Two-Axis Robot Animator MATLAB

Task Objectives:
We will use MATLAB to create a 2-axis robot simulation, find the robots work envelope, and use the Quanser Pendulum to test the simulation. We will create a function that, provided values for two D.O.F. (the two angles on our bot), will animate the robot in the provided orientation/position described by the two D.O.F. inputs. We will eventually add functionality to truly simulate movement (including physical forces acting on the robot), but for now we will be creating an animation function for universal position input. Later we can use this animation function to animate positional values generated by our simulation.

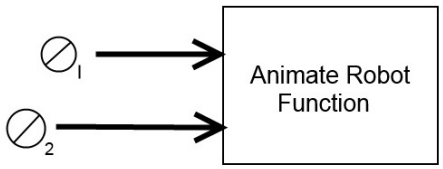

Figure 1: Simple overview of task objective.

Background (Terminology):

Links:
When representing robots, it is easiest to break the structure into multiple links. A link is a single degree of freedom (D.O.F.). It is a variable we can specify.

Types of Links:

1. rotary (can spin)
2. prismatic (sliding, linear fashion movement)

Using links it is possible to break down higher degree joints (like a ball joint) into multiple links/models occupying the same space.

Forward Kinematics:
Definition: Specifying the joint values to accomplish a robot move to a new configuration in space.
This means one is able to tell the location of the end effector by the current state of all D.O.F. in the model. For example, to locate something in 3 dimensions 6 D.O.F. are required (3 for translation, 3 for rotation/orientation). Increasing beyond 6 D.O.F. provides **"redundancy,"** which is the existence of multiple sets/solutions of joint values that lead to the same final robot orientation. Redundancy by definition is not necessary, but it can be useful.

Inverse Kinematics:
Definition: Solving a mathematical model of the robot kinematics to determine the necessary joint values to move the tool to a desired target (frame) in space.
This is the answer to the question... Given a spot in "World Space," what must the D.O.F. values be for the robot?

Singularity:
Definition: A robot singularity occurs when robot axes are redundant (more axes than necessary to cause the same motion) and the robot finds itself in a certain configuration that yields multiple solutions to the same movement. Mathematically, this can be determined by finding the situations where the inverse jacobian formulation (used to relate motion in joint space to motion in World or Cartesian space) becomes singular (determinant = 0). (source: `http://eaal.groups.et.byu.net/html/RoboticsReview/body_robotics_review.html`)

Work Envelope:
Definition: The mapping of all the possible points that the robot's end effector can touch.

Background (Transformation):

Frames

For many situations, it can be easiest to think of each limb of the robot in its own frame of reference. A frame of reference can be thought of as a coordinate space (Cartesian, polar, cylindrical, spherical etc.) with defined unit directions and a defined orientation. Frames for different limbs can overlap, but in the cases where they don't we require a method of transforming the coordinates in one frame to coordinates in another frame. If we can convert coordinates in any frame to a single world frame, then mapping movements of the robot becomes much more intuitive.

For our simple two-axis system, we have 3 frames. They are shown in figure 2.

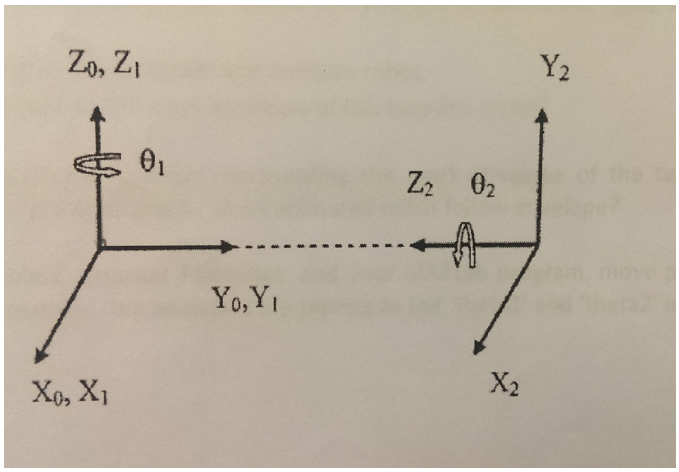

Figure 2: 3 Frames for 2 axis robot.

Frames can differ by 2 factors: location and orientation. These result from translational shifts of the origins of the frames and rotation of certain frames about any axis.

Transforming coordinates (Translation):

Besides the case where two frames are identical, the simplest transformation is one in which two frames are in identical orientations but the origin of one is in a different location than the other. Transforming coordinates between the two frames in this case is as simple as vector addition. If the coordinates of a point in frame 1 must be represented as coordinates in a frame 0, then summing the vector from the origin of frame 0 to the origin of frame 1 with the vector from the origin of frame 1 to the point in frame 1's space... the result is the coordinate of the point represented in frame 0. This simple example is shown in figure 3.

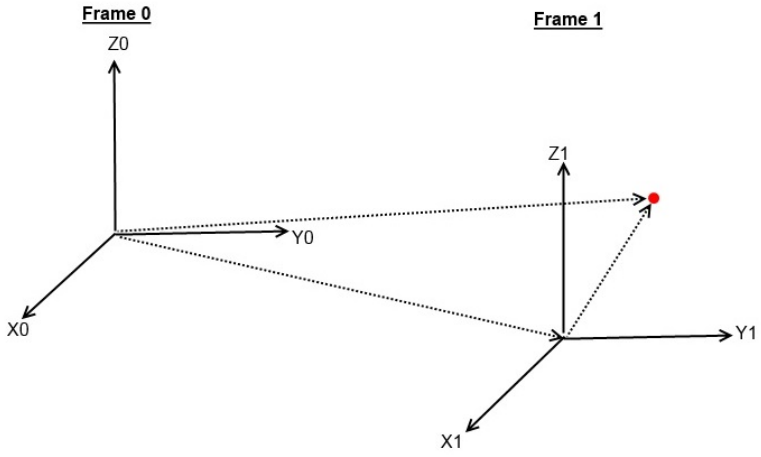

Figure 3: Simple translation example between two identically oriented but translated frames.

Transforming coordinates (Translation & Rotation):
The second form of transformation that must be accounted for is orientation. Frames can be rotated on any axis into different orientations compared to other frames. Although representing translated coordinates in other frames is a relatively easy task even with pen and paper, accounting for cascaded rotations can quickly become infeasible to do by hand with simple math. It is easier to account for both rotation and translation at the same time using Matrix Algebra.

Generalized Homogeneous Transformation Matrix or G.H.T.M.:
Function:
G.H.T.M. (denoted ${}^{i}T_{j}$) is a matrix used to convert coordinates in one frame (frame "j") to a representation in another frame (frame "i"). The use of the G.H.T.M. is as simple as multiplying a vector of coordinates from one frame by the matrix, producing a resultant vector of transformed coordinates. For example, for any point in frame #1, multiplying by ${}^{0}T_{1}$ yields the point represented in frame #0. Such transformation matrices can be multiplied to convert from one frame to a distant frame. For example, ${}^{0}T_{1} *^{1} T_{2} =^{0} T_{2}$ Understanding the make up of the G.H.T.M. is important to understanding its function.

Form:
For 3 dimensional coordinate systems, the G.H.T.M. is a square 4x4 matrix comprised of 3 separate components. These include a Rotation matrix, a coordinate vector [P1,P2,P3] that describes the vector translation between the origins, and a filler vector with a scaling factor.

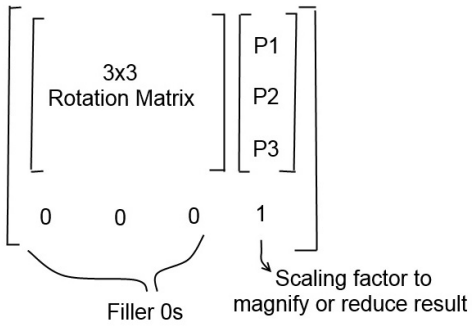

Figure 4: Composition of the Generalized Homogeneous Transformation matrix.

Rotation Vector:
Given an angle of rotation in one frame, the Rotation Matrix (denoted ${}^{i}R_{j}$) uses unit cosine vectors to convert the rotation to a value in another frame. It does this by individually calculating the projections of each axis in one frame onto every axis of the other frame. Take for example the Rotation Matrix ${}^{0}R_{1}$, describing the translation between frames shown in figure 5.

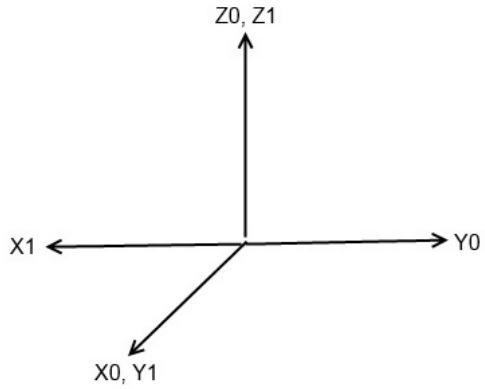

Figure 5: Two differently oriented frames, one rotated w.r.t. the other.

The matrix ${}^{0}R_{1} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ describes the relative rotation between frames 0 and 1 in figure 5.
The 1st column of ${}^{0}R_{1}$ represents how much of the x_1 axis projects onto the 3 axes x_0, y_0, z_0.
The 2nd column of ${}^{0}R_{1}$ represents how much of the y_1 axis projects onto the 3 axes x_0, y_0, z_0.
The 3rd column of ${}^{0}R_{1}$ represents how much of the z_1 axis projects onto the 3 axes x_0, y_0, z_0.
The 3x3 matrix ${}^{i}R_{j}$ makes up ${}^{i}T_{j}[1{:}3,1{:}3]$, as outlined in figure 4.

Denavit-Hartenberg Notation:
For the two axis robot model, the Generalized Homogeneous Transformation Matrix can formed as follows: (Note, $c = cosine$, $s = sine$ and θ is provided in radians.)

Parameter Description:

a_i = distance from $\hat{Z}_i$ to $\hat{Z}_{i+1}$ along $\hat{X}_i$ (a_{i-1} = distance from $\hat{Z}_{i-1}$ to $\hat{Z}_i$ along $\hat{X}_{i-1}$)

α_i = angle from $\hat{Z}_i$ to $\hat{Z}_{i+1}$ about $\hat{X}_i$ (α_{i-1} = angle from $\hat{Z}_{i-1}$ to $\hat{Z}_i$ about $\hat{X}_{i-1}$)

d_i = distance from $\hat{X}_{i-1}$ to $\hat{X}_i$ along $\hat{Z}_i$

θ_i = angle from $\hat{X}_{i-1}$ to $\hat{X}_i$ about $\hat{Z}_i$

Figure 6: Parameters for the two axis G.H.T.M.

Generalized Homogeneous Transformation Matrix:

$$ {}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Figure 7: The populated G.H.T.M. for a two axis robot.

Defining all points of the robot in their initial respective frames:
Simply define the x,y and z coordinates of each point in each limb.

```
%Origin coordinates for frames 0, 1 and 2
origin_0 = [0,0,0];
origin_1 = [0,0,0];
origin_2 = [0,0,0];

%Frame #0 x, y and z coordinates for the limb in frame #0.  (robot base)
%Ex: the 2nd point of the base is (x0(1),y0(1),z0(1)) = (0,0,-10)
x0 = [0 0];
y0 = [0 0];
z0 = [0 -10];

%Frame #1 x, and z coordinates for the limb in frame #1. (middle arm)
x1 = [0 0 0 0 0];
y1 = [0 0 6 6 8];
z1 = [0 -1 -1 0 0];

%Frame #2 x, y and z coordinates for the limb in frame #2.  (swing arm)
x2 = [0 0];
y2 = [0 12];
z2 = [0 0];
```

Drawing the base of the robot in FRAME-0:
The base of the robot is easy to draw as a simple line. It will be stationary, is symmetrical about its axis and its origin is the same as the origin of the 1st arm. Therefore we don't really treat it as its own limb, instead we treat it as a stationary variable that we draw every animation frame.

```
%%%%%%%%%%%%%%%%%%      Task #1:        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%3d plot of the base of the robot in frame 0.
figure;
plot3(x0, y0, z0) %Plot the Base
title('Frame 0, plot of the robot base');
xlabel('x'); ylabel('y'); zlabel('z');
axis([-10 10 -10 10 -10 10]);
grid on;
```

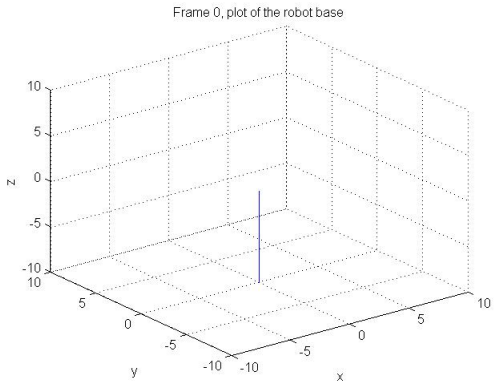

Figure 8: Output of the Task #1 Code, draws the robot base.

Drawing the first link of the robot in FRAME-1:
The first link of the robot has the following points in FRAME-1: (0,0,0),(0,0,-1),(0,6,0), and (0,8,0).

```
%%%%%%%%%%%%%%%%%%%%      Task #2:        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%3d plot of the first link (middle arm) of the robot in frame 1.
figure;
plot3(x1, y1, z1) %Plot the middle arm
title('Frame 1, plot of the middle robot arm');
xlabel('x'); ylabel('y'); zlabel('z');
grid on;
```

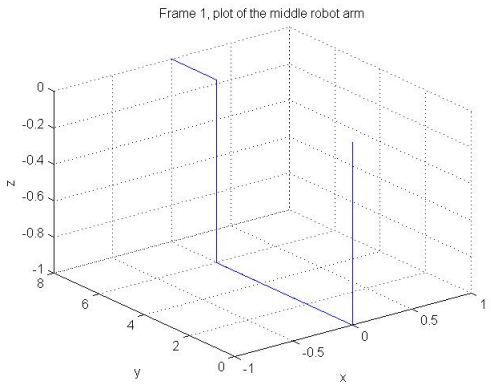

Figure 9: Output of the Task #2 Code, draws the middle arm of the robot.

Similarly, one could draw the isolated swing arm in a similar fashion (but in FRAME-2).

Converting Points Representing the First Link (middle arm):
Here is a function in MATLAB to convert a point (3d coordinate) given in FRAME-1 to a coordinates in FRAME-0 using the G.H.T.M. and an input angle indicating the rotation between the x_0 and x_1 directions about the z_1 direction (as defined in Figure 1).

```
function [pts] = trans0_1(P1x, P1y, P1z, theta1)
%inputs:
    %P1x:       x coordinate of a point in Frame 1
    %P1y:       y coordinate of a point in Frame 1
    %P1z:       z coordinate of a point in Frame 1
    %theta1:    rotation angle between X_0 & X_1 directions about the Z_1 direction

%Transforms the input coordinates (P1x,P1y,P1z) from Frame 1 into
%coordinates for Frame 0, using the Generalized Homogeneous Transformation
%Matrix (G.H.T.M.)

%Actual G.H.T.M.
T0_1 = [cos(theta1) -sin(theta1) 0 0;
        sin(theta1) cos(theta1) 0 0;
        0 0 1 0;
        0 0 0 1];
%Note: column 4, rows 1-3 of the G.H.T.M have value 0 because there is no
%translation between the origin of frames 0 and 1.  ie: the origins are
%identical.


temp = T0_1*[P1x; P1y; P1z; 1];
P0x = temp(1);
P0y = temp(2);
P0z = temp(3);
pts = [P0x,P0y,P0z];    %tranformed points now in Frame 0
end
```

