Exercise 4: Identify Position Limits

Define motor specifications

The motor spec sheet provides details about the particular motors you are using. They are rated at 6 V and 12 V. We'll use the 12 V values, but you should get the same results if you use the 6 V values. First, define the rated voltage.

```
V_rated = 12; %Volts
```

The free-run speed is the speed of the motor with no load. At 12 V it is defined as 320 rpm. Convert this to SI units of radians per second to make all calculations consistent. To do so, multiply by the following conversion factors:

$$\frac{2\pi \text{ rad}}{1 \text{ rev}} \text{ and } \frac{1 \text{ min}}{60 \text{ sec}}$$

```
w_free = 300;            %rev/min
w_free = w_free*2*pi/60; %rad/s
```

The spec page also provides the stall torque at the rated voltage as 2220 g-cm. The g in this context refers to gram-force, rather than gram mass. You can look up the conversion factor to convert this torque to SI units of N-m, or you can use the convenient `unitConversionFactor` function from Symbolic Math Toolbox.

```
u = symunit;
TauStall = 2.220; %kg-cm
TauStall = TauStall*double(unitConversionFactor(u.kgf*u.cm,u.Nm)); %N-m
```

Calculate motor constants from specs

In free-run conditions, the torque equation simplifies to $V = \omega_{\text{free}}k$. Use this relationship and the known values for *V* and ω to calculate the motor constant *k*.

```
k = V_rated/w_free; %V-s (or N-m/A)
```

In stall conditions, $\omega = 0$, and the torque equation simplifies to $\tau_{\text{stall}} = \frac{\text{Vk}}{R}$. Use this relationship and the known values for τ, *V*, and *k* to calculate the motor resistance *R*.

```
R = V_rated*k/TauStall; %Ohm
```

Calculate torque limit for available voltage

The whiteboard robot will not be operating at the rated voltage of 12 V. You will instead be using a battery. The battery provided with the kit is rated at 11.1 V. Define this as the available voltage for your battery.

```
V_battery = 11.1; %Volts
```

Consider the stall condition for your whiteboard robot with the given battery and motor constants. Calculate the theoretical stall torque for your battery.

$$\tau_{\text{stall}} = \frac{\text{V}_{\text{battery}}\text{k}}{R}$$

It's recommended not to exceed 30% of the stall torque of the motor. Given this constraint, calculate the maximum allowable torque to apply to the motor.

$$\tau_{\max} = 0.3\tau_{\text{stall}} = 0.3\frac{\text{V}_{\text{battery}}\text{k}}{R}$$

```
TauMax = 0.3*V_battery*k/R %Do not exceed 30% of the stall torque
```

Define whiteboard dimensions

Next, you will identify all positions on the whiteboard and determine which of them are places the robot will be allowed to reach. First, you need to define the size and shape of the whiteboard. You've already defined the Base dimension in a previous task and saved it in the MAT-file RobotGeometry.mat. You also need to define the height of the whiteboard. Measure this height using a tape measure or meterstick, and record it in the `H_board` variable below.

```
H_board = 0.86; %Change this to the height value for your whiteboard (meters)
load RobotGeometry.mat Base
```

Create a grid of all possible board positions

The meshgrid command allows you to create a grid of coodinates. It takes as input arguments an array of x-positions and an array of y-positions. Define the x-positions as an array of values from 0 to Base with a resolution of 1 mm. Define the y-positions from 1mm to `H_board` with a resolution of 1 mm. Do not start y at zero. The tension at y = 0 will always be infinite because the robot would not be supported from above.

```
xarray = 0:0.001:Base;
yarray = 0.001:0.001:H_board;
```

Create the grid of *x* and *y* values from these arrays.

```
[X,Y] = meshgrid(xarray,yarray);
```

Define robot constants for calculating torque

To compute torque at every point on the grid, you need some additional constants related to the robot. These values should be true for your kit, but you can take your own measurements and change the values below if you find them to be inaccurate.

```
r_spool = 0.0045;   %m
Mass = 0.335;       %kg
Weight = Mass*9.81; %N
```

Compute torque at every position

Combine all your equations from geometry, trigonometry, linear mechanics, and rotational mechanics equations to compute the motor torque at every position on the whiteboard. Apply these equations in an element-by-element manner so they apply to all points in the grid at the same time.

From the Pythagorean Theorem:

```
Z1 = sqrt(X.^2 + Y.^2);
Z2 = sqrt((Base-X).^2 + Y.^2);
```

From the Law of Sines:

```
theta1 = asin(X./Z1);
theta2 = asin((Base-X)./Z2);
```

From the free-body diagram and Newton's Third Law:

```
F1 = Weight*sin(theta2)./(2*(cos(theta1).*sin(theta2) + sin(theta1).*cos(theta2)));
F2 = Weight*sin(theta1)./(2*(cos(theta1).*sin(theta2) + sin(theta1).*cos(theta2)));
```

From the definition of tension applied at multiple points:

```
T1 = F1/2;
T2 = F2/2;
```

From the definition of torque:

```
Tau1 = T1*r_spool;
Tau2 = T2*r_spool;
```

Create one grid that stores the largest torque on either of the motors at every point.

```
Tau = max(Tau1,Tau2);
```

Plot torque at every position

Create and label a surface plot to visualize the torque at every point on the whiteboard.

```
surf(X,Y,Tau,'EdgeColor','none')
xlabel('x (meters)')
ylabel('y (meters)')
zlabel('Motor load (Newtons)')
colorbar
```

Eliminate bad regions

Define some constant distances that can be used to identify unreachable regions of the whiteboard where the robot would be too close to the pulleys or too close to the bottom of the whiteboard.

```
pulleyBuffer = 0.05; %Minimum distance to pulley (meters)
```

```
L_arm = 0.075;         %meters
H_robot = 0.16;        %meters
r_pulley = 0.0075;     %meters
```

Create a transparency mask to make the plot transparent in areas that we will not allow the robot to reach.

```
opacity = ones(size(X));
```

Remove places where torque is too high. Use a logical index to define these regions as 0.

```
opacity(Tau > TauMax) = 0;
```

Remove places where robot is too close to the pulleys.

```
nearLeftPulley = X.^2 + Y.^2 <= (L_arm + r_pulley + pulleyBuffer).^2;
nearRightPulley = (Base-X).^2 + Y.^2 <= (L_arm + r_pulley + pulleyBuffer).^2;
opacity(nearLeftPulley | nearRightPulley) = 0;
```

Remove places where robot is too close to the bottom of the board.

```
nearBottom = Y > H_board - H_robot;
opacity(nearBottom) = 0;
```

Plot torque in allowable regions

Now plot the torque again, but use the `imagesc` function to draw it as an image and scale the data to the figure's colormap. Use the opacity variable to only display parts of the image that the robot is allowed to reach. Rescale the colormap so it only goes as high as the torque limit.

```
imagesc(xarray,yarray,Tau,'AlphaData',opacity)
xlabel('x (meters)')
ylabel('y (meters)')
colorbar
ax = gca;
ax.CLim = [0 TauMax];
ax.XLim = [0 Base];
ax.YLim = [0 H_board];
axis equal tight
```

Choose drawable region limits and visualize

After choosing a point as the minimum *xy*-coordinate, enter it in the following section of code. Run this section to visualize the drawable region.

```
pickedX = 0.06; %Change this to your chosen minimum x-value (meters)
pickedY = 0.12; %Change this to your chosen minimum y-value (meters)
xLim = [pickedX Base-pickedX];
yLim = [pickedY H_board-H_robot];
hold on
rectangle('Position',[xLim(1) yLim(1) diff(xLim) diff(yLim)],...
    'EdgeColor','red','LineWidth',3)
hold off
```

Save chosen drawing limits

Save `xLim` and `yLim` for use in future exercises.

```
save WhiteboardLimits.mat xLim yLim
```