1. Lab Grades and Discussion Attendance

For lab grades, I should first submit my SID as a file called sid.txt to "Lab grades" on Gradescope. Then, the autograder would show my lab grades.

For discussion grades, the process is very similar: submit my SID as a file called sid.txt to "Discussion attendance" on Gradescope, and the autograder would show my discussion grades.

Additional information: sid.txt should contain only the SID number and nothing else (e.g. no "" marks or extra whitespace), and there's a template in the iPython folder.

2. (PRACTICE) Finding Charges from Potential Measurements

$k\frac{Q_1}{r_{1,1}} + k\frac{Q_2}{r_{2,1}} + k\frac{Q_3}{r_{3,1}} = U_1 = k\frac{4+3\sqrt{5}+\sqrt{10}}{2\sqrt{5}}$

$k\frac{Q_1}{r_{1,2}} + k\frac{Q_2}{r_{2,2}} + k\frac{Q_3}{r_{3,2}} = U_2 = k\frac{2+4\sqrt{2}}{\sqrt{2}}$

$k\frac{Q_1}{r_{1,3}} + k\frac{Q2}{r_{2,3}} + k\frac{Q3}{r_{3,3}} = U_3 = k\frac{4+\sqrt{5}+3\sqrt{10}}{2\sqrt{5}}$

Canceling k and calculating out the distances would give us:

$\frac{Q_1}{\sqrt{2}} + \frac{Q_2}{\sqrt{5}} + \frac{Q_3}{2} = \frac{4+3\sqrt{5}+\sqrt{10}}{2\sqrt{5}}$

$\frac{Q_1}{1} + \frac{Q_2}{\sqrt{2}} + \frac{Q_3}{1} = \frac{2+4\sqrt{2}}{\sqrt{2}}$

$\frac{Q_1}{2} + \frac{Q2}{\sqrt{5}} + \frac{Q3}{\sqrt{2}} = \frac{4+\sqrt{5}+3\sqrt{10}}{2\sqrt{5}}$

Using IPython to solve the linear equations,

$Q1 = -173.$

$Q2 = 490.$

$Q3 = -163.$

3. Figuring Out The Tips.

(a) No, we can't. Consider these two Cases:

(1) $T_1 = 2,\ T_2 = 2,\ T_3 = 6,\ T_4 = -2,\ T_5 = 4,\ T_6 = 0.$

which would give $P_1 = 2,\ P_2 = 4,\ P_3 = 2,\ P_4 = 1,\ P_5 = 2,\ P_6 = 1.$

(2). $T_1 = 1,\ T_2 = 3,\ T_3 = 5,\ T_4 = -1,\ T_5 = 3,\ T_6 = 1$

which would also give: $P_1 = 2,\ P_2 = 4,\ P_3 = 2,\ P_4 = 1,\ P_5 = 2,\ P_6 = 1.$

So, the two different assignments of T_1 to T_6 result in the same P_1 to P_6.

(b). Yes, we can. Since we have this system of linear equations:

$$\begin{aligned} 0.5T_1 + 0.5T_2 &= P_1 \\ 0.5T_2 + 0.5T_3 &= P_2 \\ 0.5T_3 + 0.5T_4 &= P_3 \\ 0.5T_4 + 0.5T_5 &= P_4 \\ 0.5T_5 + 0.5T_1 &= P_5 \end{aligned} \Rightarrow \begin{bmatrix} 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \end{bmatrix}$$

which leads to an augmented matrix, and by multiplying each row by 2:

$$\left[\begin{array}{ccccc|c} 1 & 1 & 0 & 0 & 0 & 2P_1 \\ 0 & 1 & 1 & 0 & 0 & 2P_2 \\ 0 & 0 & 1 & 1 & 0 & 2P_3 \\ 0 & 0 & 0 & 1 & 1 & 2P_4 \\ 1 & 0 & 0 & 0 & 1 & 2P_5 \end{array}\right]$$

For Row 5, subtract Row 1, Row 3 and add Row 2, Row 4. $\Rightarrow$

$$\left[\begin{array}{ccccc|c} 1 & 1 & 0 & 0 & 0 & 2P_1 \\ 0 & 1 & 1 & 0 & 0 & 2P_2 \\ 0 & 0 & 1 & 1 & 0 & 2P_3 \\ 0 & 0 & 0 & 1 & 1 & 2P_4 \\ 0 & 0 & 0 & 0 & 2 & 2(P_2+P_4+P_5-P_1-P_3) \end{array}\right] \Rightarrow \text{Divide Row 5 by 2.} \Rightarrow \left[\begin{array}{ccccc|c} 1 & 1 & 0 & 0 & 0 & 2P_1 \\ 0 & 1 & 1 & 0 & 0 & 2P_2 \\ 0 & 0 & 1 & 1 & 0 & 2P_3 \\ 0 & 0 & 0 & 1 & 1 & 2P_4 \\ 0 & 0 & 0 & 0 & 1 & P_2+P_4+P_5-P_1-P_3 \end{array}\right]$$

Now, Row 1: Subtract Row 2, Row 4, and Add Row 3, Row 5.
Row 2: Subtract Row 3, Row 5, Add Row 4.
Row 3: Subtract Row 4. Add Row 5.
Row 4: Subtract Row 5. $\Rightarrow$

$$\left[\begin{array}{ccccc|c} 1 & 0 & 0 & 0 & 0 & P_1+P_3+P_5-P_2-P_4 \\ 0 & 1 & 0 & 0 & 0 & P_1+P_2+P_4-P_3-P_5 \\ 0 & 0 & 1 & 0 & 0 & P_2+P_3+P_5-P_1-P_4 \\ 0 & 0 & 0 & 1 & 0 & P_1+P_3+P_4-P_2-P_5 \\ 0 & 0 & 0 & 0 & 1 & P_2+P_4+P_5-P_1-P_3 \end{array}\right]$$

(Next Page).

