Problem Set 1 – BE-130 / ES-249 (Due Friday, Feb 18)

(1) Forward kinematics:

(a) Derive the forward kinematics $[x_1 \ x_2]^T$ as a function of $[\Theta_1 \ \Theta_2]^T$ and derive the Jacobian matrix $[dx/d\Theta]$ for a two joint planar arm where the joint angles are defined "absolutely" with respect to the x-axis.

(b) Repeat for relative joint angles Θ. Note that , in general, relative angles are defined as the angle for each joint besides the 1st one being measured relative to the preceding limb segment (i.e. the elbow angle is relative to the upper arm).

(c) Determine cartesian velocity as a function of joint velocity [you can use absolute joint angles here].

(d) Determine cartesian acceleration as a function of joint acceleration & joint velocity.

(e-h) Repeat **(a-d)** for a three joint planar arm (for the relative angles in **(f)**, the elbow angle is relative to the upper arm and the wrist angle is relative to the forearm).

(2) Inverse kinematics, jacobians, & dynamics:

For the two joint absolute joint angle case above (1a):

(a) Derive the inverse kinematics: $[\Theta_1 \ \Theta_2]^T$ as a function of $[x_1 \ x_2]^T$ (note that although it might seem straightforward to invert the equations from 1a, this is surprisingly tricky, so ask for help if you get badly stuck…) *[Hint: Perhaps the easiest approach here is to sketch out the geometry and apply the law of cosines].*

(b) Derive the inverse Jacobian matrix [d**Θ**/d**X**]. [*This should be straightforward once you have **(a)***]

(c) Comment briefly on how these compare with the forward kinematics derived in **1a**.

(3) Simulating motion

(a) Create an interactive graphical simulation showing the 2-joint inverse kinematics computed from x-y mouse input, i.e. continually adjust the joint angles of a 2-link arm model so that its endpoint position matches the mouse cursor position. Please see the example MATLAB m-file **draw_arm_starter1.m** for some starter code that implements the sensing and graphics, so that this is less tedious to create.

(b) Note that the path associated with an arm movement can be represented in either cartesian space (x_1, x_2) or joint space (Θ_1, Θ_2). For a simple 2-link arm model, compare these two representations. In particular, compare hand paths for motion in cartesian space vs joint space for several movements that are 'straight' **(i)** in cartesian space and **(ii)** in joint space. Do this for movements of differing lengths and directions. Comment on the results.

Straight in joint space → curve

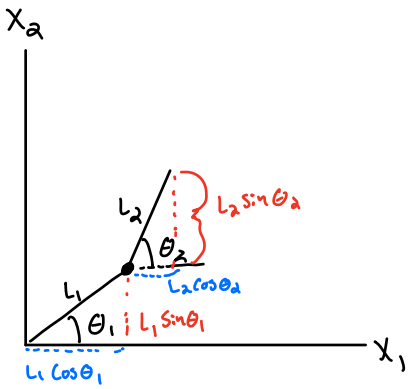
(4) Dynamics

(a) The maximum amount of force that can be produced Analytically compute (i) the minimum and (ii) the maximum achievable force levels that can be produced at any point in the workspace across all force directions for the 2-link arm simulated above, assuming that the max Torque level is T_{MAX} for both joints.

(b) Set T_{MAX}=1, and make a 2-D image showing (i) the minimum Force levels that can be produced at each (x,y) point in the workspace across all force directions for the 2-link arm simulated above.

(c) Create an interactive graphical simulation, like that in problem **3a**, that displays the force levels that can be produced at any point in the workspace for the 2-link arm simulated above. Display this as an elliptical shape centered at the current endpoint showing the max force that can be produced in each direction. Comment on the pattern.

(a) Derive the forward kinematics $[x_1 \ x_2]^T$ as a function of $[\Theta_1 \ \Theta_2]^T$ and derive the Jacobian matrix $[dx/d\Theta]$ for a two joint planar arm where the joint angles are defined "absolutely" with respect to the x-axis.



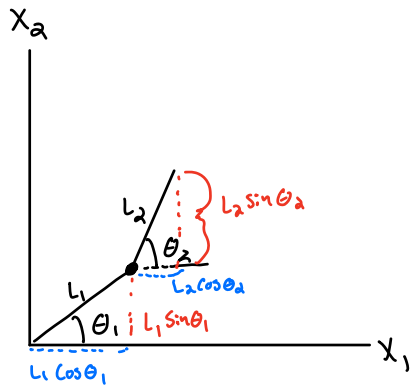
$$x_1 = L_1\cos\theta_1 + L_2\cos\theta_2$$

$$x_2 = L_1\sin\theta_1 + L_2\sin\theta_2$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} L_1\cos\theta_1 & L_2\cos\theta_2 \\ L\sin\theta_1 & L_2\sin\theta_2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\bar{J} = \begin{bmatrix} \frac{\partial x_1}{\partial \theta_1} & \frac{\partial x_1}{\partial \theta_2} \\ \frac{\partial x_2}{\partial \theta_1} & \frac{\partial x_2}{\partial \theta_2} \end{bmatrix} = \begin{bmatrix} -L_1\sin\theta_1, & -L_2\sin\theta_2 \\ L_1\cos\theta_1, & L_2\cos\theta_2 \end{bmatrix} = \begin{bmatrix} -L_1 s_1 & -L_2 s_2 \\ L_1 c_1 & L_2 c_2 \end{bmatrix}$$

(b) Repeat for relative joint angles Θ. Note that , in general, relative angles are defined as the angle for each joint besides the 1st one being measured relative to the preceding limb segment (i.e. the elbow angle is relative to the upper arm).



$$X_1 = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_{2R})$$

$$X_2 = L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_{2R})$$

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} L_1\cos\theta_1 & L_2\cos(\theta_1+\theta_{2R}) \\ L_1\sin\theta_1 & L_2\cos(\theta_1+\theta_{2R}) \end{bmatrix}$$

$$\bar{J} = \begin{bmatrix} \frac{\partial X_1}{\partial \theta_1} & \frac{\partial X_1}{\partial \theta_{2R}} \\ \frac{\partial X_2}{\partial \theta_1} & \frac{\partial X_2}{\partial \theta_{2R}} \end{bmatrix}$$

$$\bar{J} = \begin{bmatrix} -L_1\sin\theta_1 & -L_2\sin(\theta_1+\theta_{2R}) \\ L_1\cos\theta_1 & L_2\cos(\theta_1+\theta_{2R}) \end{bmatrix}$$

(c) Determine cartesian velocity as a function of joint velocity [you can use absolute joint angles here].