Converting Points Representing the Second Link (swing arm):
Similarly here is a function in MATLAB to convert a point (3d coordinate) given in FRAME-2 to a coordinates in FRAME-1 using the G.H.T.M. and an input angle indicating the rotation between the x_1 and x_2 directions about the z_2 direction (as defined in Figure 1).

```
function [pts] = trans1_2(P2x, P2y, P2z, theta2)

%inputs:
    %P2x:       x coordinate of a point in Frame 2
    %P2y:       y coordinate of a point in Frame 2
    %P2z:       z coordinate of a point in Frame 2
    %theta2:    rotation angle between X_1 & X_2 directions about the Z_2 direction

%Transforms the input coordinates (P2x,P2y,P2z) from Frame 2 into
%coordinates for Frame 1, using the Generalized Homogeneous Transformation
%Matrix (G.H.T.M.)


%define G.H.T.M. parameters between Frame 2 and Frame 1
a_1 = 0;            %distance from Z_1 to Z_2 along X_1
alpha_1 = -pi/2;    %angle from Z_1 to Z_2 about X_1
d_2 = 8;            %distance from X_1 to X_2 along Z_2

%Actual G.H.T.M.
T1_2 = [cos(theta2) -sin(theta2) 0 a_1;
        sin(theta2)*cos(alpha_1) cos(theta2)*cos(alpha_1) -sin(alpha_1) -sin(alpha_1)*d_2;
        sin(theta2)*sin(alpha_1)  cos(theta2)*sin(alpha_1) cos(alpha_1) cos(alpha_1)*d_2;
        0 0 0 1];

temp = T1_2*[P2x; P2y; P2z; 1];
P1x = temp(1);
P1y = temp(2);
P1z = temp(3);
pts = [P1x,P1y,P1z];    %tranformed points now in Frame 1
end
```

Drawing the entire robot in FRAME-0:
Now that there are functions to transform the points of each limb from their respective frame's to FRAME-0, it is possible to draw the entire robot in FRAME-0 (note this requires using both trans0_1 and trans1_2 for converting between FRAME-2 and FRAME-0).

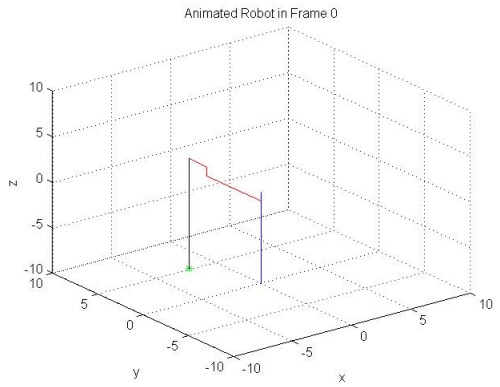

Figure 10: All Robot Limbs plotted in FRAME-0 after transforming their coordinates.

Simulating/Animating the Robot Given Input θ Values:
By defining a vector of values for $theta_1$ and $theta_2$ (the rotations of the middle and swing arms respectively) it is possible to write a function that animates the movement of the robot along the path of values by stepping through each transition.

```
%%%%%%%%%%%%%%%%%%      Task #5:        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Set of desired position values for theta1 (1st row) and theta2 (2nd row)
theta_vec = [linspace(0,6*pi,100); linspace(0,6*pi,100)];
[rows,num_of_theta_vals] = size(theta_vec);


%create a 3d animation of the entire robot given the set of desired
%position values held in theta_vec
figure;
title('Animated Robot in Frame 0');
xlabel('x');ylabel('y');zlabel('z');
for index = 1:num_of_theta_vals
    theta1 = theta_vec(1,index);
    theta2 = theta_vec(2,index);

    axis([-10 10 -10 10 -10 10]);

    %Transform points of the middle arm from Frame 1 to Frame 0
    middle_arm_pts_in_F0 = zeros(3,length(x1));
    for i=1:length(x1)
       middle_arm_pts_in_F0(:,i) = trans0_1(x1(i), y1(i), z1(i),theta1);
    end

    %Transform points of swing arm from Frame 2 to Frame 0
    swing_arm_pts_in_F1 = zeros(3,length(x2));
    swing_arm_pts_in_F0 = zeros(3,length(x2));
    for j=1:length(x2)
       %transform pts from Frame 2 to Frame 1
       swing_arm_pts_in_F1(:,j) = trans1_2(x2(j), y2(j), z2(j),theta2);
       %transform pts from Frame 1 to Frame 0
       swing_arm_pts_in_F0(:,j) = trans0_1(swing_arm_pts_in_F1(1,j),
            swing_arm_pts_in_F1(2,j), swing_arm_pts_in_F1(3,j),theta1);
    end

    %plot all limb pts (base, middle & swing arms) transformed in Frame 0
    hold off;
    % plot base
    plot3([0 0], [0 0], [0 -10]);

    %reset plot parameters
    title('Animated Robot in Frame 0');
    xlabel('x');ylabel('y');zlabel('z');
    axis([-10 10 -10 10 -10 10]);
    grid on;
    hold on;

    %plot middle arm
    plot3(middle_arm_pts_in_F0(1,:)',middle_arm_pts_in_F0(2,:)',
            middle_arm_pts_in_F0(3,:)','r');
    %plot swing arm
    plot3(swing_arm_pts_in_F0(1,:)',swing_arm_pts_in_F0(2,:)',
            swing_arm_pts_in_F0(3,:)','k');
    %plot End effector
    plot3(swing_arm_pts_in_F0(1,length(x2))',swing_arm_pts_in_F0(2,length(x2))',
            swing_arm_pts_in_F0(3,length(x2))','g*');

    pause(0.03); %control speed of animation.
end
```

Defining the Work Envelope:
Given the D.O.F. of the robot, the end effector can reach all points on a partial spherical shell. The radius of this sphere is defined by the hypotenuse of the triangle formed by the middle and swing arms. The points of cutoff for the upper and lower hemispheres is the length of the swing arm.

```
%%%%%%%%%%%%%%%%%%      Task #6:        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
limb_1_length = y1(5);
limb_2_length = y2(2);
radius = sqrt(limb_1_length^2+limb_2_length^2);

num_pts = 40;
[X,Y,Z] = sphere(num_pts);
[rows,cols] = size(Z);
bottom = 1;
top = rows;
for i=1:rows
  if (Z(i,1)<=(-limb_2_length/radius))
        bottom = i;
  end
end

X = X(bottom:end-bottom,:);       %# Keep points in this z value range
Y = Y(bottom:end-bottom,:);       %# Keep points in this z value range
Z = Z(bottom:end-bottom,:);       %# Keep points in this z value range

figure;
mesh(radius.*X,radius.*Y,radius.*Z,'EdgeColor','black');
% surf(radius.*X,radius.*Y,radius.*Z,'EdgeColor','black');
alpha(.1);
axis equal;
```

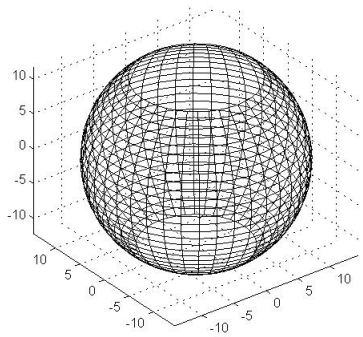

Figure 11: Plotted Work Envelope of the Robot.

Animate the Robot w/ a Work Envelope Overlay:
Plot all the previous elements together.

```
%%%%%%%%%%%%%%%%%%      Task #6 graphed all together:     %%%%%%%%%%%%%%%%%%
%create a 3d animation of the entire robot given desired position values held in theta_vec
figure;
title('Animated Robot in Frame 0');
xlabel('x');ylabel('y');zlabel('z');
for index = 1:num_of_theta_vals
    theta1 = theta_vec(1,index);
    theta2 = theta_vec(2,index);

    axis([-10 10 -10 10 -10 10]);

    %Transform points of the middle arm from Frame 1 to Frame 0
    middle_arm_pts_in_F0 = zeros(3,length(x1));
    for i=1:length(x1)
       middle_arm_pts_in_F0(:,i) = trans0_1(x1(i), y1(i), z1(i),theta1);
    end

    %Transform points of swing arm from Frame 2 to Frame 0
    swing_arm_pts_in_F1 = zeros(3,length(x2));
    swing_arm_pts_in_F0 = zeros(3,length(x2));
    for j=1:length(x2)
       %transform pts from Frame 2 to Frame 1
       swing_arm_pts_in_F1(:,j) = trans1_2(x2(j), y2(j), z2(j),theta2);
       %transform pts from Frame 1 to Frame 0
       swing_arm_pts_in_F0(:,j) = trans0_1(swing_arm_pts_in_F1(1,j),
            swing_arm_pts_in_F1(2,j), swing_arm_pts_in_F1(3,j),theta1);
    end

    %plot all limb pts (base, middle & swing arms) transformed in Frame 0
    hold off;
    % plot base
    plot3([0 0], [0 0], [0 -10]);

    %reset plot parameters
    title('Animated Robot in Frame 0');
    xlabel('x');ylabel('y');zlabel('z');
    axis([-10 10 -10 10 -10 10]);
    grid on;
    hold on;

    %plot middle arm
    plot3(middle_arm_pts_in_F0(1,:)',middle_arm_pts_in_F0(2,:)',
            middle_arm_pts_in_F0(3,:)','r');
    %plot swing arm
    plot3(swing_arm_pts_in_F0(1,:)',swing_arm_pts_in_F0(2,:)',
            swing_arm_pts_in_F0(3,:)','k');
    %plot End effector
    plot3(swing_arm_pts_in_F0(1,length(x2))',swing_arm_pts_in_F0(2,length(x2))',
            swing_arm_pts_in_F0(3,length(x2))','g*');
    %plot the work envelope
    mesh(radius.*X,radius.*Y,radius.*Z,'EdgeColor','black');
    alpha(.1);
    pause(0.03); %control speed of animation.
end
```

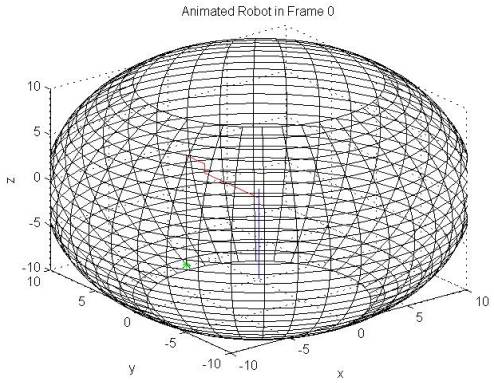

Figure 12: Robot Animation overlaid with the work envelope.

Simulating/Animating the End Effector Path:
By defining a vector of transitional values between

```
%%%%%%%%%%%%%%%%%%      Task #X Follow Movement           %%%%%%%%%%%%%%%%%%

%create a 3d animation of the entire robot given the set of desired
%position values held in theta_vec
figure;
title('Animated Robot in Frame 0');
xlabel('x');ylabel('y');zlabel('z');
End_effector_pts = zeros(3,1);
for index = 1:num_of_theta_vals
    theta1 = theta_vec(1,index);
    theta2 = theta_vec(2,index);

    axis([-10 10 -10 10 -10 10]);

    %Transform points of the middle arm from Frame 1 to Frame 0
    middle_arm_pts_in_F0 = zeros(3,length(x1));
    for i=1:length(x1)
       middle_arm_pts_in_F0(:,i) = trans0_1(x1(i), y1(i), z1(i),theta1);
    end

    %Transform points of swing arm from Frame 2 to Frame 0
    swing_arm_pts_in_F1 = zeros(3,length(x2));
    swing_arm_pts_in_F0 = zeros(3,length(x2));
    for j=1:length(x2)
       %transform pts from Frame 2 to Frame 1
       swing_arm_pts_in_F1(:,j) = trans1_2(x2(j), y2(j), z2(j),theta2);
       %transform pts from Frame 1 to Frame 0
       swing_arm_pts_in_F0(:,j) = trans0_1(swing_arm_pts_in_F1(1,j),
            swing_arm_pts_in_F1(2,j), swing_arm_pts_in_F1(3,j),theta1);
    end

    %plot all limb pts (base, middle & swing arms) transformed in Frame 0
    hold off;
    % plot base
    plot3([0 0], [0 0], [0 -10]);

    %reset plot parameters
    title('Animated Robot in Frame 0');
    xlabel('x');ylabel('y');zlabel('z');
    axis([-10 10 -10 10 -10 10]);
    grid on;
    hold on;

    %plot middle arm
    plot3(middle_arm_pts_in_F0(1,:)',middle_arm_pts_in_F0(2,:)',
           middle_arm_pts_in_F0(3,:)','r');
    %plot swing arm
```

```
    plot3(swing_arm_pts_in_F0(1,:)',swing_arm_pts_in_F0(2,:)',
            swing_arm_pts_in_F0(3,:)','k');
    %plot End effector
    End_effector_pts(1:3,index)=[swing_arm_pts_in_F0(1,length(x2))';
            swing_arm_pts_in_F0(2,length(x2))';swing_arm_pts_in_F0(3,length(x2))'];
    plot3(End_effector_pts(1,:),End_effector_pts(2,:),End_effector_pts(3,:),'g');
    %plot the work envelope
    pause(0.03); %control speed of animation.
end
```

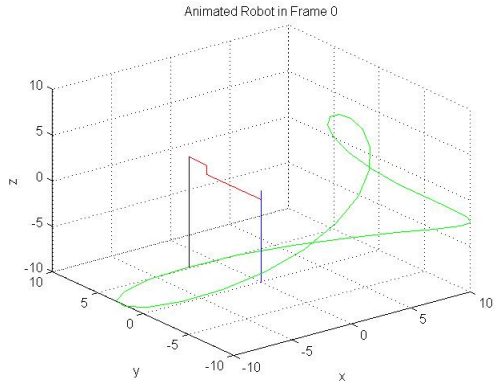

Figure 13: Robot Animation with end effector trace.

Test the Simulation Using Simulink, Quanser Pendulum and Previously Written Functions:
To test the proper functioning of the transformation and simulation code, two tests methods were implemented with prepared values for input θs. First simple sine wave inputs were generated in Simulink and fed into the animation code written previously but altered slightly to animate each instance at a time. For the second test, the pendubot was moved by hand and a Simulink program read the position values from the encoders in the robot's servos indicating the position of each limb. These measured values for θ_1 and θ_2 were fed directly into the same altered animation code as the first test. The output animation matched the movements of the robot that were generated by hand perfectly because the inputs were measured values not calculated. Unfortunately this doesn't allow us to entirely simulate the robot (without measuring values) based off input voltage alone because the code still lacks the dynamics of physics considerations. These additions will come later.