Thus, we can deduce the tips T_1 to T_5 from plates P_1 to P_5 by calculating a unique solution:

$$T_1 = P_1 - P_2 + P_3 - P_4 + P_5$$
$$T_2 = P_1 + P_2 - P_3 + P_4 - P_5$$
$$T_3 = -P_1 + P_2 + P_3 - P_4 + P_5$$
$$T_4 = P_1 - P_2 + P_3 + P_4 - P_5$$
$$T_5 = -P_1 + P_2 - P_3 + P_4 + P_5$$

(C). If and only if n is an odd integer.

For any $n \in \mathbb{Z}$ that's even, there would always be a vector that could be expressed as a linear combination of the rest of the equations, meaning that the n equations deduced are linearly dependent, so the n variables can't be solved.

On the other hand, if n is odd, then all the equations (or vectors) are linearly independent, which means that we'll have n independent equations for n variables, so we can deduce a distinct answer.

Therefore, we can determine the individual tips iff n is odd.

4. Fountain Codes

(a)

This transmission can recover from at most 3 lost symbols, and it could **always** recover from losing 1 symbol at most.

However, there are cases where it can't handle even just two lost symbols. For example, a specific pattern like losing both of the "a"s (the 1^{st} and 4^{th} symbols) – similarly, any loss that contains both "b"s (2^{nd} and 5^{th} symbols) or both "c"s (3^{rd} and 6^{th}) – can't be handled by this transmission.

4.

(b) $$\vec{k} = \begin{bmatrix} \alpha_1 & \beta_1 & \gamma_1 \\ \alpha_2 & \beta_2 & \gamma_2 \\ \alpha_3 & \beta_3 & \gamma_3 \\ \alpha_4 & \beta_4 & \gamma_4 \\ \alpha_5 & \beta_5 & \gamma_5 \\ \alpha_6 & \beta_6 & \gamma_6 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

(c) $\vec{V_1}^T = [1\ 0\ 0]$ $\vec{V_2}^T = [0\ 1\ 0]$ $\vec{V_3}^T = [0\ 0\ 1]$
$\vec{V_4}^T = [1\ 0\ 0]$ $\vec{V_5}^T = [0\ 1\ 0]$ $\vec{V_6}^T = [0\ 0\ 1]$

(d) Since $k_i = \vec{V_i}^T \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix}$ and with the information given on $\vec{k}$ and $\vec{V_i}^T$

so we only have: $1a + 0b + 0c = 7$
$1 \cdot a + 1 \cdot b + 0c = 3$
$1 \cdot a + 0 \cdot b + 1 \cdot c = 4$

$\overset{\star}{\Leftrightarrow} \left[\begin{array}{ccc|c} 1 & 0 & 0 & 7 \\ 1 & 1 & 0 & 3 \\ 1 & 0 & 1 & 4 \end{array}\right]$

Row 2: Subtract Row 1 ; Row 3: Subtract Row 1, and this leads to:

$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 7 \\ 0 & 1 & 0 & -4 \\ 0 & 0 & 1 & -3 \end{array}\right]$

So we have the solution $a = 7$, $b = -4$, $c = -3$.

Thus, Yes, I can and the message was $\begin{bmatrix} 7 \\ -4 \\ -3 \end{bmatrix}$.

P.S.:
(To justify the transformation $\star$, the system of linear equation can be expressed as

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 7 \\ 3 \\ 4 \end{bmatrix}$$

which leads to the augmented matrix)

(e)

The maximum loss is **four** symbols.

This is because that any 3 equations of the method in part (d) is linearly independent, which means that any 3 equations would give a unique solution, which could be used to recover the message. In other words, As long as the loss is less than or equal to 4 symbols, Bob can still recover the message.

(f)

Alice should use the strategy in part **(d)**.

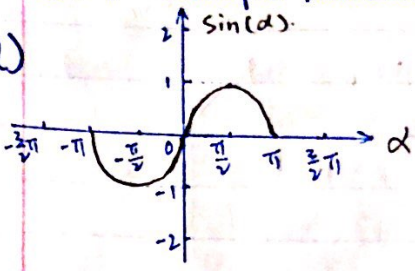
Let Alice use the strategy in part (d). Since any three equations could help Bob recover the message, so it doesn't matter which linear combination gets lost as long as other 3 are sent through. Suppose only the second linear combination is lost – with "b" being lost and the value of "a" and "c" being sent through – then her fourth linear combination would contain the value of "$a+b$". Since Bob knows the value of "a", the fourth linear combination would give him enough information to recover the entire message.

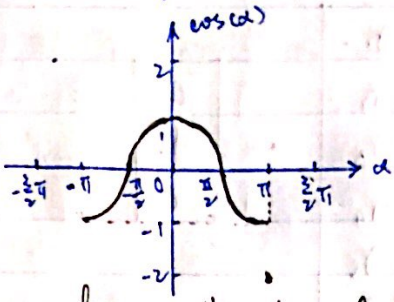
Let Alice use the strategy in part (a). Again, if only the second linear combination is lost – in this case, with "b" being lost and the value of "a" and "c" being sent through – then her fourth linear combination is useless for Bob as it only provides information for "a", which means that he can't recover the entire message.

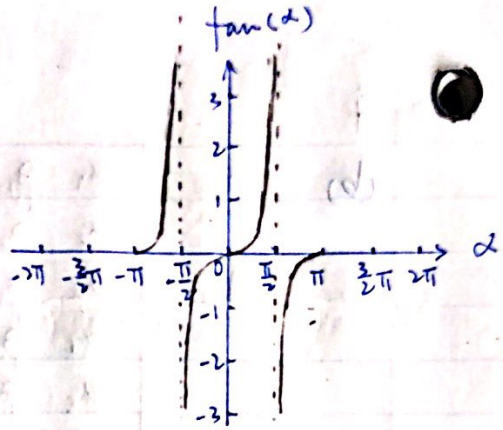
Thus, Alice should use the strategy in part (d).

5. Kinematic Model for a Simple Car

(a)







Yes, I can. As we could see, for small values of α, the value of $\sin(\alpha)$ is really close to 0 on the sin graph. Similarly, on the graphs of $\cos(\alpha)$, $\tan(\alpha)$, for small values of α ($\alpha \approx 0$); $\cos(\alpha)$ is really close to 1, and $\tan(\alpha)$ is really close to 0, which justifies the approximations.

(b). Let $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$ and $B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{bmatrix}$ and using the given information, with approximation that $\theta, \varphi \approx 0$, so $\sin\theta = \tan\varphi \approx 0$, $\cos\theta \approx 1$, we have that:

$a_{11}\cdot x[k] + a_{12}\cdot y[k] + a_{13}\cdot\theta[k] + a_{14}\cdot v[k] + b_{11}\cdot a[k] + b_{12}\cdot\varphi[k] = x[k+1] = x[k] + v[k]\cos(\theta[k])\,\Delta t.$

$a_{21}\cdot x[k] + a_{22}\cdot y[k] + a_{23}\cdot\theta[k] + a_{24}\cdot v[k] + b_{21}\cdot a[k] + b_{22}\cdot\varphi[k] = y[k+1] = y[k] + v[k]\sin(\theta[k])\,\Delta t.$

$a_{31}\cdot x[k] + a_{32}\cdot y[k] + a_{33}\cdot\theta[k] + a_{34}\cdot v[k] + b_{31}\cdot a[k] + b_{32}\cdot\varphi[k] = \theta[k+1] = \theta[k] + \frac{v[k]}{L}\tan(\varphi[k])\,\Delta t$

$a_{41}\cdot x[k] + a_{42}\cdot y[k] + a_{43}\cdot\theta[k] + a_{44}\cdot v[k] + b_{41}\cdot a[k] + b_{42}\cdot\varphi[k] = v[k+1] = v[k] + a[k]\,\Delta t.$

With given information that $L = 1.0$ m and $\Delta t = 0.1$ s, so we have that:

$x[k+1] = 1\cdot x[k] + 0\cdot y[k] + 0\cdot\theta[k] + 0.1\cdot v[k] + 0\cdot a[k] + 0\cdot\varphi[k].$

$y[k+1] = 0\cdot x[k] + 1\cdot y[k] + 0\cdot\theta[k] + 0\cdot v[k] + 0\cdot a[k] + 0\cdot\varphi[k].$

$\theta[k+1] = 0\cdot x[k] + 0\cdot y[k] + 1\cdot\theta[k] + 0\cdot v[k] + 0\cdot a[k] + 0\cdot\varphi[k].$

$v[k+1] = 0\cdot x[k] + 0\cdot y[k] + 0\cdot\theta[k] + 1\cdot v[k] + 0.1\cdot a[k] + 0\cdot\varphi[k].$

Thus, $a_{11} = a_{22} = a_{33} = a_{44} = 1$, $a_{14} = 0.1$, $b_{41} = 0.1$

$a_{12} = a_{13} = a_{21} = a_{23} = a_{24} = a_{31} = a_{32} = a_{34} = a_{41} = a_{42} = a_{43} = 0.$

$b_{11} = b_{12} = b_{21} = b_{22} = b_{31} = b_{32} = b_{42} = 0.$

Therefore, $A = \begin{bmatrix} 1 & 0 & 0 & 0.1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0.1 & 0 \end{bmatrix}$

(c) They are very similar.

Since the steering angle is so small ($\phi[k] = 0.0001$ rad), its tangent value could be approximated as 0, which is what I did in my linear approximation. In other words, because the steering angle is so small, the nonlinear system behaves just like a linear one.

(d) They are very different.

Since the steering angle is pretty big ($\phi[k] = 0.5$ rad), its tangent value could **not** be approximated as 0, which is what I did in my linear approximation. In other words, the nonlinear system in IPython takes the rather large steering angle into consideration, while my linear approximation doesn't, which is why the trajectories differ.

6. Show it.

Since $n \in \mathbb{Z}^+$, and $\{\vec{v}_1, \vec{v}_2, \cdots, \vec{v}_k\}$ is a set of linearly dependent vectors in $\mathbb{R}^n$, so by the definition of linear dependence, we have that:

there exists an index i and scalars α_j's such that

$$\vec{v}_i = \sum_{j \neq i} \alpha_j \vec{v}_j \qquad (1)$$

Take any $n \times n$ matrix A, multiply both sides of Eq.(1) by A, so

$$A\vec{v}_i = A\left(\sum_{j \neq i} \alpha_j \vec{v}_j\right).$$

and by distributivity of matrix-vector multiplication, so we have:

$$A\vec{v}_i = \sum_{j \neq i} \left(A\alpha_j \vec{v}_j\right) = \sum_{j \neq i} \alpha_j (A\vec{v}_j).$$

which means that there exists an index i and scalars α_j's for the set $\{A\vec{v}_1, A\vec{v}_2, \cdots, A\vec{v}_k\}$ such that $A\vec{v}_i = \sum_{j \neq i} \alpha_j (A\vec{v}_j)$

Thus, by definition of linear dependence, the set

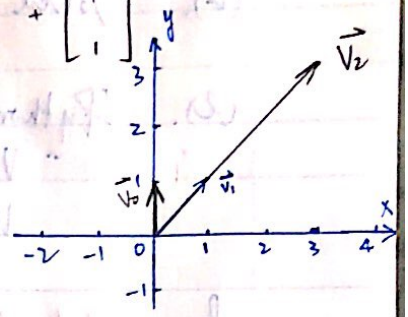
$\{A\vec{v}_1, A\vec{v}_2, \cdots, A\vec{v}_k\}$ is a set of linear dependent vectors.

Q.E.D.

7. Image Stitching.

(a). Using the given info, we have $\vec{V_2} = \begin{bmatrix} 2 & 2 \\ -2 & 2 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

So $\vec{V_2} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$



$\vec{V_2}$ is transformed from $\vec{V_0}$ by:
→. first rotating 45° clockwise and then scaled by 3.

(b). From Equation (5), we can get:

$$p_x \cdot R_{xx} + p_y \cdot R_{xy} + T_x = q_x \quad \text{(i)}$$
$$p_x \cdot R_{yx} + p_y \cdot R_{yy} + T_y = q_y \quad \text{(ii)}$$

The known values in Eq(i) are p_x, p_y, q_x. The unknowns are R_{xx}, R_{xy}, T_x.
In Eq (ii), the known values are p_x, p_y, q_y. The unknowns are R_{yx}, R_{yy}, T_y.
So, there are $\boxed{6}$ unknowns,
and I need $\boxed{6}$ independent equations to solve them all, which means I need $\boxed{3}$ pairs of common points $\vec{p}$ and $\vec{q}$.

(c) The vector of the unknown values is:

$$\begin{bmatrix} R_{xx} \\ R_{xy} \\ R_{yx} \\ R_{yy} \\ T_x \\ T_y \end{bmatrix}$$

Let the three pairs of common points be $(p_1, q_1), (p_2, q_2), (p_3, q_3)$.
So, we have linear equations:

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

which can be transformed into:

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

(d). Solved Successfully.

(e). IPython throws out an error:
"ValueError: Rows are not linearly independent. Cannot solve system of linear equations uniquely. :)"

(f). Let $\vec{p}_1, \vec{p}_2, \vec{p}_3$ be collinear, so using the Fact, we have that:
$(\vec{p}_2 - \vec{p}_1) = k(\vec{p}_3 - \vec{p}_1)$ for some $k \in \mathbb{R}$.
Thus, separating into values (sub-vectors) on x- and y- axis, we have:

$$(p_{2x} - p_{1x}) = k(p_{3x} - p_{1x}) \Rightarrow p_{2x} = k \cdot p_{3x} - (k-1)p_{1x}$$
$$\text{and } (p_{2y} - p_{1y}) = k(p_{3y} - p_{1y}) \quad p_{2y} = k \cdot p_{3y} - (k-1)p_{1y}$$

Since the linear system in part (c) could be transformed into an augmented matrix:

$$\left[\begin{array}{cccccc|c} p_{1x} & p_{1y} & 0 & 0 & 1 & 0 & q_{1x} \\ 0 & 0 & p_{1x} & p_{1y} & 0 & 1 & q_{1y} \\ p_{2x} & p_{2y} & 0 & 0 & 1 & 0 & q_{2x} \\ 0 & 0 & p_{2x} & p_{2y} & 0 & 1 & q_{2y} \\ p_{3x} & p_{3y} & 0 & 0 & 1 & 0 & q_{3x} \\ 0 & 0 & p_{3x} & p_{3y} & 0 & 1 & q_{3y} \end{array}\right]$$

Row 3: Add $(k-1)\cdot$Row 1 and Subtract $k\cdot$Row 5. $\Rightarrow$

$$\left[\begin{array}{cccccc|c} p_{1x} & p_{1y} & 0 & 0 & 1 & 0 & q_{1x} \\ 0 & 0 & p_{1x} & p_{1y} & 0 & 1 & q_{1y} \\ 0 & 0 & 0 & 0 & 0 & 0 & q_{2x} + (k-1)q_{1x} - k \cdot q_{3x} \\ 0 & 0 & p_{2x} & p_{2y} & 0 & 1 & q_{2y} \\ p_{3x} & p_{3y} & 0 & 0 & 1 & 0 & q_{3x} \\ 0 & 0 & p_{3x} & p_{3y} & 0 & 1 & q_{3y} \end{array}\right]$$

and since $p_{2x} + (k-1)p_{1x} - k\,p_{3x} = 0$
and $p_{2y} + (k-1)p_{2y} - k\,p_{3y} = 0$,
and $1 + (k-1) - k = 0$
and $0 + (k-1)\cdot 0 - k\cdot 0 = 0$, so $\Rightarrow$:

Now, we have linearly dependent vectors.
Since we now have a row of 0s, we reached a stopping condition, and since its right-hand side $q_{2x} + (k-1)q_{1x} - k\cdot q_{3x} = 0$ by definition of collinear,
Thus, this system of equations have an infinite number of solutions.

Q6.17.

8. Homework Process and Study Group

I worked alone without getting any help on Problems 1-6, except asking about format questions on Piazza and reading the Notes of the course.

I worked with Chelsea Ye (SID: 3034166490, Email: chelseaye@berkeley.edu) on Problem 7.

EE16A: Homework 2

Problem 2: Finding Charges from Potential Measurements

In [5]:
```
# Your code here.

import numpy as np
a = np.array([
    [1/np.sqrt(2), 1/np.sqrt(5), 1/2],
    [1, 1/np.sqrt(2), 1],
    [1/2, 1/np.sqrt(5), 1/np.sqrt(2)]
])

b = np.array([
    (4 + 3*np.sqrt(5) + np.sqrt(10)) / (2/np.sqrt(5)),
    (2 + 4*np.sqrt(2)) / (1/np.sqrt(2)),
    (4 + np.sqrt(5) + 3*np.sqrt(10)) / (2/np.sqrt(5))
])
x = np.linalg.solve(a, b)

print(x)
```

```
[-172.71603496  489.68629104 -162.71603496]
```

Problem 5: Kinematic Model for a Simple Car

This script helps to visualize the difference between a nonlinear model and a corresponding linear approximation for a simple car. What you should notice is that the linear model is similar to the nonlinear model when you are close to the point where the approximation is made.

First, run the following block to set up the helper functions needed to simulate the vehicle models and plot the trajectories taken.

In [7]:
```
# DO NOT MODIFY THIS BLOCK!
''' Problem/Model Setup'''
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Vehicle Model Constants
L = 1.0  # length of the car, meters
dt = 0.1 # time difference between timestep (k+1) and timestep k, seco

''' Nonlinear Vehicle Model Update Equation '''
```

```
    Nonlinear vehicle model update equation
def nonlinear_vehicle_model(initial_state, inputs, num_steps):
    x     = initial_state[0] # x position, meters
    y     = initial_state[1] # y position, meters
    theta = initial_state[2] # heading (wrt x-axis), radians
    v     = initial_state[3] # speed, meters per second

    a = inputs[0]            # acceleration, meters per second squared
    phi = inputs[1]          # steering angle, radians

    state_history = []       # array to hold state values as the time s
    state_history.append([x,y,theta,v]) # add the initial state (i.e. t

    for i in range(0, num_steps):
        # Find the next state, at time k+1, by applying the nonlinear m
        x_next     = x     + v * np.cos(theta) * dt
        y_next     = y     + v * np.sin(theta) * dt
        theta_next = theta + v/L * np.tan(phi) * dt
        v_next     = v     + a * dt

        # Add the next state to the history.
        state_history.append([x_next,y_next,theta_next,v_next])

        # Advance to the next state, at time k+1, to get ready for next
        x = x_next
        y = y_next
        theta = theta_next
        v = v_next

    return np.array(state_history)

''' Linear Vehicle Model Update Equation '''
def linear_vehicle_model(A, B, initial_state, inputs, num_steps):
    # Note: A should be a 4x4 matrix, B should be a 4x2 matrix for this

    x     = initial_state[0] # x position, meters
    y     = initial_state[1] # y position, meters
    theta = initial_state[2] # heading (wrt x-axis), radians
    v     = initial_state[3] # speed, meters per second

    a = inputs[0]            # acceleration, meters per second squared
    phi = inputs[1]          # steering angle, radians

    state_history = []       # array to hold state values as the time s
    state_history.append([x,y,theta,v]) # add the initial state (i.e. t

    for i in range(0, num_steps):
        # Find the next state, at time k+1, by applying the nonlinear m
        state_next = np.dot(A, state_history[-1]) + np.dot(B, inputs)

        # Add the next state to the history.
        state_history.append(state_next)

        # Advance to the next state, at time k+1, to get ready for next
        state = state_next
```

```
        state = state_next

    return np.array(state_history)

''' Plotting Setup'''
def make_model_comparison_plot(state_predictions_nonlinear, state_predi
    f = plt.figure()
    plt.plot(state_predictions_nonlinear[0,0], state_predictions_nonli
    plt.plot(state_predictions_nonlinear[:,0], state_predictions_nonli
    plt.plot(state_predictions_linear[:,0], state_predictions_linear[:,
    plt.legend(loc='upper left')
    plt.xlim([4, 8])
    plt.ylim([9, 12])
    plt.show()
```

Part B

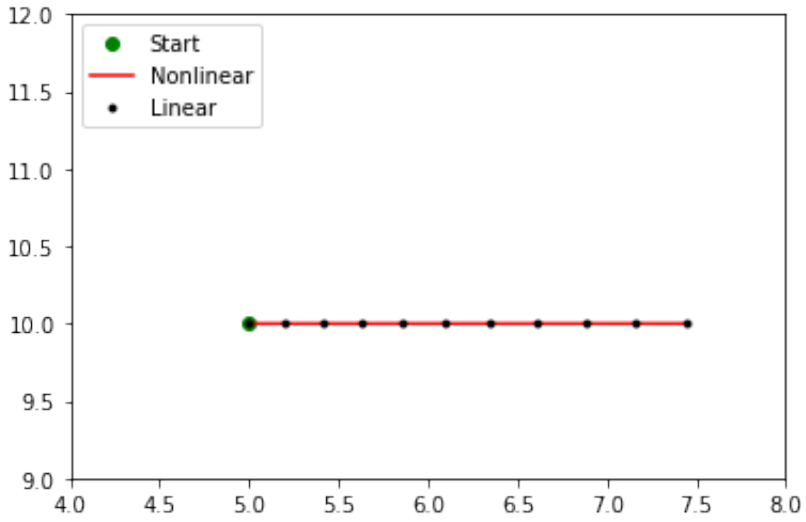
Task: Fill in the matrices A and B for the linear system approximating the nonlinear vehicle model under small heading and steering angle approximations.

In [15]:
```
# Your code here.
A = np.array([[1, 0, 0, 0.1],
              [0, 1, 0, 0],
              [0, 0, 1, 0],
              [0, 0, 0, 1]])

B = np.array([[ 0, 0],
              [ 0, 0],
              [ 0, 0],
              [ 0.1, 0]])
```

Part C

Task: Fill out the state and input values from Part C and look at the resulting plot. The plot should help you to visualize the difference between using a linear model and a nonlinear model for this specific starting state and input.

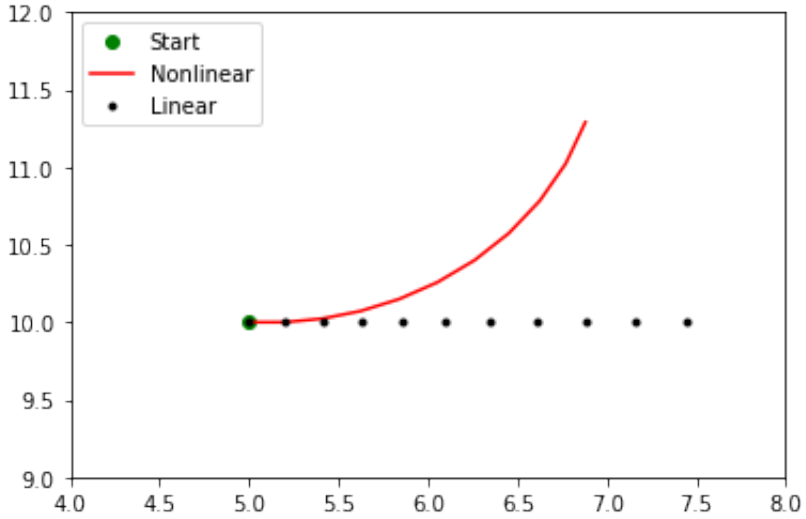
In [16]:
```
# Your code here.
x_init  = 5.0
y_init  = 10.0
theta_init = 0.0
v_init     = 2.0
a_input    = 1.0
phi_input  = 0.0001

state_init = [x_init, y_init, theta_init, v_init]
state_predictions_nonlinear = nonlinear_vehicle_model(state_init, [a_ir
state_predictions_linear = linear_vehicle_model(A, B, state_init, [a_ir

make_model_comparison_plot(state_predictions_nonlinear, state_predictio
```



Part D

Task: Fill out the state and input values from Problem D and look at the resulting plot. The plot should help you to visualize the difference between using a linear model and a nonlinear model for this specific starting state and input.

In [17]:
```
# Your code here.
x_init  = 5.0
y_init  = 10.0
theta_init = 0.0
v_init     = 2.0
a_input    = 1.0
phi_input  = 0.5

state_init = [x_init, y_init, theta_init, v_init]
state_predictions_nonlinear = nonlinear_vehicle_model(state_init, [a_ir
state_predictions_linear = linear_vehicle_model(A, B, state_init, [a_ir

make_model_comparison_plot(state_predictions_nonlinear, state_predictic
```



Problem 7: Image Stitching

This section of the notebook continues the image stitching 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 [13]:
```
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt
```

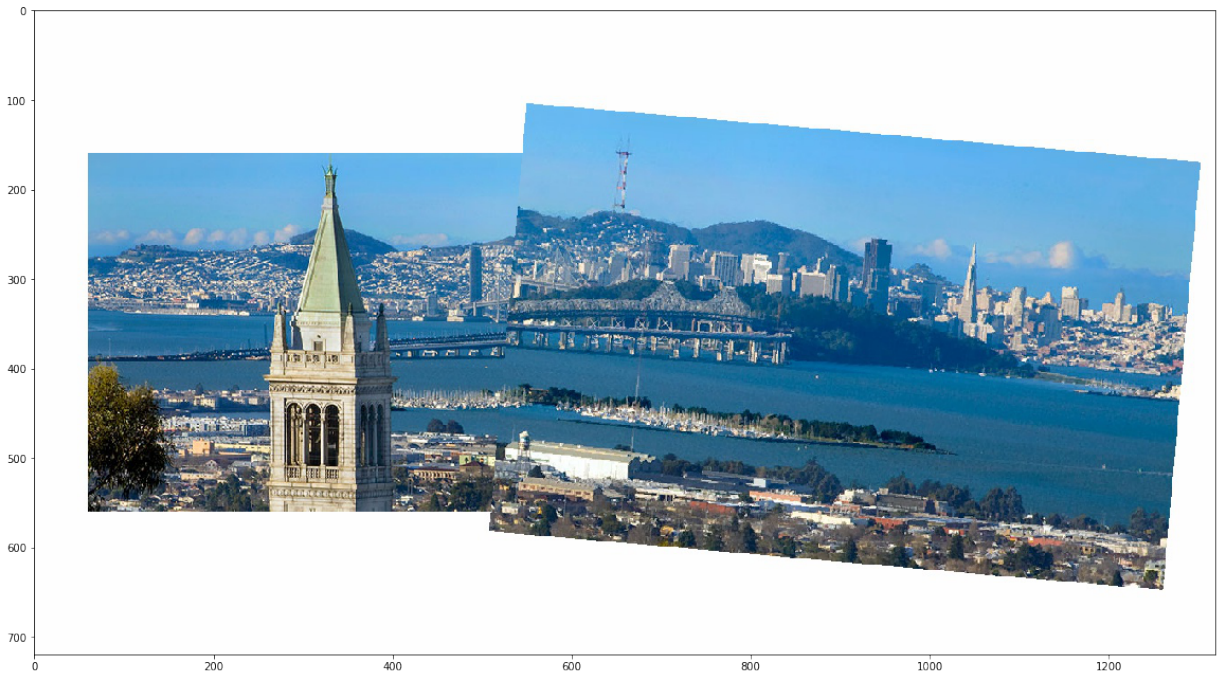
```
from mpl_toolkits.mplot3d import Axes3D
from numpy import pi, cos, exp, sin
import matplotlib.image as mpimg
import matplotlib.transforms as mtransforms


#%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,1
    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,1
    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

def solve(A,b):
    try:
        z = np.linalg.solve(A,b)
    except:
        raise ValueError('Rows are not linearly independent. Cannot sol
    return z
```

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).

Daniel 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 stitch them together.

The next block will display the two images.

In [14]:
```
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()
```

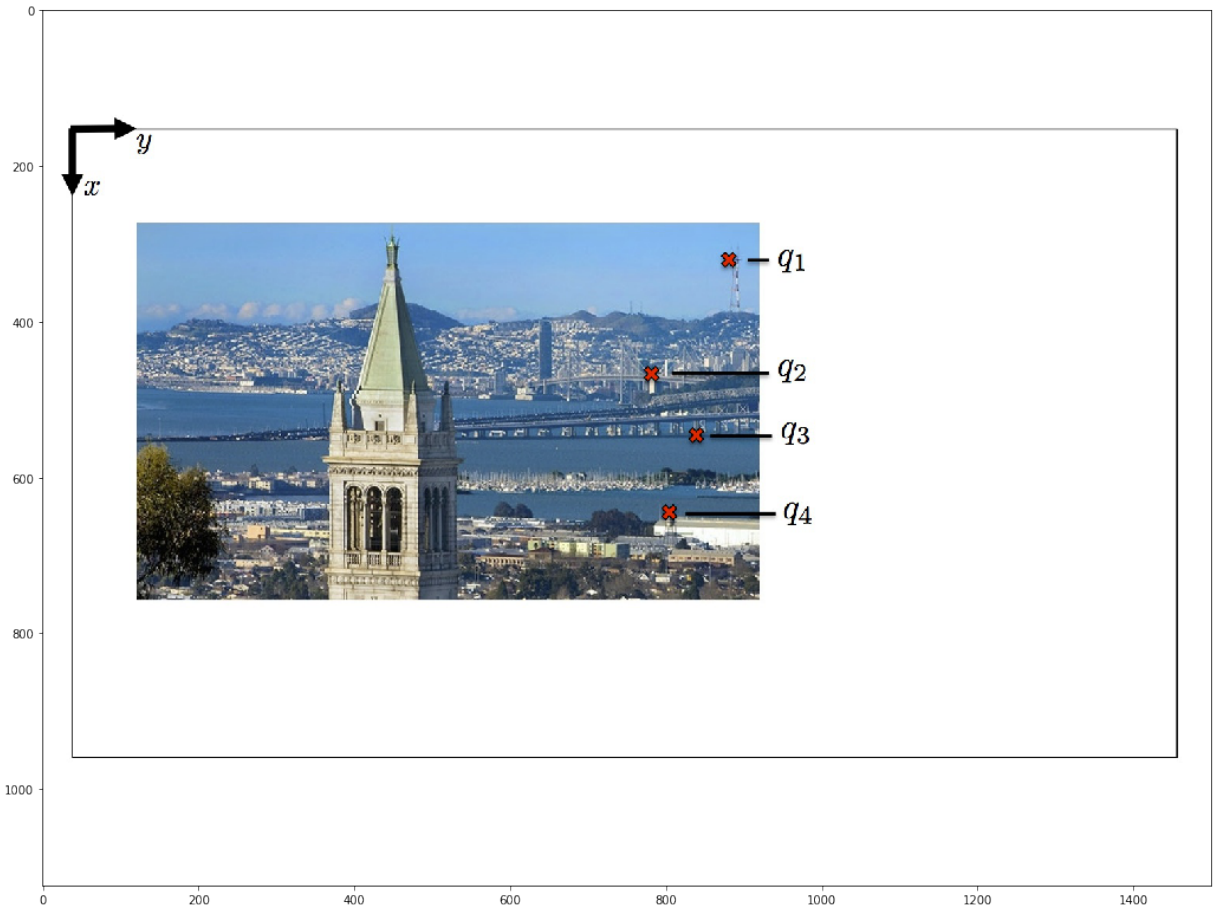


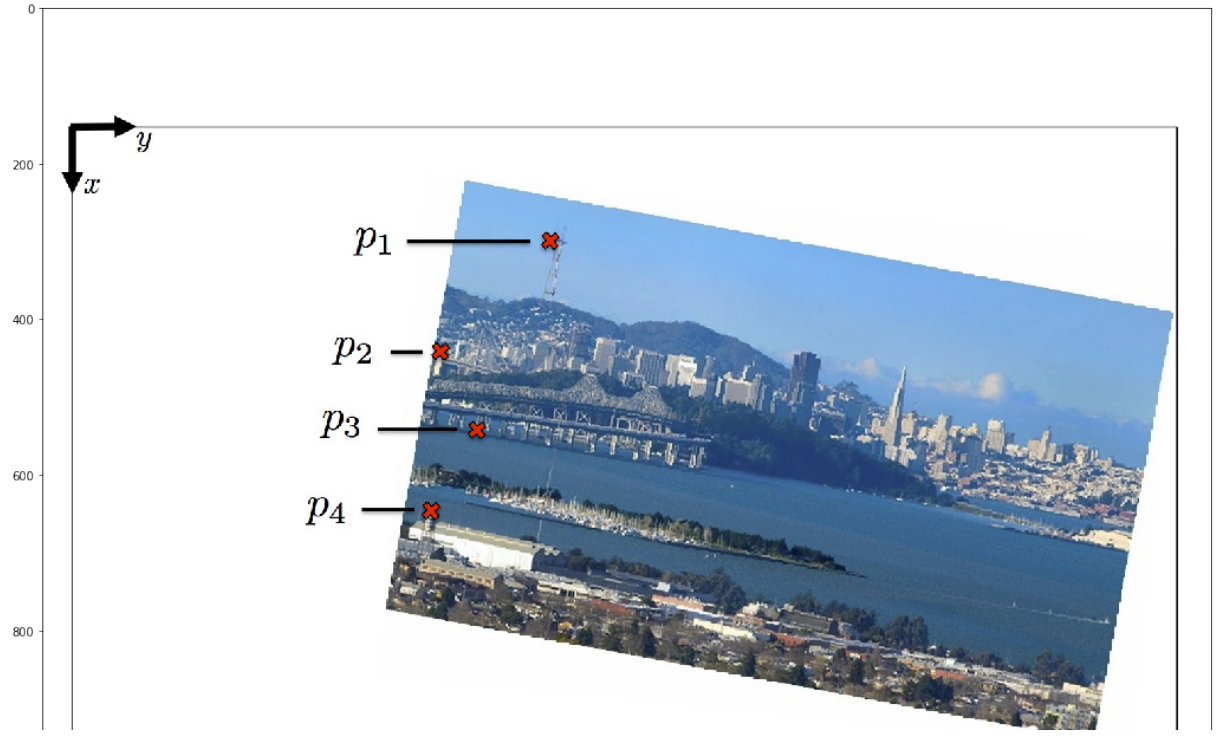
Once you apply Marcela's algorithm on the two images you get the following result (run the next block):

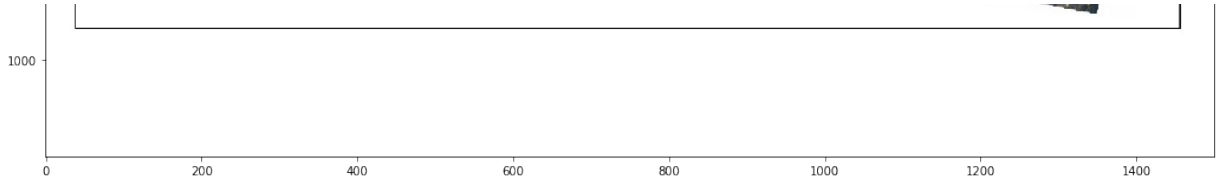
In [15]:
```
plt.figure(figsize=(20,30))

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

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

Out[15]: <matplotlib.figure.AxesImage at 0x1182dbc50>







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

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

$$\vec{q_1} = \begin{bmatrix} 162.2976 \\ 565.8862 \end{bmatrix} \quad \vec{q_2} = \begin{bmatrix} 285.4283 \\ 458.7469 \end{bmatrix} \quad \vec{q_3} = \begin{bmatrix} 385.2465 \\ 498.1973 \end{bmatrix} \quad \vec{q_4} = \begin{bmatrix} 46 \\ 45 \end{bmatrix}$$

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.

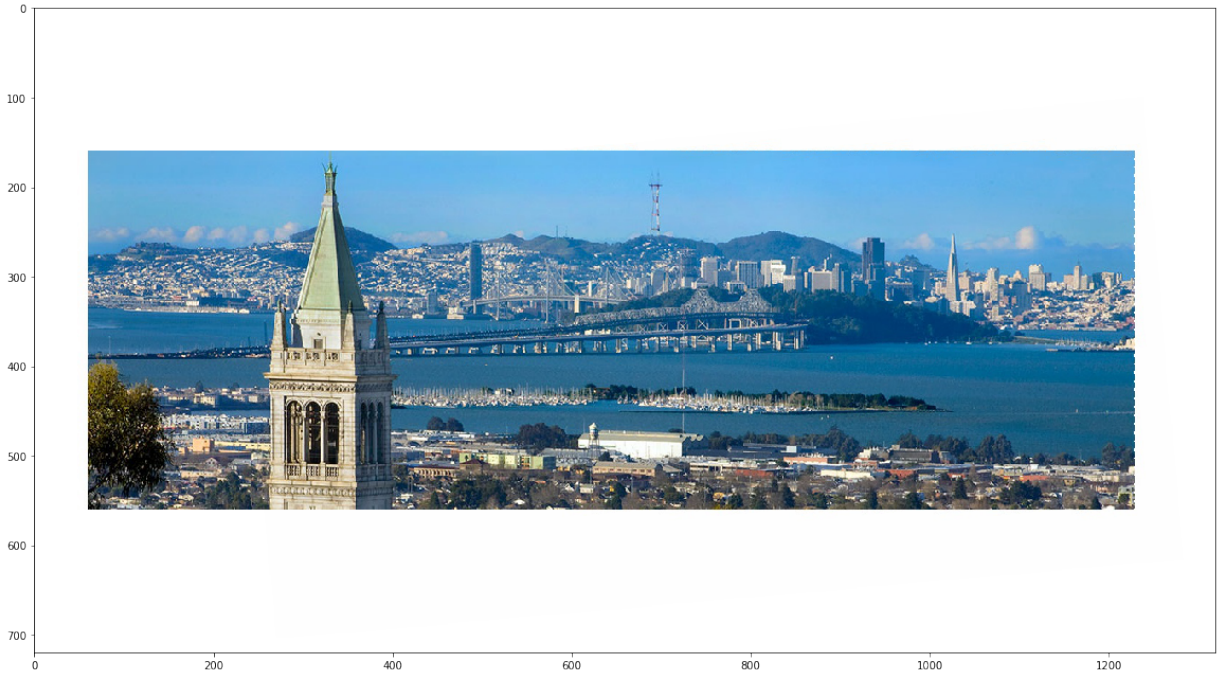
In [19]:
```
# Note that the following is a general template for solving for 6 unkn
# You do not have to use the following code exactly.
# All you need to do is to find parameters R_11, R_12, R_21, R_22, T_x,
# If you prefer finding them another way it is fine.

# 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],
              [385.2465],[498.1973]])

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

# solve the linear system for the coefficiens
z = 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 [20]:
```
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
            temp = euclidean_transform_2to1(matrix_transform,translatio
            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()
```

```
Clipping input data to the valid range for imshow with RGB data ([0.
.1] for floats or [0..255] for integers).
```



Part E: Failure Mode Points

$$\vec{p}_1 = \begin{bmatrix} 390 \\ 660 \end{bmatrix} \quad \vec{p}_2 = \begin{bmatrix} 425 \\ 645 \end{bmatrix} \quad \vec{p}_3 = \begin{bmatrix} 460 \\ 630 \end{bmatrix}$$

$$\vec{q}_1 = \begin{bmatrix} 385 \\ 450 \end{bmatrix} \quad \vec{q}_2 = \begin{bmatrix} 425 \\ 480 \end{bmatrix} \quad \vec{q}_3 = \begin{bmatrix} 465 \\ 510 \end{bmatrix}$$

In [21]:
```
# Note that the following is a general template for solving for 6 unkn
# 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
# If you prefer finding them another way it is fine.

# fill in the entries
A = np.array([[390,660,0,0,1,0],
              [0,0,390,660,0,1],
              [425,645,0,0,1,0],
              [0,0,435,645,0,1],
              [460,630,0,0,1,0],
              [0,0,460,630,0,1]])

# fill in the entries
b = np.array([[385],[450],
              [425],[480],
              [465],[510]])

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

# solve the linear system for the coefficiens
z = 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]
```

```
---------------------------------------------------------------------
-------
LinAlgError                               Traceback (most recent cal
l last)
<ipython-input-13-48a5c261aef1> in solve(A, b)
     51     try:
---> 52         z = np.linalg.solve(A,b)
     53     except:

/anaconda3/lib/python3.6/site-packages/numpy/linalg/linalg.py in sol
ve(a, b)
    389     extobj = get_linalg_error_extobj(_raise_linalgerror_sing
```

```
ular)
--> 390     r = gufunc(a, b, signature=signature, extobj=extobj)
    391

/anaconda3/lib/python3.6/site-packages/numpy/linalg/linalg.py in _ra
ise_linalgerror_singular(err, flag)
     88 def _raise_linalgerror_singular(err, flag):
---> 89     raise LinAlgError("Singular matrix")
     90

LinAlgError: Singular matrix

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent cal
l last)
<ipython-input-21-b7b55b1394a8> in <module>()
     21
     22 # solve the linear system for the coefficiens
---> 23 z = solve(A,b)
     24
     25 #Parameters for our transformation

<ipython-input-13-48a5c261aef1> in solve(A, b)
     52         z = np.linalg.solve(A,b)
     53     except:
---> 54         raise ValueError('Rows are not linearly independent.
Cannot solve system of linear equations uniquely. :)')
     55     return z

ValueError: Rows are not linearly independent. Cannot solve system o
f linear equations uniquely. :)
```

In [ ]: