ES 249 Problem Set 1

Andrew Sullivan

14 February 2019

Question 1

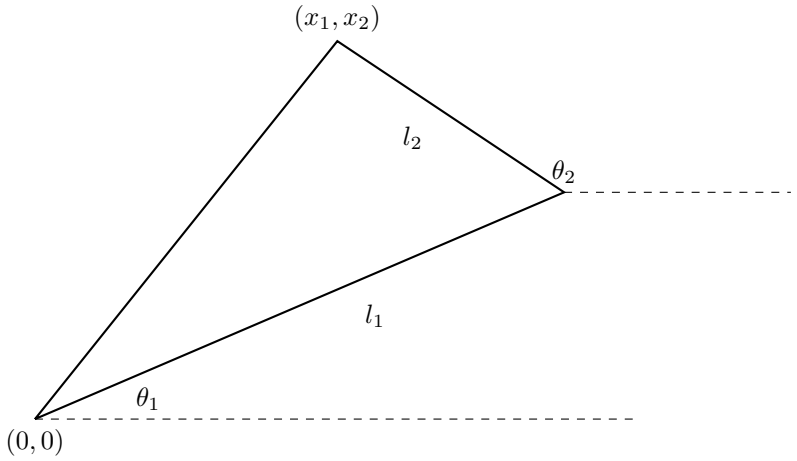


Figure 1: Absolute Kinematics Setup

a) The horizontal component of the final position, x_1, is the sum of the two horizontal contributions, and the vertical component, x_2, is the sum of the two vertical contributions:

$$x_1 = l_1 \cos\theta_1 + l_2 \cos\theta_2$$
$$x_2 = l_1 \sin\theta_1 + l_2 \sin\theta_2$$

The Jacobian matrix is then:

$$\mathbf{J} = \frac{d\mathbf{x}}{d\boldsymbol{\theta}} =$$

$$\begin{bmatrix} \frac{\partial x_1}{\partial \theta_1} & \frac{\partial x_1}{\partial \theta_2} \\ \frac{\partial x_2}{\partial \theta_1} & \frac{\partial x_2}{\partial \theta_2} \end{bmatrix} = \begin{bmatrix} -l_1 \sin\theta_1 & -l_2 \sin\theta_2 \\ l_1 \cos\theta_1 & l_2 \cos\theta_2 \end{bmatrix}$$

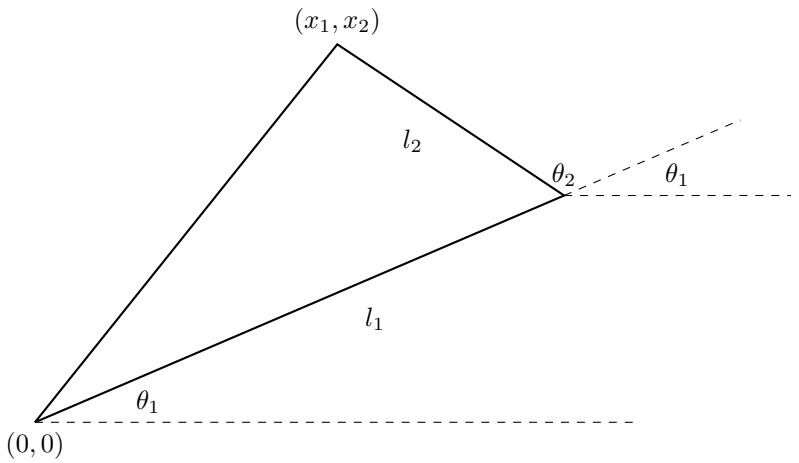


Figure 2: Relative Kinematics Setup

b) The horizontal component of the final position, x_1, is the sum of the two horizontal contributions, and the vertical component, x_2, is the sum of the two vertical contributions:

$$x_1 = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)$$
$$x_2 = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)$$

The Jacobian matrix is then:

$$\mathbf{J} = \frac{d\mathbf{x}}{d\boldsymbol{\theta}} =$$

$$\begin{bmatrix} \frac{\partial x_1}{\partial \theta_1} & \frac{\partial x_1}{\partial \theta_2} \\ \frac{\partial x_2}{\partial \theta_1} & \frac{\partial x_2}{\partial \theta_2} \end{bmatrix} = \begin{bmatrix} -l_1 \sin\theta_1 - l_2 \sin(\theta_1 + \theta_2) & -l_2 \sin(\theta_1 + \theta_2) \\ l_1 \cos\theta_1 + l_2 \cos(\theta_2 + \theta_2) & l_2 \cos(\theta_2 + \theta_2) \end{bmatrix}$$

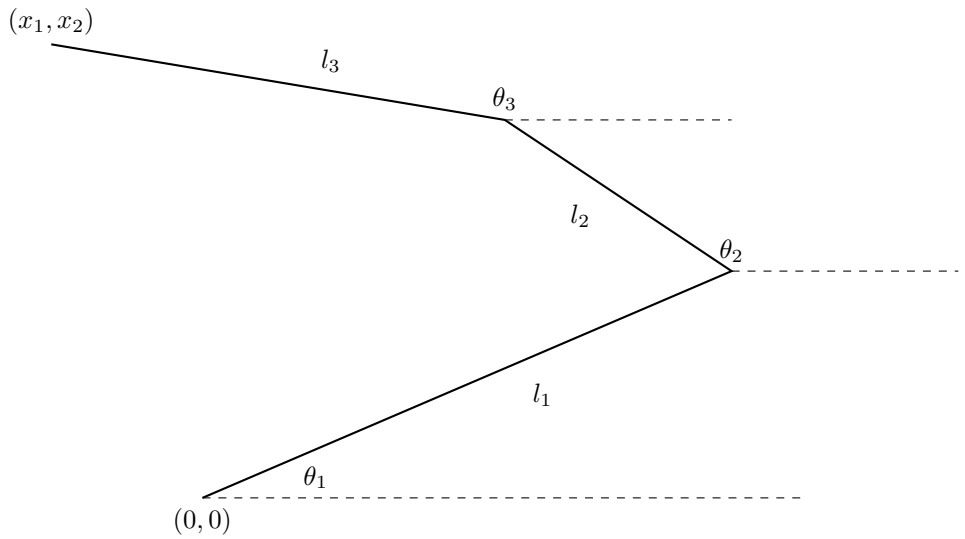


Figure 3: Three-Link Absolute Kinematics Setup

c) The horizontal component of the final position, x_1, is the sum of the two horizontal contributions, and the vertical component, x_2, is the sum of the two vertical contributions:

$$x_1 = l_1 \cos\theta_1 + l_2 \cos\theta_2 + l_3 \cos\theta_3$$
$$x_2 = l_1 \sin\theta_1 + l_2 \sin\theta_2 + l_3 \sin\theta_3$$

The Jacobian matrix is then:

$$\mathbf{J} = \frac{d\mathbf{x}}{d\boldsymbol{\theta}} =$$

$$\begin{bmatrix} \frac{\partial x_1}{\partial \theta_1} & \frac{\partial x_1}{\partial \theta_2} & \frac{\partial x_1}{\partial \theta_3} \\ \frac{\partial x_2}{\partial \theta_1} & \frac{\partial x_2}{\partial \theta_2} & \frac{\partial x_2}{\partial \theta_3} \end{bmatrix} = \begin{bmatrix} -l_1 \sin\theta_1 & -l_2 \sin\theta_2 & -l_3 \sin\theta_3 \\ l_1 \cos\theta_1 & l_2 \cos\theta_2 & -l_3 \cos\theta_3 \end{bmatrix}$$

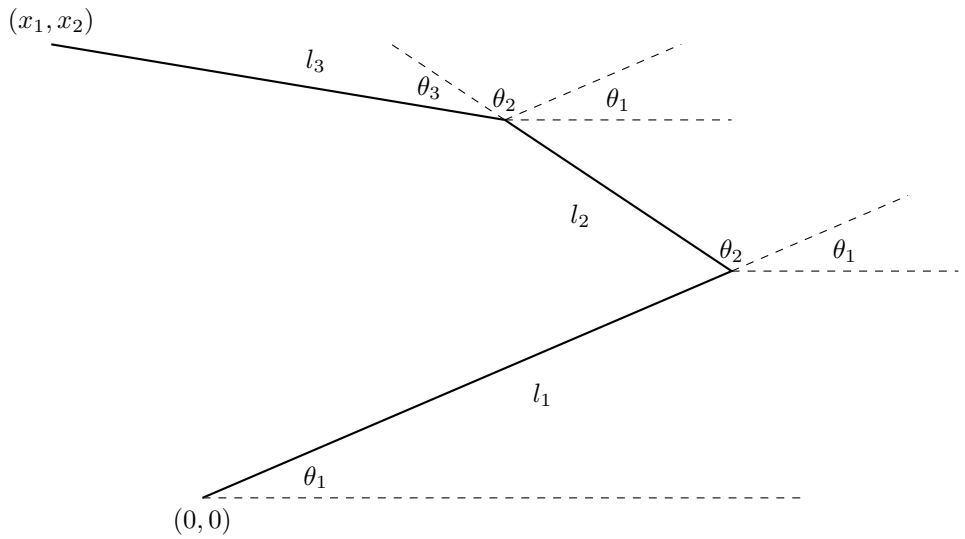


Figure 4: Three-Link Relative Kinematics Setup

d) The horizontal component of the final position, x_1, is the sum of the two horizontal contributions, and the vertical component, x_2, is the sum of the two vertical contributions:

$$x_1 = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3)$$
$$x_2 = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3)$$

The Jacobian matrix is then:

$$\mathbf{J} = \frac{d\mathbf{x}}{d\boldsymbol{\theta}} =$$

$$\begin{bmatrix} \frac{\partial x_1}{\partial \theta_1} & \frac{\partial x_1}{\partial \theta_2} & \frac{\partial x_1}{\partial \theta_3} \\ \frac{\partial x_2}{\partial \theta_1} & \frac{\partial x_2}{\partial \theta_2} & \frac{\partial x_2}{\partial \theta_3} \end{bmatrix} =$$

$$\begin{bmatrix} -l_1 \sin\theta_1 - l_2 \sin(\theta_1 + \theta_2) - l_3 \sin(\theta_1 + \theta_2 + \theta_3) & -l_2 \sin(\theta_1 + \theta_2) - l_3 \sin(\theta_1 + \theta_2 + \theta_3) & -l_3 \sin(\theta_1 + \theta_2 + \theta_3) \\ l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) & l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) & l_3 \cos(\theta_1 + \theta_2 + \theta_3) \end{bmatrix}$$

Question 2

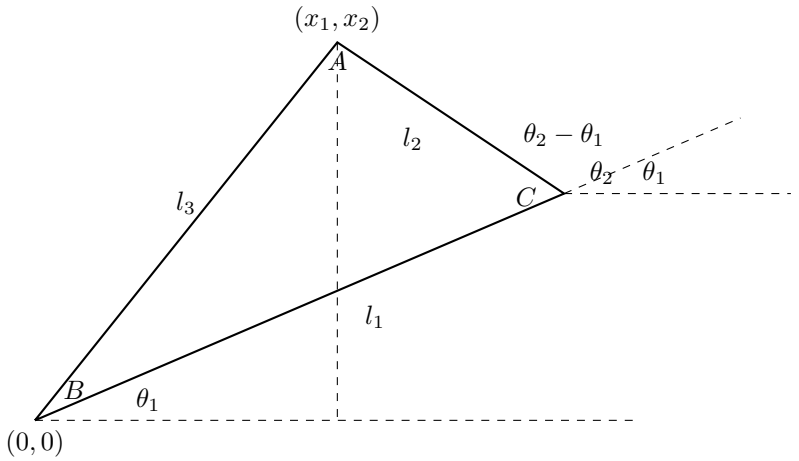


Figure 5: Inverse Kinematics Setup

a) In the above diagram, the length of the longest side is given by $l_3 = \sqrt{x_1^2 + x_2^2}$. Using the law of cosines, angle C can be found to be:

$$C = \arccos \frac{x_1^2 + x_2^2 - l_1^2 - l_2^2}{-2l_1 l_2}$$

As shown in the diagram, in the absolute configuration, the angle between the edge of length l_2 and the diagonal construction line extending from angle C is the difference between θ_2 and θ_1. This difference is also equal to $\pi - C$ radians, or:

$$\theta_2 = \theta_1 + \pi - \arccos \frac{x_1^2 + x_2^2 - l_1^2 - l_2^2}{-2l_1 l_2}$$

Using the right triangle construction line, it can be seen that $B + \theta_1 = \arctan \frac{x_2}{x_1}$, and again, through the law of cosines, $B = \arccos \frac{l_2 - x_1^2 - x_2^2 - l_1^2}{-2l_1 l_3}$. Thus,

$$\theta_1 = \arctan \frac{x_2}{x_1} - \arccos \frac{l_2^2 - x_1^2 - x_2^2 - l_1^2}{-2l_1 l_3}$$

$$\theta_2 = \pi + \arctan \frac{x_2}{x_1} - \arccos \frac{l_2^2 - x_1^2 - x_2^2 - l_1^2}{-2l_1 l_3} - \arccos \frac{x_1^2 + x_2^2 - l_1^2 - l_2^2}{-2l_1 l_2}$$

In the above solutions, the four-quadrant arctangent function is used to reduce ambiguity in solving for coordinates in different quadrants. In Matlab, the function atan2(x,y) performs this calculation. However, it should also be noted that the initial conditions for this problem do not yield a unique solution. If we consider the following geometry, where θ_2 is less than θ_1 by the same amount which it was greater above, we are unable to differentiate between the solutions using the above equations. Thus, the configuration must be known beforehand to decide between these "elbow up" and "elbow down" configurations. We can then solve for this configuration by noting that we now have $\theta_2 = \theta_1 - \pi + \arccos \frac{x_1^2 + x_2^2 - l_1^2 - l_2^2}{-2l_1 l_2}$ and $\theta_1 = B + \arctan \frac{x_2}{x_1}$, and then deriving the analogous formulae:

$$\theta_1 = \arctan \frac{x_2}{x_1} + \arccos \frac{l_2^2 - x_1^2 - x_2^2 - l_1^2}{-2l_1 l_3}$$

$$\theta_2 = -\pi + \arctan \frac{x_2}{x_1} + \arccos \frac{l_2^2 - x_1^2 - x_2^2 - l_1^2}{-2l_1 l_3} + \arccos \frac{x_1^2 + x_2^2 - l_1^2 - l_2^2}{-2l_1 l_2}$$

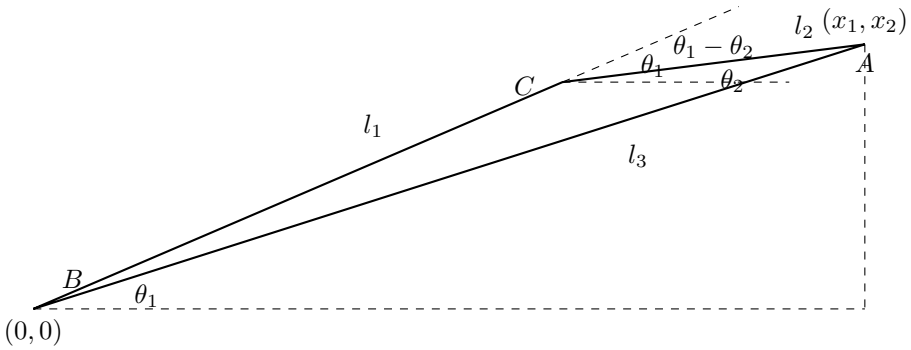


Figure 6: Inverse Kinematics Second Configuration

b) In order to find the inverse Jacobian, we differentiate the results from a):

$$\boldsymbol{J^{-1}} = \frac{d\boldsymbol{\theta}}{d\mathbf{x}}$$

The derivation of the inverse Jacobian requires the following identities:

$$\frac{\partial \arctan x}{\partial x} = \frac{1}{1+x^2}$$
$$\frac{\partial \arccos x}{\partial x} = \frac{1}{\sqrt{1-x^2}}$$

Therefore, the derivatives of the arctangent term are:

$$\frac{\partial \arctan \frac{x_2}{x_1}}{\partial x_1} = \frac{-x_2}{x_1^2+x_2^2}$$
$$\frac{\partial \arccos \frac{x_2}{x_1}}{\partial x_2} = \frac{x_1}{x_1^2+x_2^2}$$

The derivative of the second arccosine term in the equation for θ_2 is:

$$\frac{\partial}{\partial x_1} \arccos \frac{x_1^2+x_2^2-l_1^2-l_2^2}{-2l_1l_2} = \frac{x_1}{l_1l_2\sqrt{1-C^2}}$$
$$\frac{\partial}{\partial x_2} \arccos \frac{x_1^2+x_2^2-l_1^2-l_2^2}{-2l_1l_2} = \frac{x_2}{l_1l_2\sqrt{1-C^2}}$$

Here, C represents the argument to the arccosine term. Since both the numerator and denominator of the other arccosine term are dependent on the spatial variables (through l_3), the case for this term is more complicated. Here, using the quotient rule and simplifying, and using B as the argument to the arccosine term, the derivatives yield:

$$\frac{\partial}{\partial x_1} \arccos \frac{l_2^2-x_1^2-x_2^2-l_1^2}{-2l_1l_3} = \frac{-1}{\sqrt{1-B^2}}[\frac{x_1}{l_1l_3} + \frac{x_1(l_2^2-x_1^2-x_2^2-l_1^2)}{2l_1l_3^3}]$$
$$\frac{\partial}{\partial x_2} \arccos \frac{l_2^2-x_1^2-x_2^2-l_1^2}{-2l_1l_3} = \frac{-1}{\sqrt{1-B^2}}[\frac{x_2}{l_1l_3} + \frac{x_2(l_2^2-x_1^2-x_2^2-l_1^2)}{2l_1l_3^3}]$$

Thus, combining the above results, the inverse Jacobian is:

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

$$\begin{bmatrix} \frac{-x_2}{x_1^2+x_2^2} + \frac{1}{\sqrt{1-B^2}}\left[\frac{x_1}{l_1 l_3} + \frac{x_1(l_2^2-x_1^2-x_2^2-l_1^2)}{2l_1 l_3^3}\right] & \frac{x_1}{x_1^2+x_2^2} + \frac{1}{\sqrt{1-B^2}}\left[\frac{x_2}{l_1 l_3} + \frac{x_2(l_2^2-x_1^2-x_2^2-l_1^2)}{2l_1 l_3^3}\right] \\ \frac{-x_2}{x_1^2+x_2^2} + \frac{1}{\sqrt{1-B^2}}\left[\frac{x_1}{l_1 l_3} + \frac{x_1(l_2^2-x_1^2-x_2^2-l_1^2)}{2l_1 l_3^3}\right] - \frac{x_1}{l_1 l_2\sqrt{1-C^2}} & \frac{x_1}{x_1^2+x_2^2} + \frac{1}{\sqrt{1-B^2}}\left[\frac{x_2}{l_1 l_3} + \frac{x_2(l_2^2-x_1^2-x_2^2-l_1^2)}{2l_1 l_3^3}\right] - \frac{x_2}{l_1 l_2\sqrt{1-C^2}} \end{bmatrix}$$

Similarly, the inverse Jacobian for the second configuration is:

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

$$\begin{bmatrix} \frac{-x_2}{x_1^2+x_2^2} - \frac{1}{\sqrt{1-B^2}}\left[\frac{x_1}{l_1 l_3} + \frac{x_1(l_2^2-x_1^2-x_2^2-l_1^2)}{2l_1 l_3^3}\right] & \frac{x_1}{x_1^2+x_2^2} - \frac{1}{\sqrt{1-B^2}}\left[\frac{x_2}{l_1 l_3} + \frac{x_2(l_2^2-x_1^2-x_2^2-l_1^2)}{2l_1 l_3^3}\right] \\ \frac{-x_2}{x_1^2+x_2^2} - \frac{1}{\sqrt{1-B^2}}\left[\frac{x_1}{l_1 l_3} + \frac{x_1(l_2^2-x_1^2-x_2^2-l_1^2)}{2l_1 l_3^3}\right] + \frac{x_1}{l_1 l_2\sqrt{1-C^2}} & \frac{x_1}{x_1^2+x_2^2} - \frac{1}{\sqrt{1-B^2}}\left[\frac{x_2}{l_1 l_3} + \frac{x_2(l_2^2-x_1^2-x_2^2-l_1^2)}{2l_1 l_3^3}\right] + \frac{x_2}{l_1 l_2\sqrt{1-C^2}} \end{bmatrix}$$

c) From the above inverse equations, we can see certain similarities with the forward equations previously dervied. In both cases a given variable is dependent on the coordinates of both of the corresponding transformed variables (e.g. θ_1 depends on both x_1 and x_2, and x_1 depends on both θ_1 and θ_2), along with the lengths of both links. Interestingly, we can also see that, in the inverse case, the two variables are coupled- the value of θ_2 is dependent on the value of θ_1, while this is not the case for the two end-effector coordinates. This is also evident in the inverse Jacboian, in which the expression for the partial derivatives of θ_1 appear in those for θ_2. If we were instead to define relative coordinates, as in 1b), we would expect to see this dependence drop out.

d) The code found in the appendix takes in the mouse cursor coordinates as inputs, along with a user-defined variable "arm_config". If this variable is set equal to the string 'up', the configuration will flip so the arm is facing downwards. For all other values, it will face upwards. The angles are computed for fixed link lengths and lines are drawn to the location of the cursor. This plot is continuously updated as the coordinates change. The figure below displays an example of the interface.

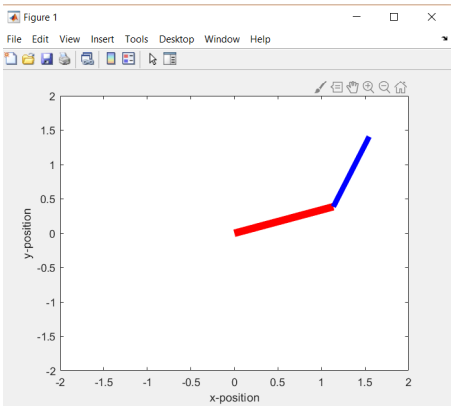


Figure 7: Example screenshot of the interface.

e) For a given end-effector force, the resultant torque at a joint can be obtained by decomposing the forces and moment arms into components along the x_1 and x_2 directions, such that $\tau = -F_{x_1} r_{x_2} + F_{x_2} r_{x_1}$. For small angular shifts $d\theta$, the two distance components correspond to $\frac{dx_1}{d\theta}$ and $\frac{dx_2}{d\theta}$, respectively. Thus, the force-torque relationship is summarized as:

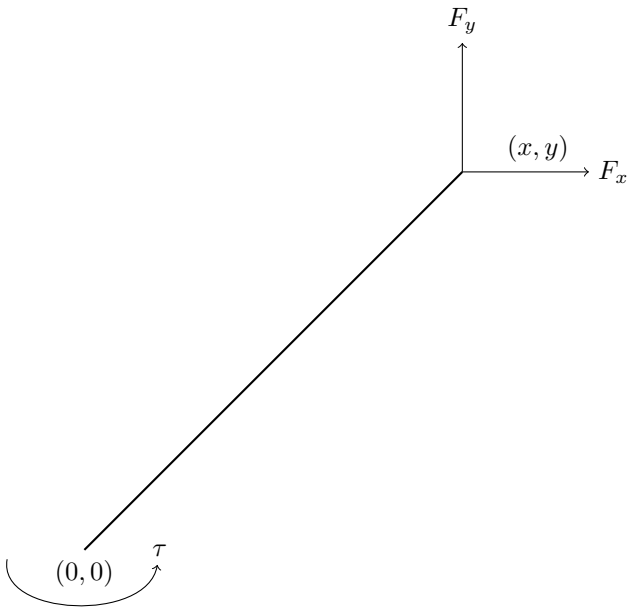


Figure 8: Forces and Torques

$$\mathbf{F} = J^{-T}\boldsymbol{\tau}$$

Here, J^{-T} denotes the transpose of the inverse of the Jacobian (or equivalently, the inverse of the transpose). Thus, for any given end-effector force, the resultant torques can be determined. In order to find the magnitudes (and directions) of the maximal and minimal forces, an eigendecomposition may be performed on the inverse transpose of the Jacobian. The resultant eigenvectors give the directions of maximum and minimum force for a given torque, and the maximum and minimum values correspond to the maximum and minimum (absolute values) of the eigenvalues. This can be seen by considering the general equation for eigenvectors, $Av = \lambda v$. The norm of the right-hand side, which equals the magnitude of the force, is $\sqrt{\lambda^2 v_1^2 + \lambda^2 v_2^2} = \lambda\sqrt{v_1^2 + v_2^2} = \lambda$, since the norm of the eigenvectors returned is 1. These may be analytically obtained by using the conventional formulation:

$$det(J^{-T} - \lambda I) = 0;$$

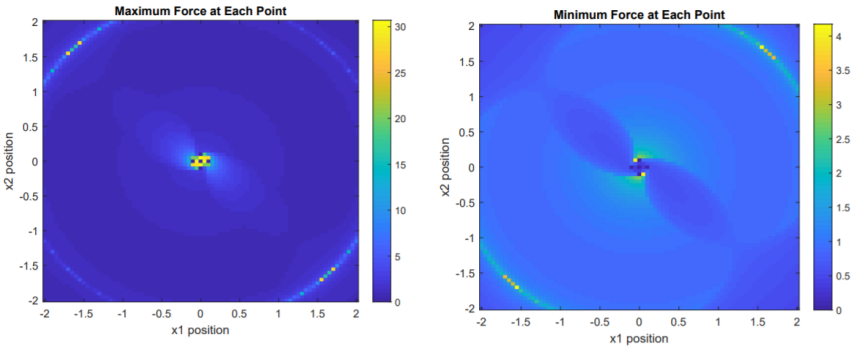
which leads to a characteristic equation of the form:

$$\lambda^2 - \lambda(J_{11}^{-T} + J_{22}^{-T}) + (J_{11}^{-T}J_{22}^{-T} - J_{12}^{-T}J_{21}^{-T}) = \lambda^2 - \lambda(trace(J^{-T})) + det(J^{-T}) = 0$$

Given the complex functional forms of each component of J^{-T}, the actual values for a given coordinate pair and their corresponding eigendecomposition are obtained using Matlab's "eig()" function on the inverse transpose of the Jacobian evaluated at the location.

f) Using the aforementioned eigendecomposition, we obtain the eignevalues with the maximum and minimum absolute values, reflecting the maximum and minimum forces that can be generated at each point along specific directions. These are plotted using the imagesc() function, and yield the following plots. Points are computed along horizontal and vertical directions from -2 to 2 with spacing of 0.05.

From these plots, we can see an inverse relationship between the maximum and minimum force. That is, regions with the largest maximum force (near the center) tend to also have the smallest minimum force, and regions with the smallest maximum force have larger minimum forces. Therefore, near the center, the dynamic range of achievable forces is greater. It should also be noted that the circular regions near the edge denote the breaking down of the above formulation, as these areas cannot be reached using lengths of 1.2 and 1.1 for each link, since this yields a maximum length of only 2.3 when the arm is fully extended. A similar reasoning accounts for the variations in the very center, as the arm cannot reach points less than 0.1 away from the center.



(a) Maximum Force at All Points. (b) Minimum Force at All Points.

Figure 9: Achievable Forces at Each Point

Bonus The second code file displayed in the appendix contains the implementation for 2d combined with a function that plots the force distribution as an ellipse. This later portion is accomplished by decomposing the inverse transpose of the Jacobian into its eigenvector and eigenvalue form. The eigenvalue with the maximum absolute value corresponds to the magnitude of the maximum force, and the equivalent is true for the eigenvalue with the minimum absolute value. The angle of the corresponding maximum direction is found using the arctangent of the x_2 and x_1 eigenvector components, and this is introduced as an offset into the cosine function used to construct the ellipse. The values of the radii along the maximum and perpendicular directions correspond to the maximum and minimum eigenvalues, respectively. An example image resulting from this code is displayed below.

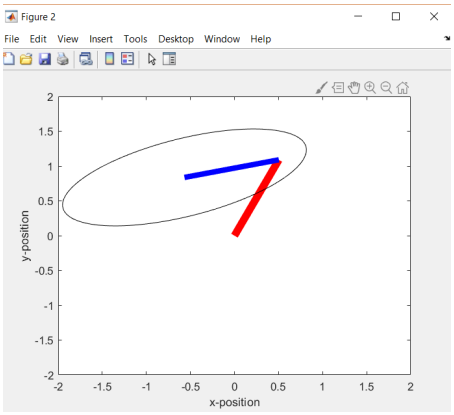


Figure 10: Example screenshot of the interface with ellipse plotting.

```
                        function draw_arm1()
arm_config = 'down';                                                    %change
   between 'up' and 'down' for different configurations
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]);
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);
            l3=sqrt(x^2+y^2);
            C=(l2^2-x^2-y^2-l1^2)/(-2*l1*l3);
            C1=1/sqrt(1-C^2);
            CC=(x^2+y^2-l1^2-l2^2)/(-2*l1*l2);
            C2=1/sqrt(1-CC^2);
            xsum=l3^2;

            if strcmp(arm_config,'up')
                theta1 = atan2(y,x)+acos(C);      % edit these 2 lines to
                   compute the joint angles based on (x,y)
                theta2 = -pi+theta1+acos(CC);
            else
                theta1 = atan2(y,x)-acos(C);      % edit these 2 lines to
                   compute the joint angles based on (x,y)
                theta2 = pi+theta1-acos(CC);
            end

            if isreal(theta1) && isreal(theta2),

                set(redline, 'xdata',[0, l1*cos(theta1)]);  % edit these 2 lines
                    to draw l1
                set(redline, 'ydata',[0, l1*sin(theta1)]);
                set(blueline, 'xdata',[l1*cos(theta1), l1*cos(theta1)+l2*cos(
                   theta2)]); % edit these 2 lines to draw l2
                set(blueline, 'ydata',[l1*sin(theta1), l1*sin(theta1)+l2*sin(
                   theta2)]);
            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
```

```
            function draw_arm2()
arm_config = 'up';                                                  %change
   between 'up' and 'down' for different configurations
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]);
xlabel('x-position')
ylabel('y-position')
ellipse=1;

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

            l3=sqrt(x^2+y^2);               %calculate relevant parameters for
               the Jacobian
            C=(l2^2-x^2-y^2-l1^2)/(-2*l1*l3);
            C1=1/sqrt(1-C^2);
            CC=(x^2+y^2-l1^2-l2^2)/(-2*l1*l2);
            C2=1/sqrt(1-CC^2);
            xsum=l3^2;
            Jtest=zeros(2,2);
            Cder=(1/(l1*l3))+((l2^2-x^2-y^2-l1^2)*1)/(2*l1*l3^3);

            if strcmp(arm_config,'up')
                theta1 = atan2(y,x)+acos(C);      % determine the angles for
                   either configuration
                theta2 = -pi+theta1+acos(CC);
                Jtest(1,1)=-y/xsum-x*C1*Cder;       %also determine the Jacobian
                Jtest(1,2)=x/xsum-y*C1*Cder;
                Jtest(2,1)=-y/xsum-x*C1*Cder+x*C2/(l1*l2);
                Jtest(2,2)=x/xsum-y*C1*Cder+y*C2/(l1*l2);
            else
                theta1 = atan2(y,x)-acos(C);
                theta2 = pi+theta1-acos(CC);
                Jtest(1,1)=-y/xsum+x*C1*Cder;
                Jtest(1,2)=x/xsum+y*C1*Cder;
                Jtest(2,1)=-y/xsum+x*C1*Cder-x*C2/(l1*l2);
                Jtest(2,2)=x/xsum+y*C1*Cder-y*C2/(l1*l2);
            end

            if isreal(theta1) && isreal(theta2),
                set(redline, 'xdata',[0, l1*cos(theta1)]);  %draw the two lines
                set(redline, 'ydata',[0, l1*sin(theta1)]);
```

```
                set(blueline, 'xdata',[l1*cos(theta1), l1*cos(theta1)+l2*cos(
                    theta2)]);
                set(blueline, 'ydata',[l1*sin(theta1), l1*sin(theta1)+l2*sin(
                    theta2)]);
            end
             if x ~= 0 || y ~= 0
                [v,e] = eig(Jtest');                          %construct the
                    eigendecomposition
                e = diag(e);
                tt=0:0.01:2*pi;
                xr = max(abs(e));
                ind = find(max(abs(e)));
                v1 = v(:,ind);
                offset = atan(v1(2)/v1(1));                  %offset from the
                    maximum eigenvector
                yr = min(abs(e));
                xx = xr*cos(tt+offset) + x;
                yy = yr*sin(tt) + y;
                if length(findall(gca,'type','line')) >2     %construct and plot
                    ellipse
                delete(ellipse);
                end
                ellipse =plot(xx,yy,'Color','k');
             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
```