Homework #2

1. Mechanical Inverses

a) $\begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix} \Rightarrow \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{array}\right] \Rightarrow \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & \frac{1}{2} \end{array}\right] \Rightarrow \left[\begin{array}{cc|cc} 0 & 1 & 1 & -\frac{1}{2} \\ 1 & 0 & 0 & \frac{1}{2} \end{array}\right] \Rightarrow \left[\begin{array}{cc|cc} 1 & 0 & 0 & \frac{1}{2} \\ 0 & 1 & 1 & -\frac{1}{2} \end{array}\right]$

$\text{Inverse} = \begin{bmatrix} 0 & \frac{1}{2} \\ 1 & -\frac{1}{2} \end{bmatrix}$

b) $\begin{bmatrix} 3 & 2 \\ 1 & -1 \end{bmatrix} \Rightarrow \left[\begin{array}{cc|cc} 3 & 2 & 1 & 0 \\ 1 & -1 & 0 & 1 \end{array}\right] \Rightarrow \left[\begin{array}{cc|cc} \sqrt{} & 0 & 1 & 2 \\ -1 & 1 & 0 & -1 \end{array}\right] \Rightarrow \left[\begin{array}{cc|cc} 1 & 0 & \frac{1}{\sqrt{}} & \frac{2}{\sqrt{}} \\ -1 & 1 & 0 & -1 \end{array}\right] \Rightarrow \left[\begin{array}{cc|cc} 1 & 0 & \frac{1}{\sqrt{}} & \frac{2}{\sqrt{}} \\ 0 & 1 & \frac{1}{\sqrt{}} & -\frac{3}{\sqrt{}} \end{array}\right]$

$\text{Inverse} = \begin{bmatrix} \frac{1}{\sqrt{}} & \frac{2}{\sqrt{}} \\ \frac{1}{\sqrt{}} & -\frac{3}{\sqrt{}} \end{bmatrix}$

c) $\begin{bmatrix} -\frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & -\frac{1}{2} \end{bmatrix} \Rightarrow \left[\begin{array}{cc|cc} -\frac{1}{2} & -\frac{\sqrt{3}}{2} & 1 & 0 \\ \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 & 1 \end{array}\right] \Rightarrow \left[\begin{array}{cc|cc} \sqrt{3} & 3 & -2\sqrt{3} & 0 \\ \sqrt{3} & -1 & 0 & 2 \end{array}\right] \Rightarrow \left[\begin{array}{cc|cc} 1 & \sqrt{3} & -2 & 0 \\ 0 & -\sqrt{3} & \frac{3}{2} & \frac{\sqrt{3}}{2} \end{array}\right] \Rightarrow \left[\begin{array}{cc|cc} 1 & 0 & -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ 0 & 1 & -\frac{\sqrt{3}}{2} & -\frac{1}{2} \end{array}\right]$

$\text{Inverse} = \begin{bmatrix} -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{\sqrt{3}}{2} & -\frac{1}{2} \end{bmatrix}$

d) $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \Rightarrow \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 & 1 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 & 0 & 0 & \frac{1}{2} \end{array}\right]$

$\text{Inverse} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix}$

e) $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \Rightarrow \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right]$

$\text{Inverse} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$

h) $\begin{bmatrix} 3 & 0 & -2 & 1 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 0 & 4 \\ 1 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \left[\begin{array}{cccc|cccc} 3 & 0 & -2 & 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 3 & 0 & 1 & 0 & 0 \\ 3 & 1 & 0 & 4 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \Rightarrow \left[\begin{array}{cccc|cccc} 0 & 0 & -3 & -3 & \frac{3}{2} & 0 & 0 & -\frac{9}{2} \\ 0 & 0 & -3 & -3 & 2 & 1 & -2 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & -3 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \Rightarrow \left[\begin{array}{cccc|cccc} 0 & 0 & -3 & -3 & \frac{3}{2} & 0 & 0 & -\frac{9}{2} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 1 & -2 & \frac{9}{2} \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & -3 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$

From row 2, we can see that the left side cannot be transformed into identity matrix, which also means that there's not pivot in every column. Thus the matrix is not invertible.

2. Mechanical Eigenvalues.

$A=\begin{bmatrix}2&3\\3&4\end{bmatrix}$. $|A-\lambda I|=\begin{vmatrix}2-\lambda&3\\3&4-\lambda\end{vmatrix}=(2-\lambda)(4-\lambda)-9=\lambda^2-6\lambda-1=0$

$\lambda=\frac{6\pm\sqrt{36+4}}{2}=3\pm\sqrt{10}$

$\lambda_1=3+\sqrt{10}\Rightarrow A-\lambda I\Rightarrow\begin{bmatrix}-1-\sqrt{10}&3\\3&1-\sqrt{10}\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}0\\0\end{bmatrix}\Rightarrow\left[\begin{array}{cc|c}-1-\sqrt{10}&3&0\\3&1-\sqrt{10}&0\end{array}\right]$

$\left[\begin{array}{cc|c}1+\sqrt{10}&-3&0\\3&1-\sqrt{10}&0\end{array}\right]\Rightarrow\left[\begin{array}{cc|c}\frac{1+\sqrt{10}}{3}&-1&0\\\frac{1+\sqrt{10}}{3}&-1&0\end{array}\right]\Rightarrow\frac{1+\sqrt{10}}{3}x_1=x_2,\ \begin{bmatrix}x_1\\x_2\end{bmatrix}=x_1\begin{bmatrix}1\\\frac{1+\sqrt{10}}{3}\end{bmatrix}\Rightarrow E\lambda_1$

$\lambda_2=3-\sqrt{10}\Rightarrow A-\lambda I\Rightarrow\left[\begin{array}{cc|c}-1+\sqrt{10}&3&0\\3&1+\sqrt{10}&0\end{array}\right]\Rightarrow\left[\begin{array}{cc|c}\sqrt{10}-1&3&0\\\sqrt{10}-1&3&0\end{array}\right]\Rightarrow\frac{1-\sqrt{10}}{3}x_1=x_2\Rightarrow\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}1\\\frac{1-\sqrt{10}}{3}\end{bmatrix}x_1\Rightarrow E\lambda_2$

3. Four Fundamental Subspaces.

$A=\begin{bmatrix}1&-1&-3&4\\3&-3&-5&8\\1&-1&-1&2\end{bmatrix}\Rightarrow\begin{bmatrix}1&-1&-3&4\\0&0&4&-4\\0&0&2&-2\end{bmatrix}\Rightarrow\begin{bmatrix}1&-1&-3&4\\0&0&1&-1\\0&0&0&0\end{bmatrix}$ (row echelon form)

Thus. $Rank(A)=2$.

a) $Col(A)\Rightarrow\left\{\begin{bmatrix}1\\3\\1\end{bmatrix}\begin{bmatrix}-3\\-5\\-1\end{bmatrix}\right\}$. dimension $=2$

b) $\begin{cases}x_1=x_2+3x_3-4x_4\\x_3=x_4\end{cases}\Rightarrow\begin{cases}x_1=x_2-x_3\\x_3=x_4\end{cases}$. $\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}=\begin{bmatrix}x_1\\x_1+x_3\\x_3\\x_3\end{bmatrix}=x_1\begin{bmatrix}1\\1\\0\\0\end{bmatrix}+x_3\begin{bmatrix}0\\1\\1\\1\end{bmatrix}$. $\dim(Null(A))=2$

$Null(A)=\left\{\begin{bmatrix}1\\1\\0\\0\end{bmatrix}\begin{bmatrix}0\\1\\1\\1\end{bmatrix}\right\}$

c) $Row(A)=\left\{\begin{bmatrix}1\\-1\\-3\\4\end{bmatrix}\begin{bmatrix}0\\0\\1\\-1\end{bmatrix}\right\}$ $\dim(Row(A))=2$

d) $A^T=\begin{bmatrix}1&3&1\\-1&-3&-1\\-3&-5&-1\\4&8&2\end{bmatrix}$ $A^T_{rref}=\begin{bmatrix}1&3&1\\0&0&0\\0&4&2\\0&-4&-2\end{bmatrix}\Rightarrow\begin{bmatrix}1&3&1\\0&2&1\\0&0&0\\0&0&0\end{bmatrix}\Rightarrow\begin{bmatrix}1&0&-\frac{1}{2}\\0&1&\frac{1}{2}\\0&0&0\\0&0&0\end{bmatrix}\Rightarrow\begin{cases}x_1=\frac{1}{2}x_3\\x_2=-\frac{1}{2}x_3\end{cases}\Rightarrow\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}=\begin{bmatrix}\frac{1}{2}x_3\\-\frac{1}{2}x_3\\x_3\end{bmatrix}=\begin{bmatrix}1\\-1\\2\end{bmatrix}x_3$

$Left\text{-}Null(A)=\left\{\begin{bmatrix}1\\-1\\2\end{bmatrix}\right\}$

4. Transition Matrix Proofs

Original $\Rightarrow \begin{bmatrix} 0 & 0 & 0 \\ b & a & 0 \\ c & 0 & 0 \end{bmatrix}$ (O) Reversed $\Rightarrow \begin{bmatrix} 0 & b & c \\ 0 & a & 0 \\ 0 & 0 & 0 \end{bmatrix}$ (R), Thus Reversed = Original^T

a) To make $O\cdot\vec{x}=\vec{y}$, and uniquely $R\cdot\vec{y}=\vec{x}$. $R\cdot O\cdot\vec{x}=R\cdot\vec{y}=\vec{x} \Rightarrow R\cdot O=I$
Thus, to find same scale vector at time n-1. Original matrix and Reversal matrix must be reversal to each other. Thus O/R must be invertible. However, both O and R only has two pivots, thus they don't have pivot in every column/row, thus, O/R are not invertible and thus, we cannot find scale vector at time n-1.

b) It means that the total amount of people in the network of nabbles remains the same

c) $A=\begin{bmatrix} 0 & 0 & 0 \\ 0.5 & 0.4 & 0.2 \\ 0 & 0.6 & 0.65 \end{bmatrix}$ since the entries in each column vector doesn't sum to 1 (actually less) the total amount of people in system decrease

d) since a uniform vector means $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ where $x_1=x_2=\cdots=x_n$, $\Rightarrow \begin{bmatrix} x \\ x \\ \vdots \\ x \end{bmatrix} \Rightarrow \vec{x}$
let $A \Rightarrow \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & a_{ij} & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$ where entries of each row sums to 1.

Thus $A\cdot\vec{x}=\vec{y} \Rightarrow y_{n1} = \overbrace{[a_{n1}\cdots a_{nn}]}^{\text{nth row in A}}\begin{bmatrix} x \\ \vdots \\ x \end{bmatrix} = x\overbrace{(a_{n1}+a_{n+}-a_{nn})}^{\text{entries of a row sum to 1}} = x\times 1 = x$

y_{n1}: element of y in nth row; $\begin{bmatrix} x \\ \vdots \\ x \end{bmatrix}$: uniform vector.

Thus elements of y are all equals to x. thus $\vec{y}=\begin{bmatrix} x \\ x \\ \vdots \\ x \end{bmatrix}$ = the uniform vector

5. Pumps Properties Proofs.

a) $A = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$

b) Since $A\begin{bmatrix} \pi_1[0] \\ \pi_2[0] \end{bmatrix} = \begin{bmatrix} \pi_1[1] \\ \pi_2[1] \end{bmatrix}$ ① $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ ② $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, thus for both initials. $\vec{\pi}[1] = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

c) No. an counterexample is in question b) that for an $\vec{\pi}[1] = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, we cannot come up an unique initial $\vec{\pi}[0]$ since both $\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$ and $\begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix}$ are possible. The transition matrix A must be invertible to make the "take back" vector unique. but $A = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$ is not since it doesn't have pivot in every column/row.

d) No. Suppose both $A\vec{\pi}_1[0] = \vec{\pi}[1]$ and $A\vec{\pi}_2[0] = \vec{\pi}[1]$, ($\vec{\pi}_1[0] \neq \vec{\pi}_2[0]$) thus by definition. if we want to find an unique answer for timestep[0] base on timestep[1], then timestep[0] = A^{-1} timestep[1] thus there should be unique $\vec{\pi}[0]$ that $A^{-1}\vec{\pi}[1] = \vec{\pi}[0]$. However. by given information. $\vec{\pi}_1[0] = A^{-1}\vec{\pi}[1] = \vec{\pi}_2[0]$. which is not right since $\vec{\pi}_1[0] \neq \vec{\pi}_2[0]$. thus. A is not invertible. and thus. there's not one way recover the state vector at previous timestep.

e) Since every initial state has unique state vector in the next timestep. we know that for every vector in Range(A) has unique $\vec{\pi}$. thus A has pivot in every column/row. and thus full rank. Thus A is invertible. Thus for every state vector we can find its unique state vector from previous timestep.

6. Image Stitching

a) $\begin{cases} q_{1x} = R_{xx}p_x + R_{xy}p_y + T_x \\ q_{1y} = R_{yx}p_x + R_{yy}p_y + T_y \end{cases}$. Unknowns are R_{xx}. R_{xy}. R_{yx}. R_{yy}. T_x. T_y

There are 6 unknowns. Thus we need 6 equations.

For every pair of pixels we can generate 2 equations. Thus we need 3 pairs of pixels

b) Let $\vec{q}_i = \begin{bmatrix} q_{ix} \\ q_{iy} \end{bmatrix}$ $\vec{p}_i = \begin{bmatrix} p_{ix} \\ p_{iy} \end{bmatrix}$. same for $q_1 \sim q_3$. $p_1 \sim p_3$.

Thus the linear equations are

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

c) $R \approx \begin{bmatrix} 1.1954 & 0.1046 \\ -0.1046 & 1.1954 \end{bmatrix}$. $T \approx \begin{bmatrix} -150 \\ -250 \end{bmatrix}$

d) If $(\vec{q}_2 - \vec{q}_1) = k(\vec{p}_2 - \vec{p}_1)$. Then the system becomes.

$$\begin{bmatrix} p_{1x} & p_{1y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{1x} & p_{1y} & 0 & 1 \\ p_{2x}-p_{1x} & p_{2y}-p_{1y} & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{2x}-p_{1x} & p_{2y}-p_{1y} & 0 & 0 \\ p_{3x}-p_{1x} & p_{3y}-p_{1y} & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{3x}-p_{1x} & p_{3y}-p_{1y} & 0 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} p_{1x} & p_{1y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{1x} & p_{1y} & 0 & 1 \\ k(p_{2x}-p_{1x}) & k(p_{2y}-p_{1y}) & 0 & 0 & 0 & 0 \\ 0 & 0 & k(p_{2x}-p_{1x}) & k(p_{2y}-p_{1y}) & 0 & 0 \\ p_{3x}-p_{1x} & p_{3y}-p_{1y} & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{3x}-p_{1x} & p_{3y}-p_{1y} & 0 & 0 \end{bmatrix} \begin{matrix} \rightarrow \text{row1} \\ \rightarrow \text{row2} \\ \rightarrow \text{row3} \\ \rightarrow \text{row4} \\ \rightarrow \text{row5} \\ \rightarrow \text{row6} \end{matrix}$$

Thus. since (row3 and row5) & (row4 and row6) are linearly dependent. the linear system is underdetermined.

→ Geometrically. if we choose pixels along one line and set R to be a transformation that stretch to an perpendicular direction. the translation doesn't work. For example. if we set all the pixels along y-axis. and we set R to be a translation the stretch through x-axis. then nothing changes.

7. Cubic Polynomials.

a) $\vec{0}$ can be represented as $t=0$ & $p_0=0$

for some scalar α: $\alpha p(t) = \alpha p_0 + \alpha p_1 t + \alpha p_2 t^2 + \alpha p_3 t^3$ (also a cubic polynomial)

for another cubic polynomials $q(t) = q_0 + q_1 t + q_2 t^2 + q_3 t^3$:

$$p(t)+q(t) = p_0+q_0 + (p_1+q_1)t + (p_2+q_2)t^2 + (p_3+q_3)t^3$$ (also a cubic polynomial)

Thus $p(t) = p_0 + p_1 t + p_2 t^2 + p_3 t^3$, where $t \in [a,b]$ and p_n as scalars. forms a vector space.

The dimension of the vector space is 4. since there are four degrees of freedom.

b) $$\vec{c}^T \vec{\varphi}(t) = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 \end{bmatrix} \begin{bmatrix} \varphi_0(t) \\ \varphi_1(t) \\ \varphi_2(t) \\ \varphi_3(t) \end{bmatrix} = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix} = c_0 + c_1 t + c_2 t^2 + c_3 t^3$$

c) Comparing $p(t)$ and $\vec{c}^T \varphi(t)$. we can see that $p_k = c_k$. which means:

$p_0 = c_0$. $p_1 = c_1$ $p_2 = c_2$ $p_3 = c_3$

d) True. because the canonical polynomials $\varphi_k(t) = t^k$ are linearly independent and span the space of the polynomials.

8. Bieber's Segway

a) $\vec{x}[1] = A\vec{x}[0] + \vec{b}u[0]$

b) $\vec{x}[2] = A\vec{x}[1] + bu[1] = A(A\vec{x}[0] + \vec{b}u[0]) + \vec{b}u[1] = A^2\vec{x}[0] + A\vec{b}u[0] + \vec{b}u[1]$

$\vec{x}[3] = A(A^2\vec{x}[0] + A\vec{b}u[0] + \vec{b}u[1]) + \vec{b}u[2] = A^3\vec{x}[0] + A^2\vec{b}u[0] + A\vec{b}u[1] + \vec{b}u[2]$

$\vec{x}[4] = A^4\vec{x}[0] + A^3\vec{b}u[0] + A^2\vec{b}u[1] + A\vec{b}u[2] + \vec{b}u[3]$

c) $\vec{x}[n] = A^n\vec{x}[0] + \sum_{i=0}^{n-1} A^i\vec{b}u[n-1-i]$, where $A^0 = I$

d) No. since we want to get $\vec{x}_f$ in two steps. from $\vec{x}_0$. we get $\vec{x}[2]$

$\vec{x}[2] = A^2\vec{x}[0] + A\vec{b}u[0] + \vec{b}u[1] \Rightarrow \vec{x}[2] - A^2\vec{x}[0] = A\vec{b}u[0] + \vec{b}u[1] = [A\vec{b}\ \vec{b}]\begin{bmatrix} u[0] \\ u[1] \end{bmatrix} = \vec{x}_f - A^2\vec{x}[0]$

Since $\vec{x}_f = \vec{0}$, we can solve $\begin{bmatrix} u[0] \\ u[1] \end{bmatrix}$ by solving $[A\vec{b}\ \vec{b} \mid -A^2\vec{x}[0]]$

Use Ipydon we can calculate $A\vec{b}$ and $-A^2\vec{x}[0]$.

We find that the rank of the matrix $[A\vec{b}\ \vec{b}\ -A^2\vec{x}[0]]$ is 3. which means the row echelon of the system is $\left[\begin{array}{cc|c} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right]$ which is not consistent. thus there doesn't exist solution for $u[0]$ and $u[1]$.

e) No. $\vec{x}[3] = A^3\vec{x}[0] + A^2\vec{b}u[0] + A\vec{b}u[1] + \vec{b}u[2] \Rightarrow A^2\vec{b}u[0] + A\vec{b}u[1] + \vec{b}u[2] = \vec{x}_f - A^3\vec{x}[0] = -A^3\vec{x}[0]$

Thus $[A^2\vec{b}\ A\vec{b}\ \vec{b}]\begin{bmatrix} u[0] \\ u[1] \\ u[2] \end{bmatrix} = -A^3\vec{x}[0]$, Use Ipydon to calculate $A^2\vec{b}$. $A\vec{b}$ and $-A^3\vec{x}[0]$ to calculate the rank of the system matrix $[A^2\vec{b}\ A\vec{b}\ \vec{b}\ -A^3\vec{x}[0]]$. we get rank = 4

which means that the echelon form of the system is $\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right]$. thus the system is inconsistent and solutions for $\begin{bmatrix} u[0] \\ u[1] \\ u[2] \end{bmatrix}$ don't exist.

f) Yes. $\vec{x}[4] = A^4\vec{x}[0] + A^3\vec{b}u[0] + A^2\vec{b}u[1] + A\vec{b}u[2] + \vec{b}u[3]$.

$A^3\vec{b}u[0] + A^2\vec{b}u[1] + A\vec{b}u[2] + \vec{b}u[3] = \vec{x}[4] - A^4\vec{x}[0] = \vec{x_f} - A^4\vec{x}[0] = -A^4\vec{x}[0]$

Thus $\begin{bmatrix} A^3\vec{b} & A^2\vec{b} & A\vec{b} & \vec{b} \end{bmatrix}\begin{bmatrix} u[0] \\ u[1] \\ u[2] \\ u[3] \end{bmatrix} = \begin{bmatrix} -A^4\vec{x}[0] \end{bmatrix}$

Use Python to calculate $A^3\vec{b}$ and $-A^4\vec{x}[0]$ and calculate the rank of system $P = \begin{bmatrix} A^3\vec{b} & A^2\vec{b} & A\vec{b} & \vec{b} \end{bmatrix}$

we get rank = 4. Thus the system is consistent and by solving the system we get $\vec{u} = \begin{bmatrix} -13.2487502 \\ 23.7332412 \\ -11.4718872 \\ 1.46141773 \end{bmatrix} \begin{matrix} \to u[0] \\ \to u[1] \\ \to u[2] \\ \to u[3] \end{matrix}$

g) As displayed above. the steering are $\vec{u} = \begin{bmatrix} u[0] \\ u[1] \\ u[2] \\ u[3] \end{bmatrix} = \begin{bmatrix} -13.2487502 \\ 23.7332412 \\ -11.4718872 \\ 1.46141773 \end{bmatrix}$

h) Since $\vec{x}[n] = A^n\vec{x}[0] + \sum_{i=0}^{n-1} A^i\vec{b}u[n-1-i] \Rightarrow \sum_{i=0}^{n-1} A^i\vec{b}u[n-1-i] = \vec{x_f} - A^n\vec{x}[0] = -A^n\vec{x}[0]$

Thus $\begin{bmatrix} A^{n-1}\vec{b} & A^{n-2}\vec{b} & \cdots & A\vec{b} & \vec{b} \end{bmatrix}\begin{bmatrix} u[0] \\ u[1] \\ \vdots \\ u[n-1] \end{bmatrix} = -A^n\vec{x}[0]$

In order to ensure there are solutions $\vec{u} = \begin{bmatrix} u[0] \\ u[1] \\ u[n-1] \end{bmatrix}$ for the system. we need to make sure the the system $\left[A^{n-1}\vec{b}\ A^{n-2}\vec{b} \cdots A\vec{b}\ \vec{b} \middle| A^n\vec{x}[0]\right]$ is consistent. and ensure that $-A^n\vec{x}[0]$ is in the range of $\begin{bmatrix} A^{n-1}\vec{b} & A^{n-2}\vec{b} & \cdots & A\vec{b} & \vec{b} \end{bmatrix}$

i) Continue the previous demonstration. since $\vec{x_f}$ can be any vector in R^4.

$\begin{bmatrix} A^{n-1}\vec{b} & A^{n-2}\vec{b} & \cdots & A\vec{b} & \vec{b} \end{bmatrix}\begin{bmatrix} u[0] \\ u[1] \\ u[n-1] \end{bmatrix} = \begin{bmatrix} \vec{x_f} - A^n\vec{x}[0] \end{bmatrix} \Rightarrow$ thus $\vec{x_f} - A^n\vec{x}[0]$ can be any vector in R^4. thus call it matrix Q

Since $\vec{x_f} - A^n\vec{x}[0]$ has to be in range(Q). the range of Q must be all R^4. the column of Q must span R^4.

9. Homework Process and Study Group
Fansheng Cheng. 3033207855.
Cong Yang. 3032217722
Samuel Hanneschen 23804699
Wayne Li (me) 3032103452

10.

EE16A: Homework 2

Problem 6: 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

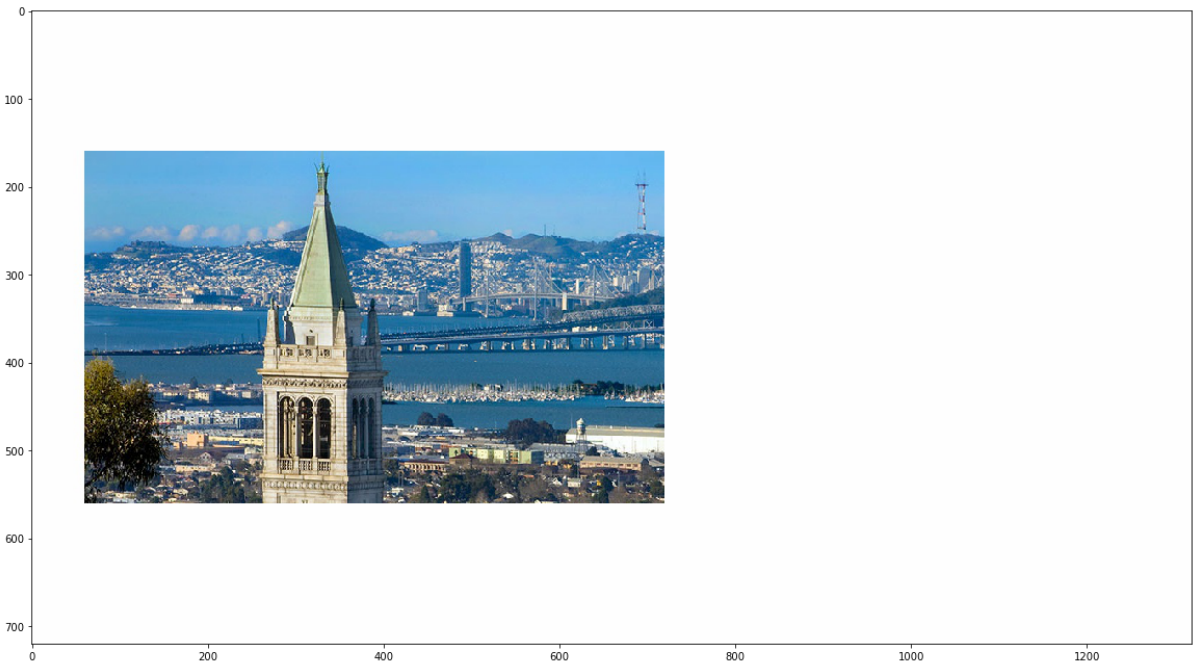
In [2]:
```
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from numpy import pi, cos, exp, sin
import matplotlib.image as mpimg
import matplotlib.transforms as mtransforms


%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).

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 stich them together.

The next block will display the two images.

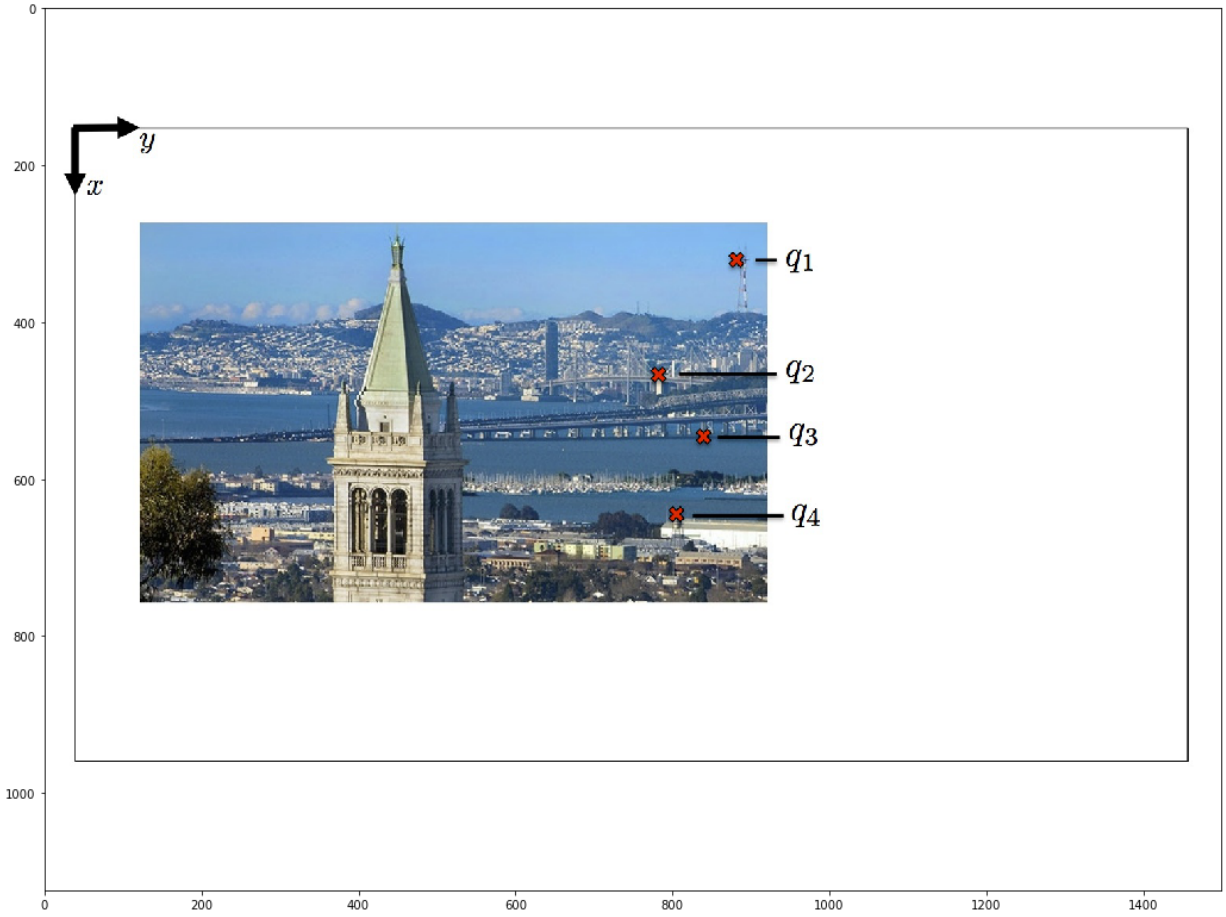
In [3]:
```
plt.figure(figsize=(20,40))

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

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

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

plt.show()
```

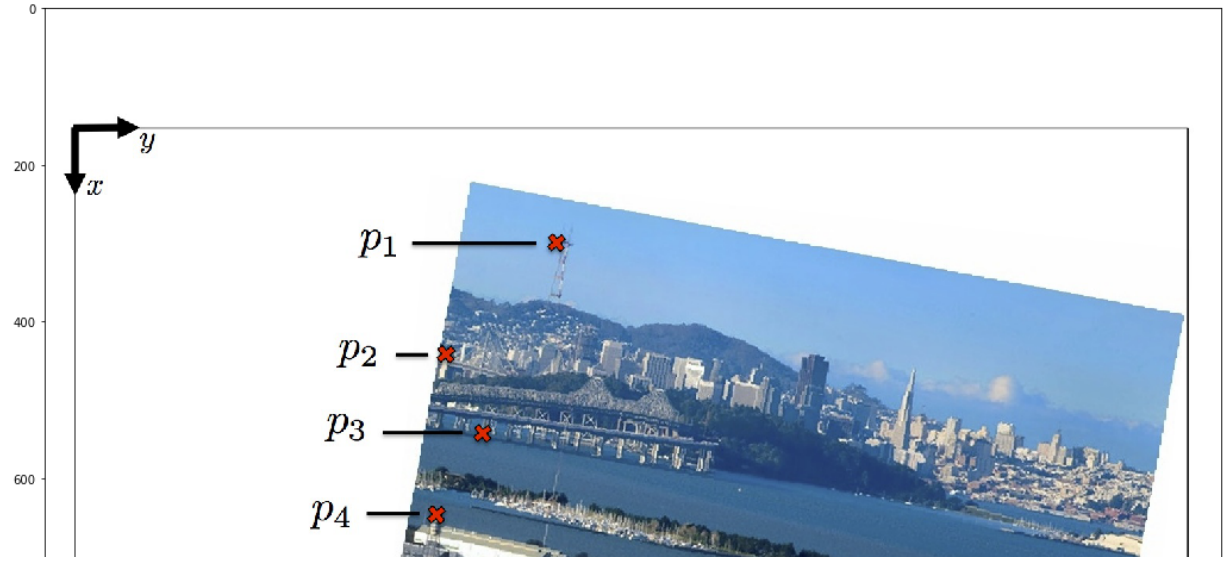


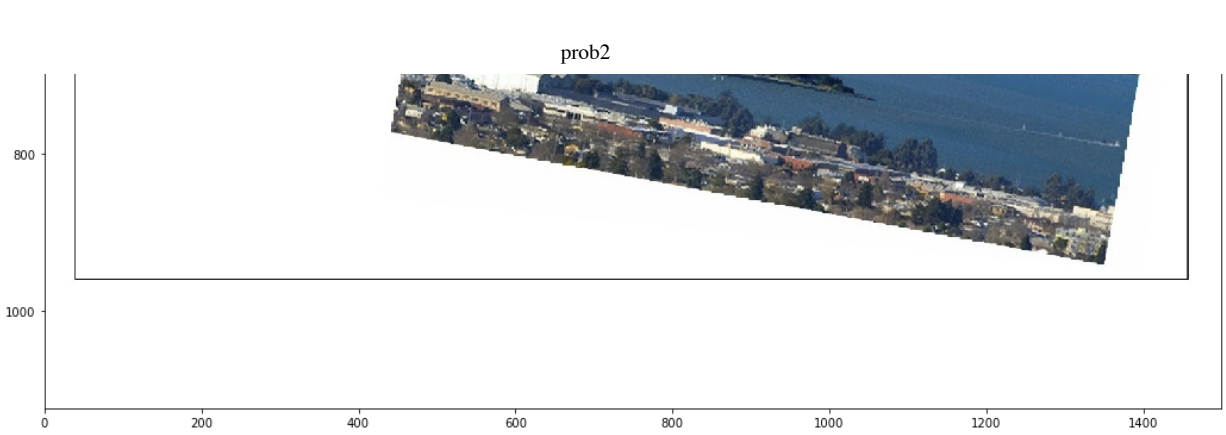
Once you apply Marcela's algorithm on the two images you get the following result (run the next block):

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 0x12c4df630>







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.

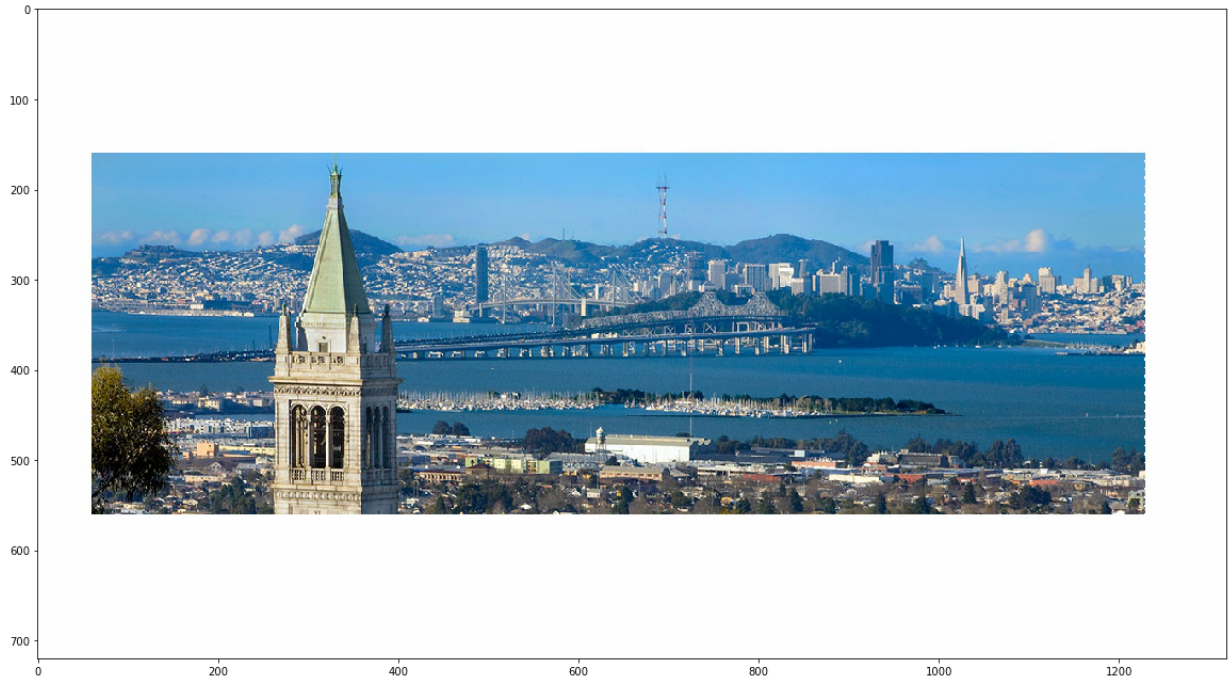
Using the points determine the parameters $R_{11}, R_{12}, R_{21}, R_{22}, T_x, T_y$ that map the points from the first image to the points in the second image by solving an appropriate system of equations. Hint: you do not need all the points to recover the parameters.

In [5]:
```
# 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],[385.2465],[498.19

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 [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 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()
```



Problem 8: Bieber's Segway

Run the following block of code first to get all the dependencies.

In [7]:
```
# %load gauss_elim.py
from gauss_elim import gauss_elim
```

In [8]:
```
from numpy import zeros, cos, sin, arange, around, hstack
from matplotlib import pyplot as plt
from matplotlib import animation
from matplotlib.patches import Rectangle
import numpy as np
from scipy.interpolate import interp1d
import scipy as sp
```

Dynamics

In [9]:
```
# Dynamics: state to state
A = np.array([[1, 0.05, -.01, 0],
              [0, 0.22, -.17, -.01],
              [0, 0.1, 1.14, 0.10],
              [0, 1.66, 2.85, 1.14]]);
# Control to state
b = np.array([.01, .21, -.03, -0.44])
nr_states = b.shape[0]

# Initial state
state0 = np.array([-0.3853493, 6.1032227, 0.8120005, -14])

# Final (terminal state)
stateFinal = np.array([0, 0, 0, 0])
```

Part (d), (e), (f)

In [10]:
```
# You may use gauss_elim to help you find the row reduced echelon form.
Ab = np.dot(A,b)
mAx = -np.dot((np.dot(A,A)),state0)
print(Ab)
print(mAx)
Matrix = ([[0.0208, 0.01, 0.02243475],
           [0.0557, 0.21, -0.30785117],
           [-0.0572, -0.03, 0.06193476],
           [-0.2385, -0.44, 1.38671326]])
np.linalg.matrix_rank(Matrix)
```

```
[ 0.0208  0.0557 -0.0572 -0.2385]
[ 0.02243475 -0.30785117  0.06193476  1.38671326]
```

Out[10]: 3

In [11]:
```
A2b = np.dot(np.dot(A,A),b)
A2 = np.dot(A,A)
A3 = np.dot(A2,A)
A3x = -np.dot(A3,state0)
print(A2b)
print(A3x)
Matrix2 = ([[0.024157, 0.0208, 0.01, 0.00642285],
            [0.024363, 0.0557, 0.21, -0.0921233],
            [-0.083488, -0.0572, -0.03, 0.17849184],
            [-0.342448, -0.2385, -0.44, 1.24633424]])
np.linalg.matrix_rank(Matrix2)
```

```
[ 0.024157  0.024363 -0.083488 -0.342448]
[ 0.00642285 -0.0921233   0.17849184  1.24633424]
```

Out[11]: 4

In [12]:
```
A3b = np.dot(A3,b)
A4 = np.dot(A3,A)
A4x = -np.dot(A4,state0)
print(A3b)
print(A4x)
Matrix3 = ([[0.02621003, 0.024157, 0.0208, 0.01],
            [0.0229773, 0.024363, 0.0557, 0.21],
            [-0.12698482, -0.083488, -0.0572, -0.03],
            [-0.58788894, -0.342448, -0.2385, -0.44]])
print(np.linalg.matrix_rank(Matrix3))
result = np.linalg.solve(Matrix3, A4x)
print(result)
```

```
[ 0.02621003  0.0229773  -0.12698482 -0.58788894]
[  3.17637529e-05  -6.30740802e-02   3.18901788e-01   1.77659810e+00]
4
[-13.24875075  23.73325125 -11.57181872   1.46515973]
```

Part (g)

Preamble

This function will take care of animating the segway.

In [13]:
```
# frames per second in simulation
fps = 20
# length of the segway arm/stick
stick_length = 1.

def animate_segway(t, states, controls, length):
    #Animates the segway

    # Set up the figure, the axis, and the plot elements we want to animate
    fig = plt.figure()

    # some config
    segway_width = 0.4
    segway_height = 0.2

    # x coordinate of the segway stick
    segwayStick_x = length * np.add(states[:, 0],sin(states[:, 2]))
    segwayStick_y = length * cos(states[:, 2])

    # set the limits
    xmin = min(around(states[:, 0].min() - segway_width / 2.0, 1), around(se
    xmax = max(around(states[:, 0].max() + segway_height / 2.0, 1), around(s

    # create the axes
    ax = plt.axes(xlim=(xmin-.2, xmax+.2), ylim=(-length-.1, length+.1), asp

    # display the current time
    time_text = ax.text(0.05, 0.9, '', transform=ax.transAxes)

    # display the current control
    control_text = ax.text(0.05, 0.8, '', transform=ax.transAxes)

    # create rectangle for the segway
    rect = Rectangle([states[0, 0] - segway_width / 2.0, -segway_height / 2]
        segway_width, segway_height, fill=True, color='gold', ec='blue')
    ax.add_patch(rect)

    # blank line for the stick with o for the ends
    stick_line, = ax.plot([], [], lw=2, marker='o', markersize=6, color='blu

    # vector for the control (force)
    force_vec = ax.quiver([],[],[],[],angles='xy',scale_units='xy',scale=1)

    # initialization function: plot the background of each frame
    def init():
        time_text.set_text('')
        control_text.set_text('')
        rect.set_xy((0.0, 0.0))
        stick_line.set_data([], [])
        return time_text, rect, stick_line, control_text

    # animation function: update the objects
    def animate(i):
        time_text.set_text('time = {:2.2f}'.format(t[i]))
        control_text.set_text('force = {:2.3f}'.format(controls[i]))
        rect.set_xy((states[i, 0] - segway_width / 2.0, -segway_height / 2))
```

```
        stick_line.set_data([states[i, 0], segwayStick_x[i]], [0, segwayStic
        return time_text, rect, stick_line, control_text

    # call the animator function
    anim = animation.FuncAnimation(fig, animate, frames=len(t), init_func=in
            interval=1000/fps, blit=False, repeat=False)
    return anim
    # plt.show()
```

Plug in your controller here

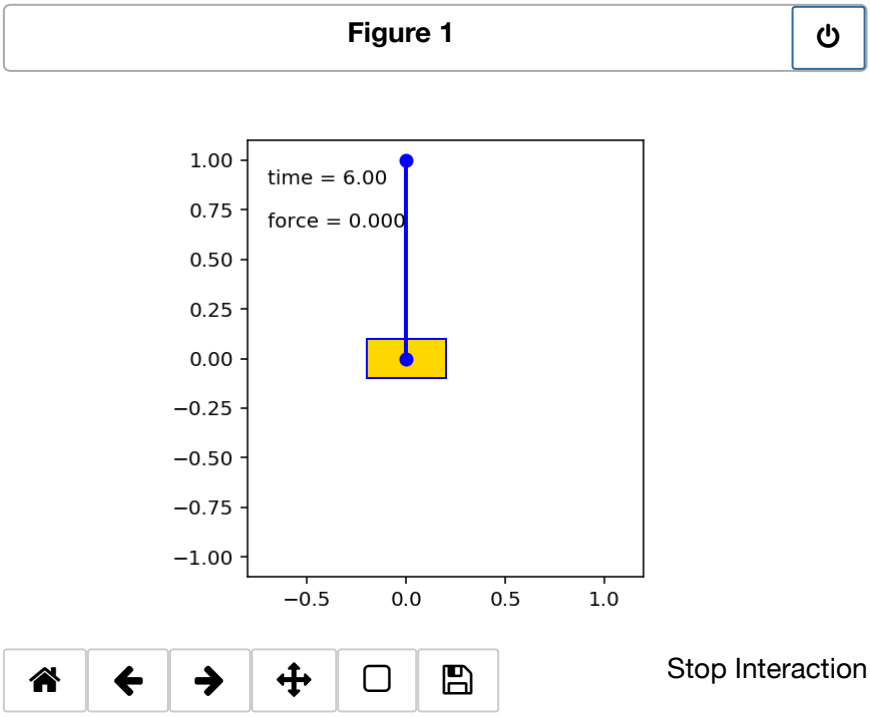
In [14]:
```
controls = result # np.array([0,0,0,0]) # here
```

Simulation

In [15]:
```
# This will add an extra couple of seconds to the simulation after the input
# the effect of this is just to show how the system will continue after the
controls = np.append(controls,[0, 0])

# number of steps in the simulation
nr_steps = controls.shape[0]

# We now compute finer dynamics and control vectors for smoother visualizati
Afine = sp.linalg.fractional_matrix_power(A,(1/fps))
Asum = np.eye(nr_states)
for i in range(1, fps):
    Asum = Asum + np.linalg.matrix_power(Afine,i)

bfine = np.linalg.inv(Asum).dot(b)

# We also expand the controls in the "intermediate steps" (only for visuali
controls_final = np.outer(controls, np.ones(fps)).flatten()
controls_final = np.append(controls_final, [0])

# We compute all the states starting from x0 and using the controls
states = np.empty([fps*(nr_steps)+1, nr_states])
states[0,:] = state0;
for stepId in range(1,fps*(nr_steps)+1):
    states[stepId, :] = np.dot(Afine,states[stepId-1, :]) + controls_final[s

# Now create the time vector for simulation
t = np.linspace(1/fps,nr_steps,fps*(nr_steps),endpoint=True)
t = np.append([0], t)
```

Visualization

In [17]:
```
%matplotlib nbagg
# %matplotlib qt
anim = animate_segway(t, states, controls_final, stick_length)
anim
```



Out[17]: <matplotlib.animation.FuncAnimation at 0x12d10f978>

In [ ]:

In [ ]: