in the second image by solving an appropriate system of equations. Hint: you do not need all the points to recover the parameters.

In [4]:
```
# Note that the following is a general template for solving for 6 unknowns i
# 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],
              [0,0,310,620,0,1],
              [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],[465.7892],[455.01

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]
```

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 [5]:
```
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 image
            temp = euclidean_transform_2to1(matrix_transform,translation,ima
            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 [ ]:
```
```