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} L_1\cos\theta_1 & L_2\cos\theta_2 \\ L_1\sin\theta_1 & L_2\sin\theta_2 \end{bmatrix}$$

$$\downarrow$$

$$\frac{dX_1}{dt} = -L\sin\theta_1\dot{\theta}_1 - L_2\sin\theta_2\dot{\theta}_2$$

$$\frac{dX_2}{dt} = L_1\cos\theta_1\dot{\theta}_1 + L_2\cos\theta_2\dot{\theta}_2$$

$$\dot{\vec{X}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -L_1\sin\theta_1\dot{\theta}_1 & -L_2\sin\theta_2\dot{\theta}_2 \\ L_1\cos\theta_1\dot{\theta}_1 & L_2\cos\theta_2\dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} -L_1 s_1 & -L_2 s_2 \\ L_1 c_1 & L_2 c_2 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

$$\dot{\vec{X}} = \bar{J} \cdot \dot{\vec{\theta}}$$

(d) Determine cartesian acceleration as a function of joint acceleration & joint velocity.

$$\ddot{\vec{X}} = \begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix}$$

$$\dot{\vec{X}} = J(\theta)\dot{\vec{\theta}}$$

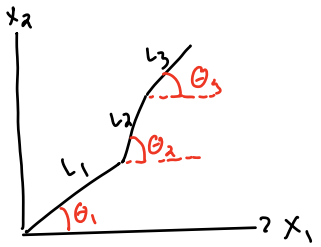
$$\ddot{\vec{X}} = J(\theta)\ddot{\vec{\theta}} + \dot{J}(\theta)\left(\dot{\vec{\theta}}\right)^2$$ ← derivative, apply product rule

$$J(\theta) = \begin{bmatrix} -L_1 c_1 & -L_2 c_2 \\ -L_1 s_1 & -L_2 s_2 \end{bmatrix}$$

$$\ddot{\vec{X}} = \begin{bmatrix} -L_1 s_1 & -L_2 s_2 \\ L_1 c_1 & L_2 c_2 \end{bmatrix}\begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} + \begin{bmatrix} -L_1 c_1 & -L_2 c_2 \\ -L_1 s_1 & -L_2 s_2 \end{bmatrix}\begin{bmatrix} \dot{\theta}_1^2 \\ \dot{\theta}_2^2 \end{bmatrix}$$

(e-h) Repeat **(a-d)** for a three joint planar arm (for the relative angles in **(f)**, the elbow angle is relative to the upper arm and the wrist angle is relative to the forearm).

e)



$$c_i = \cos(\theta_i)$$

$$s_i = \sin(\theta_i)$$

$$x_1 = L_1 c_1 + L_2 c_2 + L_3 c_3$$

$$x_2 = L_1 s_1 + L_2 s_2 + L_3 s_3$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{bmatrix} L_1 c_1 & L_2 c_2 & L_3 c_3 \\ L_1 s_1 & L_2 s_2 & L_3 s_3 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$J = \begin{bmatrix} \frac{\partial x_1}{\partial \theta_1} & \frac{\partial x_1}{\partial \theta_2} & \frac{\partial x_2}{\partial \theta_3} \\ \frac{\partial x_2}{\partial \theta_2} & \frac{\partial x_2}{\partial \theta_2} & \frac{\partial x_2}{\partial \theta_3} \end{bmatrix} = \begin{bmatrix} -L_1 s_1 & -L_2 s_2 & -L_3 s_3 \\ L_1 c_1 & L_2 c_2 & L_3 c_3 \end{bmatrix}$$ ← absolute angles

f)

$$x_1 = L_1\cos\theta_1 + L_2\cos(\theta_1+\theta_{2R}) + L_3\cos(\theta_1+\theta_{2R}+\theta_{3R})$$

$$x_2 = L_1\sin\theta_1 + L_2\sin(\theta_1+\theta_{2R}) + L_3\cos(\theta_1+\theta_{2R}+\theta_{3R})$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} L_1\cos\theta_1 & L_2\cos(\theta_1+\theta_{2R}) & L_3\cos(\theta_1+\theta_{2R}+\theta_{3R}) \\ L_1\sin\theta_1 & L_2\sin(\theta_1+\theta_{2R}) & L_3\cos(\theta_1+\theta_{2R}+\theta_{3R}) \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\bar{J} = \begin{bmatrix} -L_1\sin\theta_1 & -L_2\sin(\theta_1+\theta_{2R}) & -L_3\sin(\theta_1+\theta_{2R}+\theta_{3R}) \\ L_1\cos\theta_1 & L_2\cos(\theta_1+\theta_{2R}) & L_3\cos(\theta_1+\theta_{2R}+\theta_{3R}) \end{bmatrix}$$

g)

$$\dot{x}_1 = -L_1 s_1\dot{\theta}_1 - L_2 s_2\dot{\theta}_2 - L_3 s_3\dot{\theta}_3$$

$$\dot{x}_2 = L_1 c_1\dot{\theta}_1 + L_2 c_2\dot{\theta}_2 + L_3 c_3\dot{\theta}_3$$

$$\vec{\dot{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -L_1 s_1\dot{\theta}_1 & -L_2 s_2\dot{\theta}_2 & -L_3 s_3\dot{\theta}_3 \\ L_1 c_1\dot{\theta}_1 & L_2 c_2\dot{\theta}_2 & L_3 c_3\dot{\theta}_3 \end{bmatrix}$$

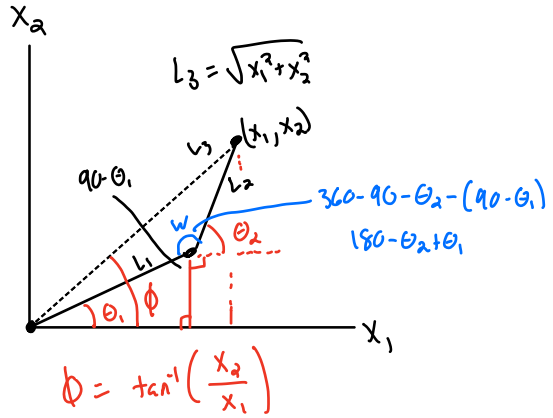
$$\vec{\dot{x}} = \begin{bmatrix} -L_1 s_1 & -L_2 s_2 & -L_3 s_3 \\ L_1 c_1 & L_2 c_2 & L_3 c_3 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \qquad \vec{\dot{x}} = \bar{J}\cdot\vec{\dot{\theta}}$$

h)

$$\vec{\ddot{x}} = \begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} = \bar{J}(\theta)\vec{\ddot{\theta}} + \dot{\bar{J}}(\theta)\vec{\dot{\theta}}^2$$

$$= \begin{bmatrix} -L_1 s_1 & -L_2 s_2 & -L_3 s_3 \\ L_1 c_1 & L_2 c_2 & L_3 c_3 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1^2 \\ \dot{\theta}_2^2 \\ \dot{\theta}_3^2 \end{bmatrix} + \begin{bmatrix} -L_1 c_1 & -L_2 c_2 & -L_3 c_3 \\ -L_1 s_1 & -L_2 s_2 & -L_3 s_3 \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_1 \\ \ddot{\theta}_3 \end{bmatrix}$$

(a) Derive the inverse kinematics: $[\Theta_1\ \Theta_2]^T$ as a function of $[x_1\ x_2]^T$ (note that although it might seem straightforward to invert the equations from 1a, this is surprisingly tricky, so ask for help if you get badly stuck…) *[Hint: Perhaps the easiest approach here is to sketch out the geometry and apply the law of cosines].*



By law of cosine w/ triangle formed by L_3, L_2, L_1

$$L_2^2 = L_1^2 + L_3^2 - 2L_1L_3\cos(\phi - \theta_1)$$

$$\frac{L_2^2 - L_1^2 - L_3^2}{-2L_1L_3} = \cos(\phi - \theta_1)$$

$$\phi - \theta_1 = \cos^{-1}\left(\frac{L_2^2 - L_1^2 - L_3^2}{-2L_1L_3}\right)$$

$$\theta_1 = \phi - \cos^{-1}\left(\frac{L_2^2 - L_1^2 - L_3^2}{-2L_1L_3}\right)$$

$$\theta_1 = \tan^{-1}\left(\frac{x_2}{x_1}\right) - \cos^{-1}\left(\frac{L_2^2 - L_1^2 - L_3^2}{-2L_1L_3}\right) = \tan^{-1}\left(\frac{x_2}{x_1}\right) - \cos^{-1}\left(\frac{L_2^2 - L_1^2 - (x_1^2 + x_2^2)}{-2L_1\sqrt{x_1^2 + x_2^2}}\right)$$

By Law of cosines again:

$$L_3^2 = L_1^2 + L_2^2 - 2L_1L_2\cos(w)$$

$$w = \cos^{-1}\left(\frac{L_3^2 - L_1^2 - L_2^2}{-2L_1L_2}\right) = \cos^{-1}\left(\frac{x_1^2 + x_2^2 - L_1^2 - L_2^2}{-2L_1L_2}\right)$$

(next page)

$$w = 180 - \theta_2 + \theta_1$$

$$\theta_2 = 180 + \theta_1 - w$$

$$\theta_2 = 180 - \theta_1 - \cos^{-1}\left(\frac{x_1^2 + x_2^2 - L_1^2 - L_2^2}{-2L_1L_2}\right)$$

$$= 180 - \cos^{-1}\left(\frac{x_1^2 + x_2^2 - L_1^2 - L_2^2}{-2L_1L_2}\right) - \tan^{-1}\left(\frac{x_2}{x_1}\right) - \cos^{-1}\left(\frac{L_2^2 - L_1^2 - (x_1^2 + x_2^2)}{-2L_1\sqrt{x_1^2 + y_2^2}}\right)$$

$$\begin{pmatrix}\theta_1 \\ \theta_2\end{pmatrix} = \begin{bmatrix} \tan^{-1}\left(\frac{x_2}{x_1}\right) - \cos^{-1}\left(\frac{L_2^2 - L_1^2 - (x_1^2 + x_2^2)}{-2L_1\sqrt{x_1^2 + y_2^2}}\right) \\ 180 - \cos^{-1}\left(\frac{x_1^2 + x_2^2 - L_1^2 - L_2^2}{-2L_1L_2}\right) + \tan^{-1}\left(\frac{x_2}{x_1}\right) - \cos^{-1}\left(\frac{L_2^2 - L_1^2 - (x_1^2 + x_2^2)}{-2L_1\sqrt{x_1^2 + y_2^2}}\right) \end{bmatrix}$$

(b) Derive the inverse Jacobian matrix [d**Θ**/d**X**]. [*This should be straightforward once you have* ***(a)***]

$$J = \begin{pmatrix} \frac{\partial\theta_1}{\partial x_1} & \frac{\partial\theta_1}{\partial x_2} \\ \frac{\partial\theta_2}{\partial x_1} & \frac{\partial\theta_2}{\partial x_2} \end{pmatrix}$$

From online partial calculator:

$$\frac{\partial\theta_1}{\partial x_1} = \frac{\frac{x_1}{L_1\sqrt{x_1^2 + x_2^2}} + \frac{x\left(-L_1^2 + L_2^2 - x_1^2 - x_2^2\right)}{2L_1\left(x_1^2 + x_2^2\right)^{3/2}}}{\sqrt{1 - \frac{\left(-L_1^2 + L_2^2 - x_1^2 - x_2^2\right)^2}{4L_1^2\left(x_1^2 + x_2^2\right)}}} - \frac{x_2}{x_1^2\left(1 + \frac{x_2^2}{x_1^2}\right)}$$

$$\frac{\partial \theta_1}{\partial x_2} = \frac{\frac{x_2}{L_1\sqrt{x_1^2+x_2^2}} + \frac{x_2\left(L_1^2+L_2^2-x_1^2-x_2^2\right)}{2L_1\left(x_1^2+x_2^2\right)^{3/2}}}{\sqrt{1-\frac{\left(-L_1^2+L_2^2-x_1^2-x_2^2\right)^2}{4L_1^2\left(x_1^2+x_2^2\right)}}} + \frac{1}{x_1\left(1+\frac{x_2^2}{x_1^2}\right)}$$

$$\frac{\partial \theta_2}{\partial x_1} = \frac{\frac{x_1}{L_1\sqrt{x_1^2+x_2^2}} + \frac{x_1\left(-L_1^2+L_2^2-x_1^2-x_2^2\right)}{2L_1\left(x_1^2+x_2^2\right)^{3/2}}}{\sqrt{1-\frac{\left(-L_1^2+L_2^2-x_1^2-x_2^2\right)^2}{4L_1^2\left(x_1^2+x_2^2\right)}}} - \frac{x_2}{x_1^2\left(1+\frac{x_2^2}{x_1^2}\right)} - \frac{x_1}{L_1L_2\sqrt{1-\frac{\left(-L_1^2-L_2^2+x_1^2+x_2^2\right)^2}{4L_1^2L_2^2}}}$$

$$\frac{\partial \theta_2}{\partial x_2} = \frac{\frac{x_2}{L_1\sqrt{x_1^2+x_2^2}} + \frac{x_2\left(-L_1^2+L_2^2-x_1^2-x_2^2\right)}{2L_1\left(x_1^2+x_2^2\right)^{3/2}}}{\sqrt{1-\frac{\left(-L_1^2+L_2^2-x_1^2-x_2^2\right)^2}{4L_1^2\left(x_1^2+x_2^2\right)}}} + \frac{1}{x_1\left(1+\frac{x_2^2}{x_1^2}\right)} - \frac{x_2}{L_1L_2\sqrt{1-\frac{\left(-L_1^2-L_2^2+x_1^2+x_2^2\right)^2}{4L_1^2L_2^2}}}$$

(c) Comment briefly on how these compare with the forward kinematics derived in **1a**.

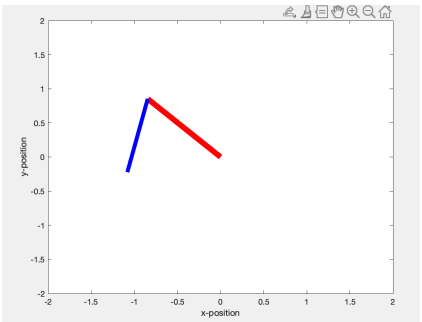
The inverse kinematics end up becoming much more complicated in their undifferentiated form due to the angle relationships. This then makes the Jacobean much harder to generate due to having to handle pythagorean distances to represent lengths at difference angles instead of just magnitude and join angle.

3)

(a) Create an interactive graphical simulation showing the 2-joint inverse kinematics computed from x-y mouse input, i.e. continually adjust the joint angles of a 2-link arm model so that its endpoint position matches the mouse cursor position. Please see the example MATLAB m-file **draw_arm_starter1.m** for some starter code that implements the sensing and graphics, so that this is less tedious to create.

(Reference code in draw_arm.m)

Link: https://github.com/alxrod/BE130_psets



(b) Note that the path associated with an arm movement can be represented in either cartesian space (x_1, x_2) or joint space (Θ_1, Θ_2). For a simple 2-link arm model, compare these two representations. In particular, compare hand paths for motion in cartesian space vs joint space for several movements that are 'straight' **(i)** in cartesian space and **(ii)** in joint space. Do this for movements of differing lengths and directions. Comment on the results.

The relationship that you get between cartesian and joint space is that constant/linear movements in joint space translate to curved movement in the Cartesian space. The longer the straight joint movement, the more curve int he cartesian space. For example, if theta2 states constant and theta1 just linearly increases from 0 to 360, you get a circular hand pattern (curved cartesian movement). Meanwhile. if you cause straight cartesian movement from the origin/shoulder outward, you get nonlinear joint movement where first theta2 increases then once the forearm is fully extended then theta1 increases as the whole upper arm extends. This is because there is a non linear relationship depending on cos and sin that causes one of the cartesian/joint cords to be straight while the other is curved.

(4) Dynamics

(a) The maximum amount of force that can be produced Analytically compute (i) the minimum and (ii) the maximum achievable force levels that can be produced at any point in the workspace across all force directions for the 2-link arm simulated above, assuming that the max Torque level is T_{MAX} for both joints.

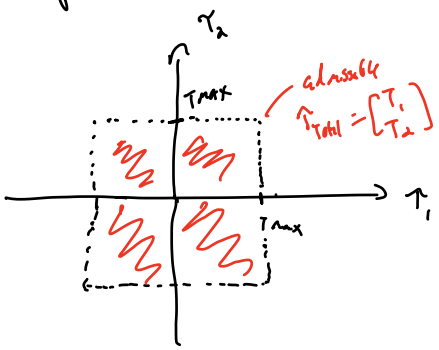
$$T = J^T F$$

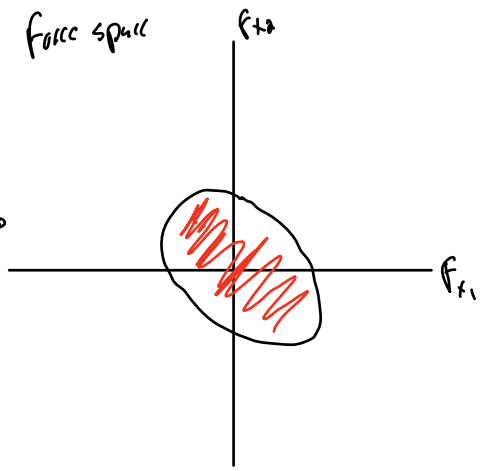
$$(J^T)^{-1} T = (J^T)^{-1} (J^T) F$$

$$(J^T)^{-1} T = F \quad \text{(via mechanics)}$$

$$\vec{F} = \begin{bmatrix} F_{x_1} \\ F_{x_2} \end{bmatrix} = \begin{bmatrix} F_{TOT} \cos\phi \\ F_{TOT} \sin\phi \end{bmatrix}$$

forces produced from T_1 T_2





$$J = \begin{bmatrix} -L_1 S_1 & -L_2 S_2 \\ L_1 C_1 & L_2 C_2 \end{bmatrix} \rightarrow J^T = \begin{bmatrix} -L_1 S_1 & L_1 C_1 \\ -L_1 S_2 & L_2 C_2 \end{bmatrix}$$

$$\downarrow \qquad J^T \qquad F$$

$$\tau = \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = \begin{bmatrix} -L_1 S_1 & L_1 C_1 \\ -L_1 S_2 & L_2 C_2 \end{bmatrix} \begin{bmatrix} F_{TOT} \cos\phi \\ F_{TOT} \sin\phi \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} -L_1 F_T S_1 \cos\phi + L_1 F_T C_1 \cos\phi \\ -L_1 F_T S_2 \cos\phi + L_2 F_T C_2 \sin\phi \end{bmatrix} = \begin{bmatrix} L_1 F_T \sin(\phi - \theta_1) \\ L_2 F_T \sin(\phi - \theta_2) \end{bmatrix} = \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = F_T \begin{bmatrix} L_1 \sin(\phi - \theta_1) \\ L_2 \sin(\phi - \theta_2) \end{bmatrix}$$

$$\tau = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} F_T$$

$$F_T = \frac{\tau_1}{\alpha_1} \qquad F_{TMAX} = \frac{T_{max}}{\alpha_1}$$

$$F_T = \frac{\tau_2}{\alpha_2} \qquad = \frac{T_{max}}{\alpha_2}$$

$$\rightarrow F_{TMAX} = \min\left(\frac{T_{max}}{\alpha_1}, \frac{T_{max}}{\alpha_2} \right)$$

$$F_{TMAX} = \min\left(\frac{T_{max}}{L_1 \sin(\phi - \theta_1)}, \frac{T_{max}}{L_2 \sin(\phi - \theta_2)} \right)$$

Maximum = denominator smallest, when $\sin(\phi - \theta) = 0$

$\sin = 0$ at $0°$ and $180°$ so minimum values:

$\phi = \theta_1 | \theta_2, \theta_1 | \theta_2 + 180$

minimum = denominator largest, when $\sin(\phi - \theta) = 1$

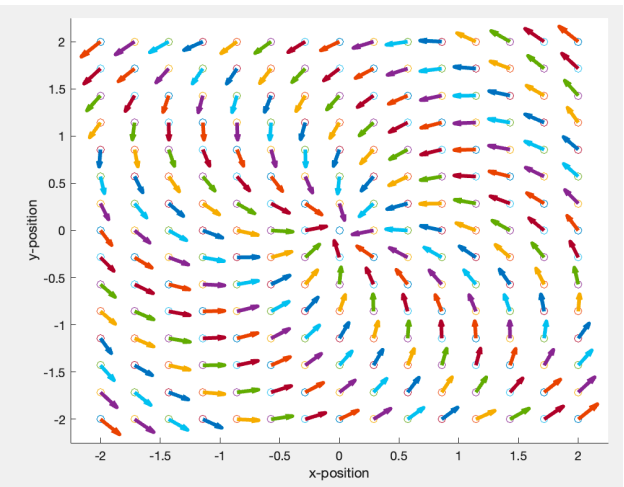
$\sin = 1$ at 90 and 270

optimal angles:

$\phi = \theta_1 | \theta_2 + 90, \theta_1 | \theta_2 + 270$

4b)

(b) Set T_{MAX}=1, and make a 2-D image showing (i) the minimum Force levels that can be produced at each (x,y) point in the workspace across all force directions for the 2-link arm simulated above.



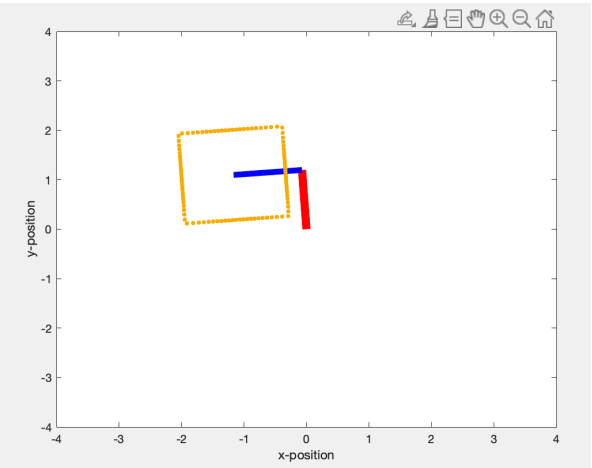
Reference code in max_force_diagram.m

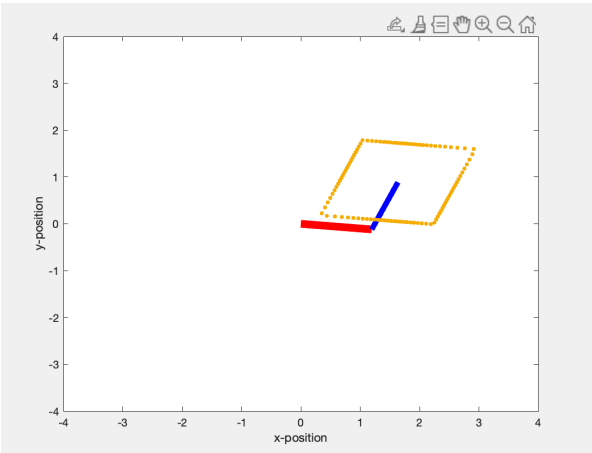
This diagram shows a vector at uniform points across the arm workspace. Each vector has the magnitude of the max force magnitude in the direction required to generate the mininum amount of force. Due to the equations above and l2>l1, we know that the max force is always tmax/L2 because the max we can get the sin function is 1. Therefore, the largest denominator is L2*1 so the smallest max force for phi is when we have TMAX/L2. We then can calculate which direction that phi is in at each one of these points and graph the corresponding vectors.

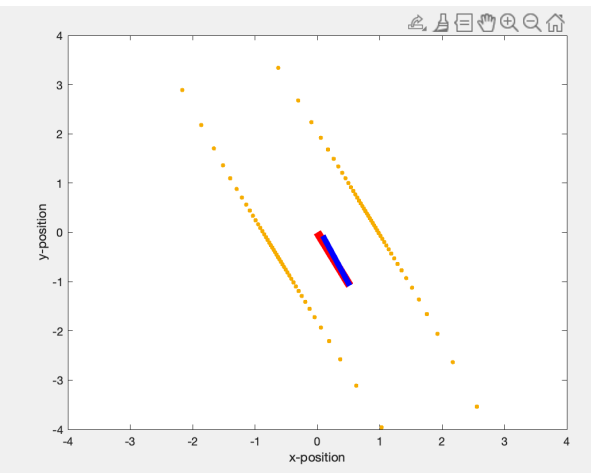
It is also **important to note** that this graph only shows one minimum vector per point. However, in reality, for every minimum vector, that phi-180 gives another minimum vector in the opposite direction. Therefore, the true vector field should have two opposite facing vectors per coordinate, however, I left the opposites out to prevent the diagram from being cluttered.

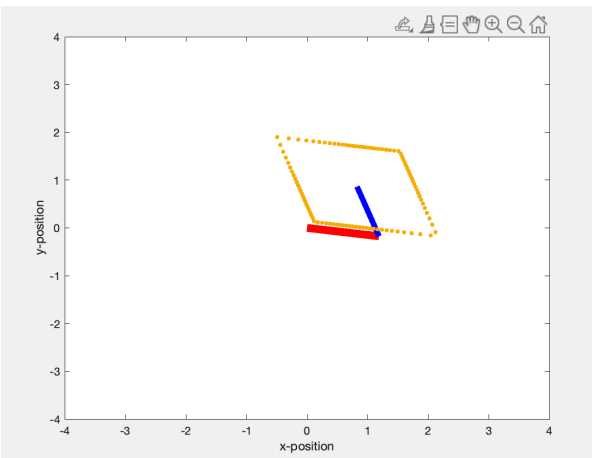
Link: https://github.com/alxrod/BE130_psets

(c) Create an interactive graphical simulation, like that in problem **3a**, that displays the force levels that can be produced at any point in the workspace for the 2-link arm simulated above. Display this as an elliptical shape centered at the current endpoint showing the max force that can be produced in each direction. Comment on the pattern.







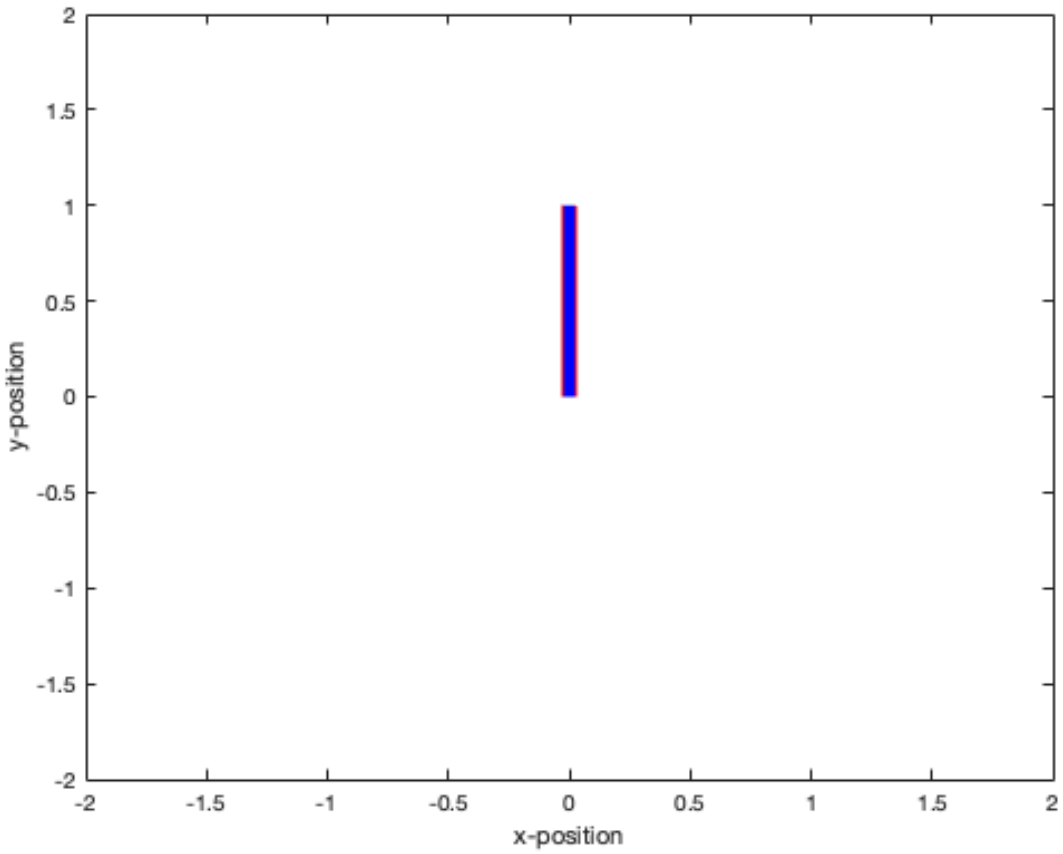


In this interactive program, you can see that the max force diagram shows a diamond like formation surrounding our hand joint. If you look at the right hand images, you can see that the minimum max force at any x y position is always existing on two opposite corners of the diamond 180 degrees from each other around the hand position. You can slo see that the minimum force magnitude is always constant in all four diagrams, the only thing that changes is which corners of the diamond are the minimums. In the top right image you can see that the minimums are the top left and bottom right corners. However, if you move the elbow just a little bit (bottom right diagram), the corners flip and the top right and bottom left corners become the minimum max forces. You can also see that in the bottom left diagram shows that as phi approaches = theta1 or theta2, the denominator of the force equation approaches zero, making the force infinitely large.

Link: https://github.com/alxrod/BE130_psets

```
function draw_arm1()
fig=figure;  ax=gca;
set (fig, 'WindowButtonMotionFcn', @(obj,event)mousemovedetected());   %
 "mousemovedetected" will be main function.
% It handles reading the current mouse position whenever movement is detected,
 and then redrawing the screen based on the detected position

L1=1.2;   L2=1.1;
redline  = plot([0 0],[1 0],'r','linewidth',7); hold on;  % create the red
 line for L1 that will be continuously repositioned based on the mouse cursor
blueline = plot([0 0],[1 0],'b','linewidth',5);           % create the blue
 line for L2
xlim([-2 2]); ylim([-2 2]);
% linspace
xlabel('x-position')
ylabel('y-position')

    function mousemovedetected()
        if overaxis(ax),
            C = get (ax, 'CurrentPoint');   % read the current mouse position
 (x,y)
            x = C(1,1);
            y = C(1,2);

            theta1 = atan2(y,x) - acos( (L2^2 - L1^2 - x^2 - y^2) /
 ( -2*L1*sqrt( (x^2+y^2) ) ) );
            theta2 = pi - acos( (x^2+y^2-L1^2-L2^2) / (-2*L1*L2) ) +
 atan2(y,x) - acos( (L2^2-L1^2-x^2-y^2) / (-2*L1*sqrt( (x^2+y^2)) ) ) ;

            l1_x = cos(theta1)*L1;
            l1_y = sin(theta1)*L1;

            l2_x = cos(theta2)*L2;
            l2_y = sin(theta2)*L2;


            if isreal(theta1) & isreal(theta2),

                set(redline, 'xdata',[0, l1_x]);  % edit these 2 lines to draw
 L1
                set(redline, 'ydata',[0, l1_y]);
                set(blueline, 'xdata',[l1_x, l1_x+l2_x]); % edit these 2 lines
 to draw L2
                set(blueline, 'ydata',[l1_y, l1_y+l2_y]);
            end
        end % end the if statement
    end % end the mousemovedetected function

    function z = overaxis(ax)   % determines whether the cursor is over the
 specified axis 'ax'
        C = get (ax, 'CurrentPoint'); Cx=C(1,1); Cy=C(1,2);
```

```
        z = (Cx>ax.XLim(1)) & (Cx<ax.XLim(2)) & (Cy>ax.YLim(1)) &
 (Cy<ax.YLim(2));
    end % end the overaxis function
end
```

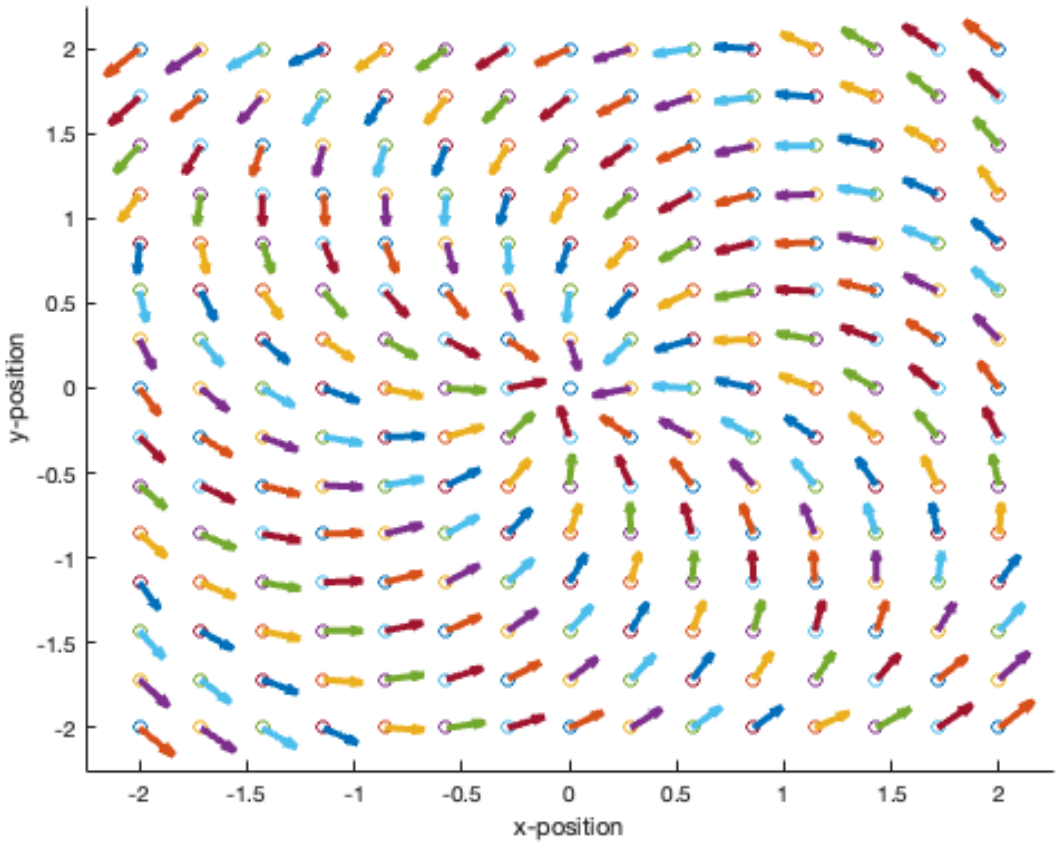


Published with MATLAB® R2021b

```
function force_diagram()
fig=figure;  ax=gca;
% "mousemovedetected" will be main function.
% It handles reading the current mouse position whenever movement is detected,
 and then redrawing the screen based on the detected position

L1=1.2;   L2=1.1;   TMAX=1;
xlim([-2.25 2.25]); ylim([-2.25 2.25]);
xlabel('x-position')
x_ar = linspace(-2,2,15);
y_ar = linspace(-2,2,15);
ylabel('y-position')

hold on;
for i = 1:length(x_ar)
    for j = 1:length(y_ar)
        x = x_ar(i);
        y = y_ar(j);

        theta1 = atan2(y,x) - acos( (L2^2 - L1^2 - x^2 - y^2) /
 ( -2*L1*sqrt( (x^2+y^2) ) ) );
        theta2 = pi - acos( (x^2+y^2-L1^2-L2^2) / (-2*L1*L2) ) + atan2(y,x) -
 acos( (L2^2-L1^2-x^2-y^2) / (-2*L1*sqrt( (x^2+y^2)) ) ) ;


        %If sin evals to 1, TMAX/L1*SIN always > TMAX/L2*SIN since L2>l1
        op = TMAX / L2;

        phi = theta2 + (pi/2);


        deltax = op*cos(phi);
        deltay = op*sin(phi);

        plot(x,y,'o');
        quiver(x, y, deltax, deltay,0.2,'linewidth',3, 'MaxHeadSize', 5);

    end
end
hold off;


end
```

```
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
```

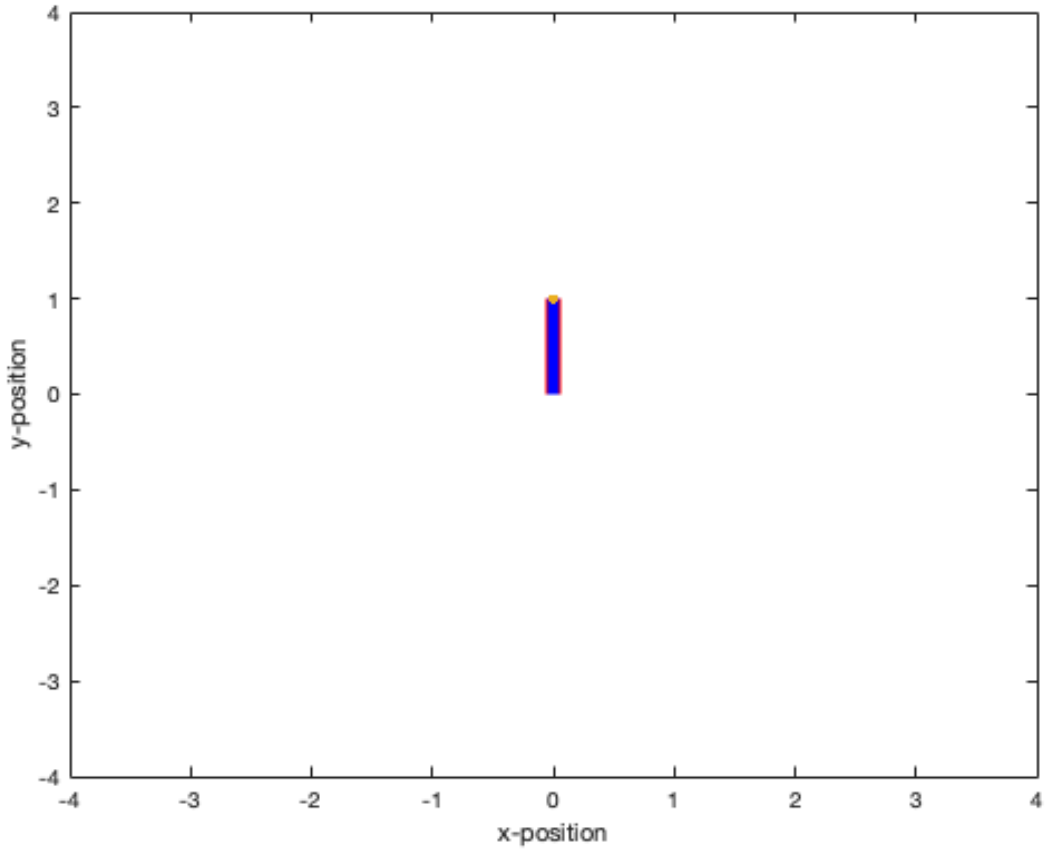
```
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
Warning: Using only the real component of complex data.
```



Published with MATLAB® R2021b

```
function draw_arm1()
fig=figure;  ax=gca;
set (fig, 'WindowButtonMotionFcn', @(obj,event)mousemovedetected());   %
 "mousemovedetected" will be main function.
% It handles reading the current mouse position whenever movement is detected,
 and then redrawing the screen based on the detected position

L1=1.2;   L2=1.1;   TMAX=1;
redline  = plot([0 0],[1 0],'r','linewidth',7); hold on;  % create the red
 line for L1 that will be continuously repositioned based on the mouse cursor
blueline = plot([0 0],[1 0],'b','linewidth',5);
f_points = plot([0],[1],'.','MarkerSize',10);
xlim([-4 4]); ylim([-4 4]);
phi_set = linspace(0,2*pi,100);
% linspace
xlabel('x-position')
ylabel('y-position')

    function mousemovedetected()
        if overaxis(ax),
            C = get (ax, 'CurrentPoint');   % read the current mouse position
 (x,y)
            x = C(1,1);
            y = C(1,2);

            theta1 = atan2(y,x) - acos( (L2^2 - L1^2 - x^2 - y^2) /
 ( -2*L1*sqrt( (x^2+y^2) ) ) );
            theta2 = pi - acos( (x^2+y^2-L1^2-L2^2) / (-2*L1*L2) ) +
 atan2(y,x) - acos( (L2^2-L1^2-x^2-y^2) / (-2*L1*sqrt( (x^2+y^2)) ) ) ;

            l1_x = cos(theta1)*L1;
            l1_y = sin(theta1)*L1;

            l2_x = cos(theta2)*L2;
            l2_y = sin(theta2)*L2;

            force_points = {}
            for p = 1:length(phi_set)
                val1 = abs(TMAX/ (L1*sin(phi_set(p) - theta1)) );
                val2 = abs(TMAX/ (L2*sin(phi_set(p) - theta2)) );

                mag = min([val1, val2]);
%                 fprintf("min %i\n", mag)
%                 mag=1

                deltax = mag*cos(phi_set(p));
                deltay = mag*sin(phi_set(p));

                p_x = l1_x+l2_x+deltax;
                p_y = l1_y+l2_y+deltay;

                force_points{end+1} = [p_x p_y];
```

```
            end

            if isreal(theta1) & isreal(theta2),
                set(redline, 'xdata',[0, l1_x]);  % edit these 2 lines to draw
 L1
                set(redline, 'ydata',[0, l1_y]);
                set(blueline, 'xdata',[l1_x, l1_x+l2_x]); % edit these 2 lines
 to draw L2
                set(blueline, 'ydata',[l1_y, l1_y+l2_y]);

                XY = cell2mat(force_points);
                X = XY(:,1:2:end);
                Y = XY(:,2:2:end);

                set(f_points, 'xdata', X);
                set(f_points, 'ydata', Y);
            end
        end % end the if statement
    end % end the mousemovedetected function

    function z = overaxis(ax)   % determines whether the cursor is over the
 specified axis 'ax'
        C = get (ax, 'CurrentPoint'); Cx=C(1,1); Cy=C(1,2);
        z = (Cx>ax.XLim(1)) & (Cx<ax.XLim(2)) & (Cy>ax.YLim(1)) &
 (Cy<ax.YLim(2));
    end % end the overaxis function
end
```



Published with MATLAB® R2021b