EECS 16A HW03

Bryan Ngo

2019-09-17

0 Matrix Multiplication

0.a

Given $\mathbf{A} \in \mathbb{R}^{3\times 2}$ and $\mathbf{C} \in \mathbb{R}^{2\times 4}$, the resulting matrix product will be $\mathbf{AB} \in \mathbb{R}^{3\times 4}$.

0.b

$$\begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & -1 & 0 \\ -3 & 0 & 2 & -1 \end{bmatrix} \tag{1}$$

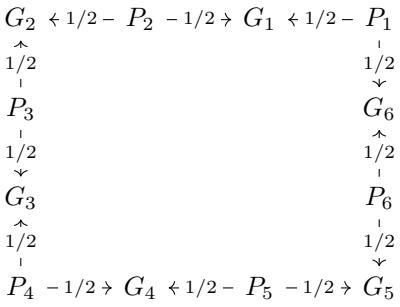
$$= \begin{bmatrix} 1(1)+0(-3) & 2(1)+0(0) & 1(-1)+0(2) & 1(0)+0(-1) \\ 2(1)+1(-3) & 2(1)+1(0) & 2(-1)+1(2) & 2(0)+1(-1) \\ 0(1)+1(-3) & 0(1)+1(0) & 0(-1)+1(2) & 0(0)+1(-1) \end{bmatrix} \tag{2}$$

$$= \begin{bmatrix} 1 & 2 & -1 & 0 \\ -1 & 2 & 0 & -1 \\ -3 & 0 & 2 & -1 \end{bmatrix} \tag{3}$$

1 Figuring Out the Tips

1.a

We can visualize the tipping process as a state diagram of the dinner:



Converting this state diagram into a transition matrix $\mathbf{T}$,

$$\mathbf{T} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} \end{bmatrix} \tag{4}$$

Converting to an augmented matrix,

$$\left[\begin{array}{cccccc|c} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & G_1 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & G_2 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & G_3 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & G_4 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & G_5 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & G_6 \end{array}\right] \tag{5}$$

$$\xRightarrow{2r \to r} \left[\begin{array}{cccccc|c} 1 & 1 & 0 & 0 & 0 & 0 & 2G_1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 2G_2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 2G_3 \\ 0 & 0 & 0 & 1 & 1 & 0 & 2G_4 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2G_5 \\ 1 & 0 & 0 & 0 & 0 & 1 & 2G_6 \end{array}\right] \xRightarrow{r_6 - r_1 \to r_6} \left[\begin{array}{cccccc|c} 1 & 1 & 0 & 0 & 0 & 0 & 2G_1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 2G_2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 2G_3 \\ 0 & 0 & 0 & 1 & 1 & 0 & 2G_4 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2G_5 \\ 0 & -1 & 0 & 0 & 0 & 1 & 2G_6 - 2G_1 \end{array}\right] \tag{6}$$

$$\xRightarrow{r_6 + r_2 \to r_6} \left[\begin{array}{cccccc|c} 1 & 1 & 0 & 0 & 0 & 0 & 2G_1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 2G_2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 2G_3 \\ 0 & 0 & 0 & 1 & 1 & 0 & 2G_4 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2G_5 \\ 0 & 0 & 1 & 0 & 0 & 1 & 2G_6 + \cdots \end{array}\right] \xRightarrow{r_6 - r_3 \to r_6} \left[\begin{array}{cccccc|c} 1 & 1 & 0 & 0 & 0 & 0 & 2G_1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 2G_2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 2G_3 \\ 0 & 0 & 0 & 1 & 1 & 0 & 2G_4 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2G_5 \\ 0 & 0 & 0 & -1 & 0 & 1 & 2G_6 + \cdots \end{array}\right] \tag{7}$$

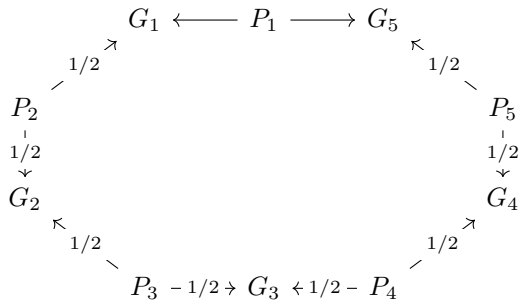
$$\xRightarrow{r_6 + r_4 \to r_6} \left[\begin{array}{cccccc|c} 1 & 1 & 0 & 0 & 0 & 0 & 2G_1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 2G_2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 2G_3 \\ 0 & 0 & 0 & 1 & 1 & 0 & 2G_4 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2G_5 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2G_6 + \cdots \end{array}\right] \xRightarrow{r_6 - r_5 \to r_6} \left[\begin{array}{cccccc|c} 1 & 1 & 0 & 0 & 0 & 0 & 2G_1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 2G_2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 2G_3 \\ 0 & 0 & 0 & 1 & 1 & 0 & 2G_4 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2G_5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2G_6 + \cdots \end{array}\right] \tag{8}$$

As clearly demonstrated, the row of zeroes tells us there are either 0 or ∞ solutions, depending on the values of G.

From the geometry of the setup, we can clearly see there is symmetry between alternating plates. That is, if we switched the position of two plates $P_{1,3,5}$ or $P_{2,4,6}$, the solution between them would be indistinguishable. This means that there are multiple starting conditions that map to the same result, making it impossible to recover the original tip values.

1.b

We can create a similar state diagram as in 1.a:



Our transition matrix $\mathbf{T}$ is also very similar:

$$\mathbf{T} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \end{bmatrix} \tag{9}$$

Performing similar operations on the augmented matrix,

$$\left[\begin{array}{ccccc|c} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & G_1 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & G_2 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & G_3 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & G_4 \\ \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & G_5 \end{array}\right] \tag{10}$$

$$\overset{2r\to r}{\Longrightarrow} \left[\begin{array}{ccccc|c} 1 & 1 & 0 & 0 & 0 & 2G_1 \\ 0 & 1 & 1 & 0 & 0 & 2G_2 \\ 0 & 0 & 1 & 1 & 0 & 2G_3 \\ 0 & 0 & 0 & 1 & 1 & 2G_4 \\ 1 & 0 & 0 & 0 & 1 & 2G_5 \end{array}\right] \overset{r_5-r_1\to r_5}{\Longrightarrow} \left[\begin{array}{ccccc|c} 1 & 1 & 0 & 0 & 0 & 2G_1 \\ 0 & 1 & 1 & 0 & 0 & 2G_2 \\ 0 & 0 & 1 & 1 & 0 & 2G_3 \\ 0 & 0 & 0 & 1 & 1 & 2G_4 \\ 0 & -1 & 0 & 0 & 1 & 2G_5 - 2G_1 \end{array}\right] \tag{11}$$

$$\overset{r_5+r_2\to r_5}{\Longrightarrow} \left[\begin{array}{ccccc|c} 1 & 1 & 0 & 0 & 0 & 2G_1 \\ 0 & 1 & 1 & 0 & 0 & 2G_2 \\ 0 & 0 & 1 & 1 & 0 & 2G_3 \\ 0 & 0 & 0 & 1 & 1 & 2G_4 \\ 0 & 0 & 1 & 0 & 1 & 2G_5 + \cdots \end{array}\right] \overset{r_5-r_3\to r_5}{\Longrightarrow} \left[\begin{array}{ccccc|c} 1 & 1 & 0 & 0 & 0 & 2G_1 \\ 0 & 1 & 1 & 0 & 0 & 2G_2 \\ 0 & 0 & 1 & 1 & 0 & 2G_3 \\ 0 & 0 & 0 & 1 & 1 & 2G_4 \\ 0 & 0 & 0 & -1 & 1 & 2G_5 + \cdots \end{array}\right] \tag{12}$$

$$\overset{r_5+r_4\to r_5}{\Longrightarrow} \left[\begin{array}{ccccc|c} 1 & 1 & 0 & 0 & 0 & 2G_1 \\ 0 & 1 & 1 & 0 & 0 & 2G_2 \\ 0 & 0 & 1 & 1 & 0 & 2G_3 \\ 0 & 0 & 0 & 1 & 1 & 2G_4 \\ 0 & 0 & 0 & 0 & 2 & 2G_5 + \cdots \end{array}\right] \overset{\frac{1}{2}r_5\to r_5}{\Longrightarrow} \left[\begin{array}{ccccc|c} 1 & 1 & 0 & 0 & 0 & 2G_1 \\ 0 & 1 & 1 & 0 & 0 & 2G_2 \\ 0 & 0 & 1 & 1 & 0 & 2G_3 \\ 0 & 0 & 0 & 1 & 1 & 2G_4 \\ 0 & 0 & 0 & 0 & 1 & G_5 + \cdots \end{array}\right] \tag{13}$$

$$\tag{14}$$

We discover that we end up with an upper triangular matrix, yielding a unique solution. Notice here the symmetry between the plates does not exist here. This means that every combination of plates and tips is distinguishable from the other.

1.c

We can induce that any odd number of tippers does not yield a unique solution because we can iterate the row operations such that we end up with a row of zeroes, whereas with the odd case there is 1 less column, so we are left with 1 pivot at the end of the row operations, yielding a unique solution.

2 Show It!

2.a

Proof. Suppose the matrix multiplication

$$\mathbf{Ax} = \mathbf{b} \tag{15}$$

where $\mathbf{A}$ is a transformation matrix upon a vector $\mathbf{x}$ creating the resultant $\mathbf{b}$. Then, an infinite solution set suggests

$$\mathbf{A}\mathbf{x}_1 = \mathbf{A}\mathbf{x}_2 = \cdots = \mathbf{A}\mathbf{x}_\mathrm{n} \tag{16}$$

where $\mathbf{x}_1 \neq \mathbf{x}_2 \neq \cdots \neq \mathbf{x}_n$. Moving the entire set to the other side,

$$\mathbf{A}\mathbf{x}_1 - \mathbf{A}\mathbf{x}_2 - \cdots - \mathbf{A}\mathbf{x}_\mathrm{n} = 0 \tag{17}$$
$$\mathbf{A}(\mathbf{x}_1 - \mathbf{x}_2 - \cdots - \mathbf{x}_\mathrm{n}) = 0 \tag{18}$$

Letting $\mathbf{v}$ represent the above vector,

$$\mathbf{A}\mathbf{v} = 0 \tag{19}$$
$$\begin{bmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n\end{bmatrix}\begin{bmatrix}\mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n\end{bmatrix} = 0 \tag{20}$$
$$\mathbf{a}_1\mathbf{v}_1 + \mathbf{a}_2\mathbf{v}_2 + \cdots + \mathbf{a}_n\mathbf{v}_n = 0 \tag{21}$$

where $\mathbf{a}$ is a column vector. By definition of linear dependence, $\mathbf{a}$ cannot all be zero, as that would mean $\mathbf{A}$ is the zero matrix. This means that the columns of $\mathbf{A}$ are linearly dependent if there is an infinite set of solutions. □

2.b

Proof. Given the set of vectors $\mathcal{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$, the span can be represented as

$$\mathrm{span}(\mathcal{V}) = \left\{ \sum_{i=1}^{n} c_i\mathbf{v}_i \,\middle|\, c \in \mathbb{R} \right\} = \{c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n | c \in \mathbb{R}\} \tag{22}$$

Let $\mathcal{V}' = \{\mathbf{v}_1 + \mathbf{v}_2, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$. Then, its span is

$$\mathrm{span}(\mathcal{V}') = \{c_1(\mathbf{v}_1 + \mathbf{v}_2) + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n | c \in \mathbb{R}\} \tag{23}$$
$$\mathrm{span}(\mathcal{V}') = \{c_1\mathbf{v}_1 + c_1\mathbf{v}_2 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n | c \in \mathbb{R}\} \tag{24}$$
$$\mathrm{span}(\mathcal{V}') = \{c_1\mathbf{v}_1 + (c_1 + c_2)\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n | c \in \mathbb{R}\} \tag{25}$$

Since c is arbitrary, we can simply reassign $c_1 + c_2$ to any convenient scalar. This means that the span of the set is still preserved and $\mathrm{span}(\mathcal{V}) = \mathrm{span}(\mathcal{V}')$. □

2.c

Proof. Given a linearly dependent set of vectors, by definition they satisfy the following property:

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n = 0 \tag{26}$$

where not all $c = 0$. If we apply a transformation matrix $\mathbf{A}$ to the set, then we get

$$\mathbf{A}(c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n) = \mathbf{A}0 \tag{27}$$
$$\mathbf{A}c_1\mathbf{v}_1 + \mathbf{A}c_2\mathbf{v}_2 + \cdots + \mathbf{A}c_n\mathbf{v}_n = 0 \tag{28}$$
$$c_1\mathbf{A}\mathbf{v}_1 + c_2\mathbf{A}\mathbf{v}_2 + \cdots + c_n\mathbf{A}\mathbf{v}_n = 0 \tag{29}$$

We can justify the last step because scalar multiplication is preserved under a matrix transformation. By definition, the transformed vectors satisfy linear dependence. □

3 Quadracopter Tranformations

3.a

Making the rotation matrix $\mathbf{R}_a$ simply involves multiplying the two matrices

$$\mathbf{R}_a = \begin{bmatrix} \cos(\pi/3) & -\sin(\pi/3) & 0 \\ \sin(\pi/3) & \cos(\pi/3) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\pi/6) & -\sin(\pi/6) \\ 0 & \sin(\pi/6) & \cos(\pi/6) \end{bmatrix} \tag{30}$$

$$= \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ 0 & \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \tag{31}$$

$$= \begin{bmatrix} \frac{1}{2}(1) - \frac{\sqrt{3}}{2}(0) + 0(0) & \frac{1}{2}(0) - \frac{\sqrt{3}}{2}\left(\frac{\sqrt{3}}{2}\right) + 0\left(\frac{1}{2}\right) & \frac{1}{2}(0) - \frac{\sqrt{3}}{2}\left(-\frac{1}{2}\right) + 0\left(\frac{\sqrt{3}}{2}\right) \\ \frac{\sqrt{3}}{2}(1) + \frac{1}{2}(0) + 0(0) & \frac{\sqrt{3}}{2}(0) + \frac{1}{2}\left(\frac{\sqrt{3}}{2}\right) + 0\left(\frac{1}{2}\right) & \frac{\sqrt{3}}{2}(0) + \frac{1}{2}\left(-\frac{1}{2}\right) + 0\left(\frac{\sqrt{3}}{2}\right) \\ 0(1) + 0(0) + 1(0) & 0(0) + 0\left(\frac{\sqrt{3}}{2}\right) + 1\left(\frac{1}{2}\right) & 0(0) + 0\left(-\frac{1}{2}\right) + 1\left(\frac{\sqrt{3}}{2}\right) \end{bmatrix} \tag{32}$$

$$= \begin{bmatrix} \frac{1}{2} & -\frac{3}{4} & \frac{\sqrt{3}}{4} \\ \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{4} & -\frac{1}{4} \\ 0 & \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \tag{33}$$

3.b

Finding the reverse multiplication $\mathbf{R}_b$,

$$\mathbf{R}_b = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ 0 & \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ \frac{3}{4} & \frac{\sqrt{3}}{4} & \frac{1}{2} \\ \frac{\sqrt{3}}{4} & \frac{1}{4} & \frac{\sqrt{3}}{2} \end{bmatrix} \tag{34}$$

3.c

Multiplying $\mathbf{r} = \begin{bmatrix}1\\1\\1\end{bmatrix}$ by R_a and R_b,

$$\mathbf{R}_a\mathbf{r} = \begin{bmatrix} \frac{1}{2} & -\frac{3}{4} & \frac{\sqrt{3}}{4} \\ \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{4} & -\frac{1}{4} \\ 0 & \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix}1\\1\\1\end{bmatrix} \tag{35}$$

$$= \begin{bmatrix} \frac{1}{2} - \frac{3}{4} + \frac{\sqrt{3}}{4} \\ \frac{\sqrt{3}}{2} + \frac{\sqrt{3}}{4} - \frac{1}{4} \\ 0 + \frac{1}{2} + \frac{\sqrt{3}}{2} \end{bmatrix} \tag{36}$$

$$= \begin{bmatrix} \frac{-1+\sqrt{3}}{4} \\ \frac{3\sqrt{3}-1}{4} \\ \frac{1+\sqrt{3}}{2} \end{bmatrix} \tag{37}$$

$$\mathbf{R}_b\mathbf{r} = \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ \frac{3}{4} & \frac{\sqrt{3}}{4} & \frac{1}{2} \\ \frac{\sqrt{3}}{4} & \frac{1}{4} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix}1\\1\\1\end{bmatrix} \tag{38}$$

$$= \begin{bmatrix} \frac{1}{2} - \frac{\sqrt{3}}{2} + 0 \\ \frac{3}{4} + \frac{\sqrt{3}}{4} + \frac{1}{2} \\ \frac{\sqrt{3}}{4} + \frac{1}{4} + \frac{\sqrt{3}}{2} \end{bmatrix} \tag{39}$$

$$= \begin{bmatrix} \frac{1-\sqrt{3}}{2} \\ \frac{5+\sqrt{3}}{4} \\ \frac{1+3\sqrt{3}}{4} \end{bmatrix} \tag{40}$$

As is demonstrated clearly, the two resultant vectors are different.

3.d

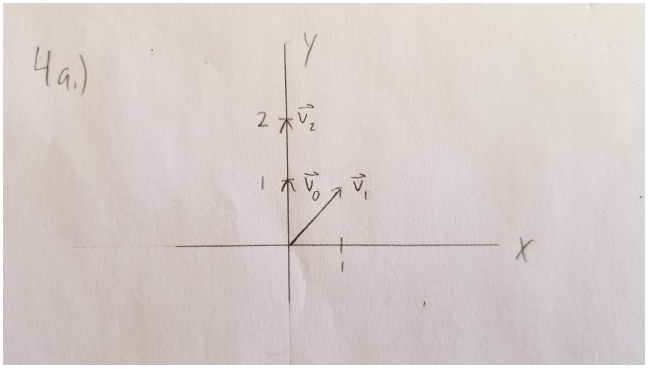
Given a displacement vector $\mathbf{r}$ with magnitude 1 and rotation matrices $\mathbf{R}_x(\theta), \mathbf{R}_y(\psi), \mathbf{R}_z(\phi)$, $||\mathbf{s}|| = 1$. Intuitively, this makes sense, as rotation of any vector in space does not actually scale the vector or change its magnitude as a whole.

4 Image Stitching

4.a

Plugging in our vectors,

$$\mathbf{v}_2 = \begin{bmatrix} 2 & 2 \\ -2 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix} \tag{41}$$



4.b

$$\begin{bmatrix} q_x \\ q_y \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{xx} & \mathbf{R}_{xy} \\ \mathbf{R}_{yx} & \mathbf{R}_{yy} \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix} \tag{42}$$

$$q_x = \mathbf{R}_{xx} p_x + \mathbf{R}_{xy} p_y + T_x \tag{43}$$

$$q_y = \mathbf{R}_{yx} p_x + \mathbf{R}_{yy} p_y + T_y \tag{44}$$

Here, $\mathbf{q}, \mathbf{v}$ are known, while $\mathbf{R}, \mathbf{T}$ are unknown. This means there are 6 unknown values with 2 equations. At minimum, 6 equations are required to find a unique solution. Since each point has 2 coordinates (i.e. p, q), a total of 6 equations is required at minimum for a unique solutions, with 3 pairs of points.

4.c

If we take 3 measurements, we obtain the following system of equations:

$$q_{x1} = \mathbf{R}_{xx} p_{x1} + \mathbf{R}_{xy} p_{y1} + T_x \tag{45}$$

$$q_{y1} = \mathbf{R}_{yx} p_{x1} + \mathbf{R}_{yy} p_{y1} + T_y \tag{46}$$

$$q_{x2} = \mathbf{R}_{xx} p_{x2} + \mathbf{R}_{xy} p_{y2} + T_x \tag{47}$$

$$q_{y2} = \mathbf{R}_{yx} p_{x2} + \mathbf{R}_{yy} p_{y2} + T_y \tag{48}$$

$$q_{x3} = \mathbf{R}_{xx} p_{x3} + \mathbf{R}_{xy} p_{y3} + T_x \tag{49}$$

$$q_{y3} = \mathbf{R}_{yx} p_{x3} + \mathbf{R}_{yy} p_{y3} + T_y \tag{50}$$

Our vector of unknowns is

$$\begin{bmatrix} \mathbf{R}_{xx} \\ \mathbf{R}_{xy} \\ \mathbf{R}_{yx} \\ \mathbf{R}_{yy} \\ T_x \\ T_y \end{bmatrix} \tag{51}$$

With this information, we can convert the above set of measurements into a system of equations:

$$\begin{bmatrix} p_{x1} & p_{y1} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{x1} & p_{y1} & 0 & 1 \\ p_{x2} & p_{y2} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{x2} & p_{y2} & 0 & 1 \\ p_{x3} & p_{y3} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{x3} & p_{y3} & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{xx} \\ \mathbf{R}_{xy} \\ \mathbf{R}_{yx} \\ \mathbf{R}_{yy} \\ T_x \\ T_y \end{bmatrix} = \begin{bmatrix} q_{x1} \\ q_{y1} \\ q_{x2} \\ q_{y2} \\ q_{x3} \\ q_{y3} \end{bmatrix} \tag{52}$$

4.d

See `iPython` notebook.

4.e

Since the points are collinear, that means that the values of subsequent points are linearly dependent. As we have proven, any matrix with linearly dependent columns does not have a unique solution. As indicated in the `iPython` notebook, the cell returns a similar error.

5 Properties of Pump Systems

5.a

Given the pump system as given, we can write the state steps as

$$x_a[n] + x_b[n] = x_a[n+1] \tag{53}$$
$$0 = x_b[n+1] \tag{54}$$

5.b

The transition matrix for the system is

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \tag{55}$$

5.c

Computing the results of both initial conditions,

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = 0.5 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 0.5 \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{56}$$
$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix} = 0.3 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 0.7 \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{57}$$

5.d

You *cannot* recover $x[0]$ because there are two initial conditions for which $x[0]$ lead to the same $x[1]$, making the question inconclusive.

5.e

You cannot use the resultant state of the pumps to recover the initial state. This is because since effectively all the water goes into pump A, meaning that as long as the initial condition satisfies the condition that they sum to the resultant vector, it is a valid starting condition. What this tells us is that $\mathbf{A}$ is *uninvertible* and does not yield a unique solution.

5.f

The transition matrix for the system shown is

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0.4 & 0.5 & 0.2 \\ 0 & 0.6 & 0.35 \end{bmatrix} \tag{58}$$

The sum of the column entries is $0.4, 1.1, 0.55$. If we subtract each pump's inputs from its outputs, we can find the net change per timestep:

$$\Delta P_1 = -0.4 \tag{59}$$
$$\Delta P_2 = 0 \tag{60}$$
$$\Delta P_3 = 0.4 \tag{61}$$

Summing up the net changes tells us that the net change in the entire system is 0, telling us that this transition matrix is *conservative*.

6 Homework Process and Study Group

I did this homework by myself.

prob3

September 20, 2019

1 EECS16A: Homework 3

1.1 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 HW3 zip file.
Run the next block of code before proceeding

[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

def solve(A,b):
    try:
        z = np.linalg.solve(A,b)
    except:
        raise ValueError('Rows are not linearly independent. Cannot solve␣
 ↪system of linear equations uniquely. :)')
    return z
```

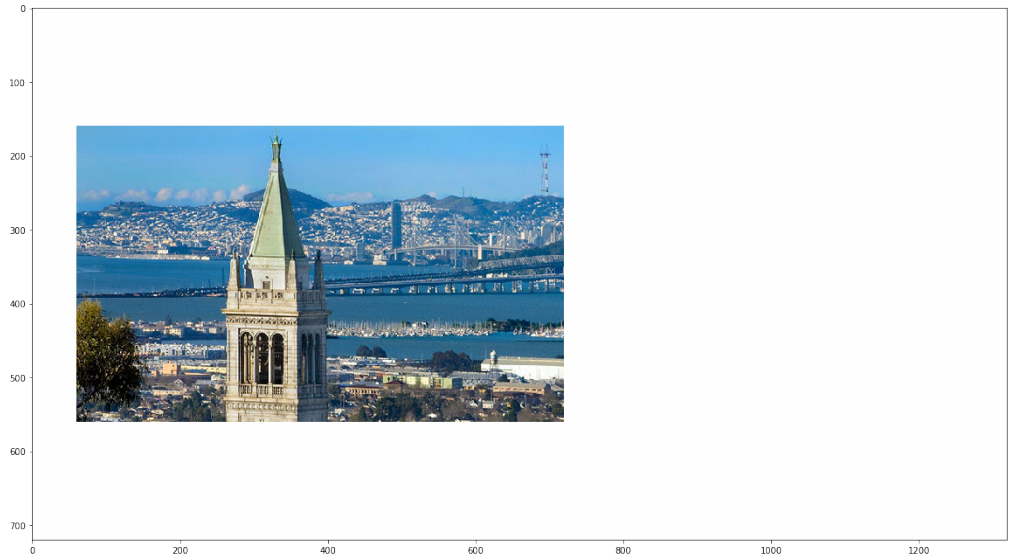
We will stick to a simple example and just consider stitching two images (if you can stitch two pictures, then you could conceivably stitch more by applying the same technique over and over again).

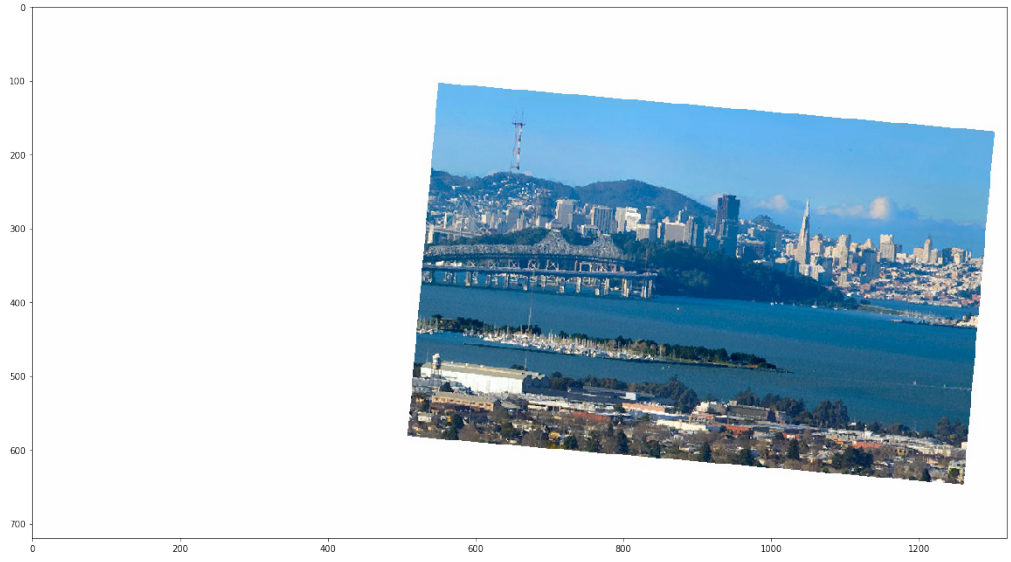
Daniel decided to take an amazing picture of the Campanile overlooking the bay. Unfortunately, the field of view of his camera was not large enough to capture the entire scene, so he decided to take two pictures and stitch them together.

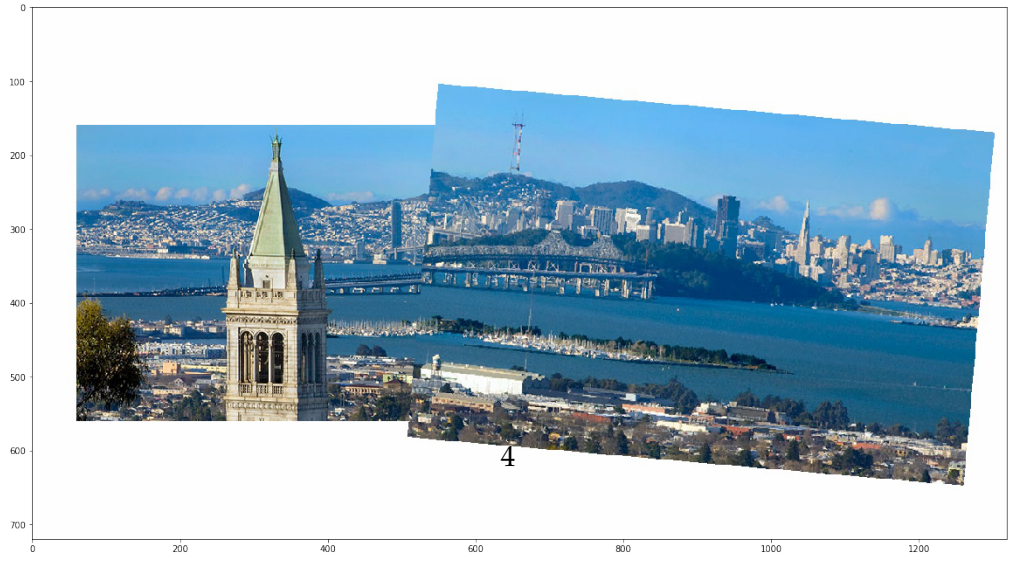
The next block will display the two images.

[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()
```


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

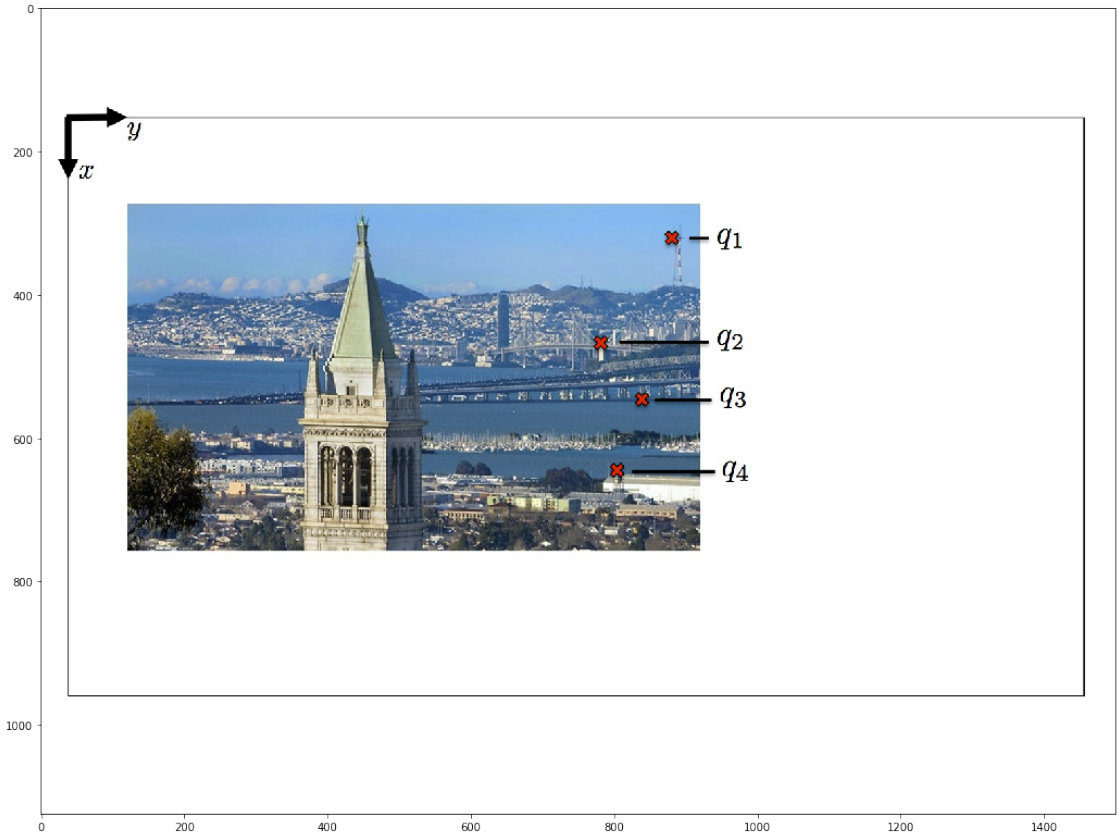

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

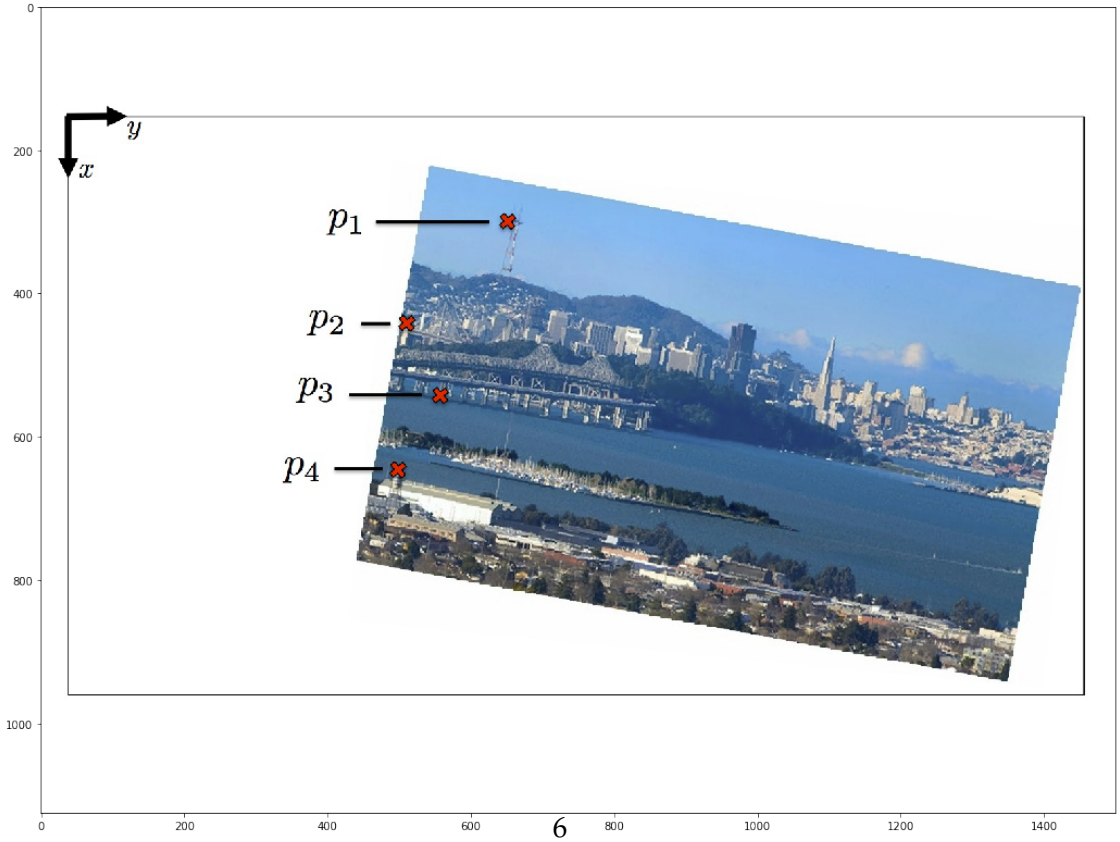

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

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

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

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

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

[3]:
```
<matplotlib.image.AxesImage at 0x7f5abf2d3cc0>
```


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


y
x
p_1
p_2
p_3
p_4
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

$$\vec{p}_1 = \begin{bmatrix} 200 \\ 700 \end{bmatrix} \qquad \vec{p}_2 = \begin{bmatrix} 310 \\ 620 \end{bmatrix} \qquad \vec{p}_3 = \begin{bmatrix} 390 \\ 660 \end{bmatrix} \qquad \vec{p}_4 = \begin{bmatrix} 460 \\ 630 \end{bmatrix} \tag{1}$$

$$\vec{q}_1 = \begin{bmatrix} 162.2976 \\ 565.8862 \end{bmatrix} \qquad \vec{q}_2 = \begin{bmatrix} 285.4283 \\ 458.7469 \end{bmatrix} \qquad \vec{q}_3 = \begin{bmatrix} 385.2465 \\ 498.1973 \end{bmatrix} \qquad \vec{q}_4 = \begin{bmatrix} 465.7892 \\ 455.0132 \end{bmatrix} \tag{2}$$

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.

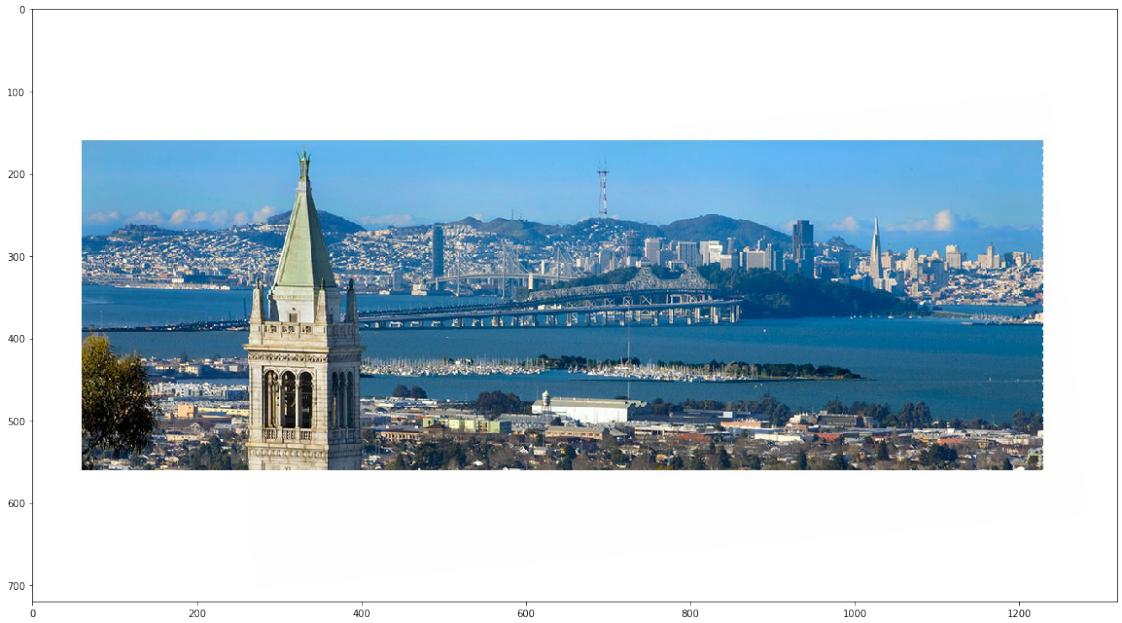
[4]:
```
# Note that the following is a general template for solving for 6 unknowns from
 ↪6 equations represented as Az = b.
# You do not have to use the following code exactly.
# All you need to do is to find parameters R_11, R_12, R_21, R_22, T_x, T_y.
# If you prefer finding them another way it is 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],
              [0,0,310,620,0,1],
              [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.

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

[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␣
 ↪image1[row,col,2] > 0.995:
            temp =␣
 ↪euclidean_transform_2to1(matrix_transform,translation,image2,position,LL,UL)
            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).
```



1.1.2 Part E: Failure Mode Points

$$\vec{p_1} = \begin{bmatrix} 390 \\ 660 \end{bmatrix} \qquad \vec{p_2} = \begin{bmatrix} 425 \\ 645 \end{bmatrix} \qquad \vec{p_3} = \begin{bmatrix} 460 \\ 630 \end{bmatrix} \tag{3}$$

$$\vec{q_1} = \begin{bmatrix} 385 \\ 450 \end{bmatrix} \qquad \vec{q_2} = \begin{bmatrix} 425 \\ 480 \end{bmatrix} \qquad \vec{q_3} = \begin{bmatrix} 465 \\ 510 \end{bmatrix} \tag{4}$$

[5]:
```
# Note that the following is a general template for solving for 6 unknowns 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([[390,660,0,0,1,0],
              [0,0,390,660,0,1],
              [425,645,0,0,1,0],
              [0,0,425,645,0,1],
              [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 call␣
↪last)

       <ipython-input-1-4ee2c81e8dee> in solve(A, b)
        51     try:
   ---> 52         z = np.linalg.solve(A,b)
        53     except:


       ~/anaconda3/lib/python3.7/site-packages/numpy/linalg/linalg.py in␣
↪solve(a, b)
       402     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
   --> 403     r = gufunc(a, b, signature=signature, extobj=extobj)
       404


       ~/anaconda3/lib/python3.7/site-packages/numpy/linalg/linalg.py in␣
↪_raise_linalgerror_singular(err, flag)
        96 def _raise_linalgerror_singular(err, flag):
   ---> 97     raise LinAlgError("Singular matrix")
        98


       LinAlgError: Singular matrix


   During handling of the above exception, another exception occurred:


       ValueError                                Traceback (most recent call␣
↪last)
```

```
      <ipython-input-5-48bdfaeadea9> in <module>
       19
       20 # solve the linear system for the coefficiens
  ---> 21 z = solve(A,b)
       22
       23 #Parameters for our transformation


      <ipython-input-1-4ee2c81e8dee> 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 of␣
↪linear equations uniquely. :)
```