ME 4140: Robotics

Project Talus

December 14, 2017

Group Members:

Aldo Arroyo

Nathan Daniel

Kent Holmquist

Robert Lannom

Ryan Nash

Hayden Pirrera

Andreas Sauter

Shane Terry

Assignment

For this project, we were instructed to pick a robot arm and be able to perform all of the forward and inverse kinematics to produce an animation of TALUS in Matlab.

Background

Talus is a 3D printed humanoid that students at Tennessee Technological University are developing as a showpiece and teaching tool. Most of the printing and assembly of the physical components has been completed. However, at the beginning of this semester very little electrical work and almost no programming had been completed. The programming of an Arduino UNO is necessary in order to get Talus moving. The end goal is to have him as a greeter upon entering the iMakerspace. Talus will be able to greet visitors with a simple “Hello”, or with something as complex as giving them a tour of the iMakerspace. Talus could be able to point at object within the room and say a short description about that object.

STL Files

The STL files comprising TALUS’s body were each downloaded from the inMoov website. These files were created and perfected for 3D printing, and therefore many files contained multiple parts within them. The other problem with the downloaded files was that they could not be modified to look like Talus. In order to get the STL files to work as an actual part, a mesh enabler was used. Once the file was run through the mesh enabler attachment on Inventor it could be used as a solid rigid body. With all the parts converted into a usable rigid body, TALUS could now be assembled on Inventor. The assemblies of each body part consisted of 10 to over 100 different parts. The constraint and grounding features of Inventor were the two most useful ways of making all the parts fit together. The main issue with the converted STL files was that the parts did not have any clear geometry lines, this made it very difficult for Inventor to correctly constrain the parts to one another. This issue made the process of virtually constructing Talus more difficult than other assemblies we’ve done in Inventor. The most difficult part of Talus was the head because of the difficult shape lines that had to be connected and properly aligned.

Forward Kinematics

Using the Denavit-Hartenburg (DH) convention taught in ME4140: Introduction to Robotics, local frames were assigned to each link on TALUS as shown in **Table 1** and **Figure 1**.

Table 1: *Frame Assignments*

Joint Description	Frame
Lower Stomach	F_0
Torso	F_1
Right and Left Shoulder Pivots	F_{R2} , F_{L2}
Right and Left Shoulder Rotators	F_{R3} , F_{L3}
Right and Left Biceps	F_{R4} , F_{L4}
Right and Left Forearms	F_{R5} , F_{L5}
Right and Left Hands	F_{R6} , F_{L6}

Figure 1: *Frame Assignments*

Each arm was observed to have five Degrees of Freedom (DoF), with a 6th being added by the tilting torso joint at F_0. A table of DH parameters was constructed for each arm (**Table 2** and **Table 3**) which defined homogeneous transformation matrices between all adjacent links.These matrices were then multiplied out to create transformations from the base frame to each other joint frame. The section of MATLAB code used to do this is included in **Appendix B**.

Table 2: *Right Arm DH Parameter Table*

i	α	**a**	**d**	θ
1	0^o	a_1	0	θ^*_{R1}
2	90^o	a_2	0	θ^*_{R2}
3	90^o	0	d_3	θ^*_{R3}
4	90^o	a_4	d_4	θ^*_{R4}
5	90^o	0	0	θ^*_{R5}
6	0^o	0	d_6	θ^*_{R6}

Table 3: *Left Arm DH Parameter Table*

i	α	**a**	**d**	θ
1	0^o	a_1	0	θ^*_{L1}
2	90^o	a_2	0	θ^*_{L2}
3	90^o	0	d_3	θ^*_{L3}
4	90^o	a_4	d_4	θ^*_{L4}
5	90^o	0	0	θ^*_{L5}
6	0^o	0	d_6	θ^*_{L6}

Next, the STL files discussed in the previous section were imported into MATLAB using the pre-written *stlread* function (**Appendix I**) and the code shown in **Appendix C**. Using the *patch* function, the vertices of these files could be plotted in a figure, and rotations and translations were applied to them ensure they were aligned with their assigned frames (**Appendix D**). Joint angles were chosen to set TALUS to a “Zero Position”, or a rest configuration (**Table 4**) and used to evaluate the homogeneous transformation matrices. These transformations were applied to each link and the links were plotted in a MATLAB Figure. This section of the code is included in **Appendix E** and the result is shown in **Figure 2**.

Table 4: *"Zero Position" Joint Angles*

Joint	Description	Angle
Joint 1	Torso tilt	115º
Joint 2	Shoulder tilt	155º
Joint 3	Shoulder rotation	90º
Joint 4	Bicep twist	180º
Joint 5	Elbow bend	100º
Joint 6	Wrist twist	0º

Figure 2: *TALUS "Zero Position". Note that the forward kinematics used for positioning links was only valid for the right arm, so only the right arm is show.*

Finally, a while loop was created to increment one or multiple joint variables, recalculate the transformation matrices, apply them to their respective links, plot the links, and then pause for a short time. This creates the impression of motion and allows the visualization of TALUS' forward kinematics operation. This code was included in **Appendix F**.

Inverse Kinematics

Initially, it may seem that the final three joints of each arm may be modelled as Roll-Pitch-Roll spherical wrists; however, upon closer inspection, this is not the case (see side views in **Figure 1**). The axis of Joint 4 does not actually intersect the axis of Joint 5. The two are instead separated by the Denavit-Hartenburg parameter listed as a4; however, in order to simplify the calculations required for inverse kinematics, a4 may be assumed to be 0. Making this assumption allows either of TALUS' arms to be partitioned into two three-joint segments, with the first including the stomach base up to the "wrist center" (in this case, the elbow) and the second encompassing the "wrist center" to the hand.

Assuming a given position and orientation for TALUS' end effector, we obtain the position of the wrist p_c to be equal to:

$$x' = x - d_6 r_{13}$$
$$y' = y - d_6 r_{23}$$
$$z' = z - d_6 r_{33}$$

Setting this equal to final column of the transformation matrix from frame 0 to 4 yields the following three equations:

$$x' = a_2 cos(\theta_1 + \theta_2) + d_3 sin(\theta_1 + \theta_2) + a_1 cos(\theta_1) + d_4 cos(\theta_1 + \theta_2) sin(\theta_3)$$
$$y' = a_2 sin(\theta_1 + \theta_2) - d_3 cos(\theta_1 + \theta_2) + a_1 sin(\theta_1) + d_4 sin(\theta_1 + \theta_2) sin(\theta_3)$$
$$z' = -d_4 cos(\theta_3)$$

The last is easily solvable, resulting in two solution for theta3:

$$\theta_3 = cos^{-1}(-z'/d_4)$$
$$\theta_3 = -cos^{-1}(-z'/d_4)$$

Squaring and adding the first two eliminates theta1 and yields:

$$x'^2 + y'^2 = a_1^2 - d_4^2 cos(\theta_3)^2 + a_2^2 + d_3^2 + d_4^2 + 2a_1 d_3 sin(\theta_2) + 2a_2 d_4 sin(\theta_3) + 2a_1 a_2 cos(\theta_2) + 2a_1 d_4 cos(\theta_2) sin(\theta_3)$$

And using common solution method 6, two solutions for theta2 may be found:

θ_2 = 2*atan((4*a1*d3 + 2*(((2*a1*a2 - d4^2*cos(θ_3)^2 - (x - d6*r13)^2 - (y - d6*r23)^2+ a1^2 + d3^2 + 2*a1*d4*sin(θ_3) + 2*a2*d4*sin(θ_3))*(8*a1*a2 + 4*d4^2*cos(θ_3)^2 + 4*(x - d6*r13)^2 + 4*(y - d6*r23)^2 - 4*a1^2 - 4*d3^2 + 8*a1*d4*sin(θ_3) - 8*a2*d4*sin(θ_3)))/4 + 4*a1^2*d3^2)^(1/2))/(4*a1*a2 + 2*d4^2*cos(θ_3)^2 + 2*(x - d6*r13)^2 + 2*(y - d6*r23)^2 - 2*a1^2 - 2*d3^2 + 4*a1*d4*sin(θ_3) - 4*a2*d4*sin(θ_3)))

θ_2 = 2*atan((4*a1*d3 - 2*(((2*a1*a2 - d4^2*cos(θ_3)^2 - (x - d6*r13)^2 - (y -d6* r23)^2 + a1^2 + d3^2 + 2*a1*d4*sin(θ_3) + 2*a2*d4*sin(θ_3))*(8*a1*a2+4*d4^2* cos(θ_3)^2 + 4*(x - d6*r13)^2 + 4*(y - d6*r23)^2 - 4*a1^2 - 4*d3^2 + 8*a1*d4* sin(θ_3) - 8*a2*d4*sin(θ_3)))/4 + 4*a1^2*d3^2)^(1/2))/(4*a1*a2 + 2*d4^2*cos(θ_3)^2 +2*(x - d6*r13)^2 + 2*(y - d6*r23)^2 - 2*a1^2 - 2*d3^2 + 4*a1*d4*sin(θ_3) - 4*a2* d4*sin(θ_3)))

And using common solution method 6, two solutions for theta1 may be found:

θ_1 = 2*atan((2*((a2*sin(θ_2) - d3*cos(θ_2) + d4*sin(θ_2)*sin(θ_3))^2 + ((a1 - x + d6*r13 + a2*cos(θ_2) + d3*sin(θ_2) + d4*cos(θ_2)*sin(θ_3))*(4*a1 + 4*x - 4*d6*r13 + 4*a2* cos(θ_2) + 4*d3*sin(θ_2) + 4*d4*cos(θ_2)*sin(θ_3)))/4)^(1/2) + 2*d3*cos(θ_2) - 2*a2* sin(θ_2) - 2*d4*sin(θ_2)*sin(θ_3))/(2*a1 + 2*x - 2*d6*r13 + 2*a2*cos(θ_2) + 2*d3* sin(θ_2) + 2*d4*cos(θ_2)*sin(θ_3)))

θ_1 = -2*atan((2*((a2*sin(θ_2) - d3*cos(θ_2) + d4*sin(θ_2)*sin(θ_3))^2 + ((a1 - x + d6*r13 + a2*cos(θ_3) + d3*sin(θ_2) + d4*cos(θ_2)*sin(θ_3))*(4*a1 + 4*x - 4*d6*r13 + 4*a2* cos(θ_2) + 4*d3*sin(θ_2) + 4*d4*cos(θ_2)*sin(θ_3)))/4)^(1/2) - 2*d3*cos(θ_2) + 2*a2* sin(θ_2) + 2*d4*sin(θ_2)*sin(θ_3))/(2*a1 + 2*x - 2*d6*r13 + 2*a2*cos(θ_2) + 2*d3* sin(θ_2) + 2*d4*cos(θ_2)*sin(θ_3)))

Resulting in a total of 8 solutions. These solutions have been validated both visually and with Forward Kinematics. The code to calculate the first three joint angles is found in **Appendix G**.

Next we find the orientation of the wrist:

To accomplish this, first we multiply the transpose of the rotation matrix of frame 0 to 3 and multiply it by the given rotation matrix of frame 0 to 6. This gives us a completely known rotation matrix of frame 3 to 6. This rotation matrix is then set equal to the forward kinematics rotation matrix of frame 3 to 6. Using these matrices we can set each element in each matrix equal to each other and solve for the remaining joint variables.

Theta 5 has two solutions. It is found by setting the element in the 3rd row and 3rd column of each matrix equal to each other and is given as:

$$\theta_5 = cos^{-1}(-(r_{13}cos(\theta_1+\theta_2)sin(\theta_3) - r_{33}cos(\theta_3) + r_{23}sin(\theta_1+\theta_2)sin(\theta_3)))$$
$$\theta_5 = -cos^{-1}(-(r_{13}cos(\theta_1+\theta_2)sin(\theta_3) - r_{33}cos(\theta_3) + r_{23}sin(\theta_1+\theta_2)sin(\theta_3)))$$

Using the common solution technique 5, we can solve for theta 4 as:

θ_4 = atan2((r13*sin(θ_1 + θ_2) - r23*cos(θ_1 + θ_2))/sin(θ_5) , (r33*sin(θ_3) + r13*cos(θ_1 + θ_2)*cos(θ_3) + r23*sin(θ_1 + θ_2)*cos(θ_3))/sin(θ_5)))

Using the common solution technique 5 once again, we can solve for theta 6 as:

θ_6 = atan2((-r12*cos(θ_1 + θ_2)*sin(θ_3) - r32*cos(θ_3) + r22*sin(θ_1 + θ_2)*sin(θ_3))/sin(θ_5), (r11*cos(θ_1 + θ_2)*sin(θ_3) - r31*cos(θ_3) + r21*sin(θ_1 + θ_2)*sin(θ_3))/sin(θ_5)))

This results in two more solutions for the orientation, for a total of 16 solutions to a given Position/Orientation problem for each of TALUS's arms. The solutions for the first three joint variables were checked by applying the "Zero Position" joint angles (**Table 4**) to TALUS and solving the forward kinematics for the end effector position and orientation. Applying inverse kinematics to this position and orientation resulted in approximately correct angles for Joints 1 through 3, but incorrect angles for Joints 4 through 6. The code to calculate the rest of the joint angles is found in **Appendix H**.

Hardware and Control

Controlling the physical robot TALUS presented an entirely different set of challenges that controlling the virtual one. The controllers employed include the Arduino Unos (*image at right*), Megas, and Duemilanove (Dues). These were chosen for their low cost and ease of programming. A basic Arduino program used for testing and demonstrations is shown below.

Each joint is actuated by a servo motor (*image at right*) with a 180º feedback potentiometer. Most of the servos drive a worm gear which indirectly controls the joint, rather than it being tied directly to the servo. This required us to make alterations to the servo so that the feedback potentiometer was tied to the joint motion and not the motor's motion. We had to disassemble the servos, pull out and rewire the potentiometers, and grind off the limiters built into the servos to prevent them from rotating more than 180º. We could then attach the potentiometers directly to the joints. A control system diagram for the unaltered

servos is shown in **Figure 3a**, and a diagram for the altered servos is shown in **Figure 3b**.

Figure 3: *Servo Control Diagrams*

Conclusion

This project had four major topics: assembling and formatting the STL files, solving the forward kinematics and animating a virtual TALUS, solving the inverse kinematics, and assembling the hardware and electronics necessary to get the physical robot moving. STL files now exist for all rigid bodies comprising the joints of TALUS, and forward kinematics may be applied to the right arm to create an animation in joint space. Inverse kinematics may be applied only to the first three joints of the right arm. Future work on the simulation front will include completing forward kinematics for the left arm and head, verifying inverse kinematics for the right wrist, and completing inverse kinematics for the left arm and head.

Significant progress was also made on the hardware challenges associated with getting the physical robot moving. Every joint on the right arm is currently tied to a servo and can be programmed to move, and parts have been ordered for the head and left arm. Future work on this front will include setup on the head and left arm, development of power and signal distribution circuitry, complex Arduino-programmed motion, and the integration of a tablet running MATLAB with the Arduino circuitry in order to perform those complex motions with more predictability. Once these steps have been completed, TALUS will act as a show-piece and interface element to Tennessee Technological University's iMakerSpace and an excellent teaching tool for future robotics courses.

Appendix A: *Setup*

```
%TALUS
format compact
format short

syms theta1
syms thetaR2 thetaR3 thetaR4 thetaR5 thetaR6
syms thetaL2 thetaL3 thetaL4 thetaL5 thetaL6
syms a1 a2 a4 d3 d4 d6
syms one
dtr = pi/180;
```

Appendix B: *Calculate Transformation Matrices*

```
%% Calculate the Transformation Matrices

%Fixed values
a1 =  -820;  %distance from z0 to z1 along x1
a2 =  -100;  %distance from z1 to z2 along x2
a4 =   50; %For forward
%a4 = 0; %For inverse
d3 =  -200;  %distance from x2 to x3 along z2
d4 =  -700;  %distance from x3 to x4 along z3
d6 =  -880;  %distance form x5 to x6 along z5

%DH Tables
DH_R = [0  a1 0   theta1
     90*dtr  a2 0   thetaR2
     90*dtr  0  d3  thetaR3
     90*dtr  a4 d4  thetaR4
     90*dtr  0  0   thetaR5
     0       0  d6  thetaR6];
n_R = length(DH_R(:,1));

DH_L = [0  a1 0   theta1-(70*pi/180)
     90*dtr  a2 0   thetaL2
     90*dtr  0  d3  thetaL3
```

```
      90*dtr   a4 d4   thetaL4
      90*dtr   0  0    thetaL5
      0        0  d6   thetaL6];
n_L = length(DH_L(:,1));

%Transformations between adjacent frames
Ti_R = one*ones(4,4,n_R);
for i = 1:n_R
      Ti_R(:,:,i) = [cos(DH_R(i,4)), -sin(DH_R(i,4))*cos(DH_R(i,1)),
sin(DH_R(i,4))*sin(DH_R(i,1)), DH_R(i,2)*cos(DH_R(i,4))
                  sin(DH_R(i,4)),  cos(DH_R(i,4))*cos(DH_R(i,1)),
-cos(DH_R(i,4))*sin(DH_R(i,1)), DH_R(i,2)*sin(DH_R(i,4))
                  0,               sin(DH_R(i,1)),
cos(DH_R(i,1)),            DH_R(i,3)
                  0,               0                          0,
      1];
end

Ti_L = one*ones(4,4,n_L);
for i = 1:n_L
      Ti_L(:,:,i) = [cos(DH_L(i,4)), -sin(DH_L(i,4))*cos(DH_L(i,1)),
sin(DH_L(i,4))*sin(DH_L(i,1)), DH_L(i,2)*cos(DH_L(i,4))
                 sin(DH_L(i,4)),  cos(DH_L(i,4))*cos(DH_L(i,1)),
-cos(DH_L(i,4))*sin(DH_L(i,1)), DH_L(i,2)*sin(DH_L(i,4))
                  0,               sin(DH_L(i,1)),
cos(DH_L(i,1)),            DH_L(i,3)
                  0,               0                        0,
      1];
end

%Transformations from base frame
T_R = one*ones(4,4,n_R);
T_R(:,:,1) = Ti_R(:,:,1);
for i = 2:n_R
      T_R(:,:,i) = T_R(:,:,i-1)*Ti_R(:,:,i);
      T_R(:,:,i)=(T_R(:,:,i));
end

T_L = one*ones(4,4,n_L);
T_L(:,:,1) = Ti_L(:,:,1);
for i = 2:n_L
      T_L(:,:,i) = T_L(:,:,i-1)*Ti_L(:,:,i);
      T_L(:,:,i)=(T_L(:,:,i));
```

```
End
```

Appendix C: *Import STL Files*

```
%% Import STLs
L0_import = stlread('LowerStomach.stl');L1_import =
stlread('Torso.stl');

LR2_import = stlread('Right_Shoulder1.stl');LR3_import =
stlread('Right_Shoulder2.stl');LR4_import =
stlread('Right_Bicep.stl');
LR5_import = stlread('Right_Forearm.stl');LR6_import =
stlread('Right_Hand.stl');

LL2_import = stlread('Blank.stl');LL3_import =
stlread('Left_Shoulder2.stl');LL4_import =
stlread('Left_Bicep.stl');
LL5_import = stlread('Blank.stl');LL6_import =
stlread('Left_Hand.stl');

LH2_import = stlread('NeckTop.stl');LH3_import =
stlread('Face.stl');LH4_import = stlread('Jaw.stl');
```

Appendix D: *Align Links to Assigned Frames*

```
%% Align Parts
%*********************************************************************
*****
%Transformations for aligned parts
%*********************************************************************
*****
R0 = [-1  0  0; 0  0 -1; 0 -1  0]';o0 = [-370 170 -140];
R1 = [0.4501 -0.8868 -0.105; 0.893 0.447 0.0529; 0 -0.1175 0.9931]; o1
= [265 -740 365];

RR2 = [ 0 1 0;-1 0 0;0 0 1]';oR2 = [-17 -15 -112];
```

```
RR3 = [-0.4226 -0.9062  0.0158; -0.6409  0.3111 -0.7081; 0.6409
-0.2865 -0.7122]';oR3 = [-800 900 2946];
RR4 = [0.9752 -0.0945  0.2; 0.1392  0.9649 -0.2228; -0.1720  0.2451
0.9541]';oR4 = [150 -450 200];
RR5 = [ 0.6691 -0.7431 0; -0.7431 -0.6691 0; 0 0 -1]';oR5 = [190 350
140];
RR6 = [1 0 0; 0 1 0; 0 0 1]';oR6 = [185 130 -400];

RL2 = [ 1 0 0; 0 1 0; 0 0 1]'; oL2 = [0 0 0];
z1 = 18*dtr;
x1 = -48*dtr;
z2 = -29*dtr;
RL3 = [-0.4244 -0.9047 -0.038; 0.6288 -0.3249 0.7064; -0.6515 0.2758
0.7068]'; oL3 = [1492 527.5 -2885];
RL4 = [ 1 0 0; 0 1 0; 0 0 1]'; oL4 = [0 0 0];
RL5 = [ 1 0 0; 0 1 0; 0 0 1]'; oL5 = [0 0 0];
RL6 = [ 1 0 0; 0 1 0; 0 0 1]'; oL6 = [0 0 0];

RH2 = [ 1 0 0; 0 1 0; 0 0 1]'; oH2 = [0 0 0];
RH3 = [ 1 0 0; 0 1 0; 0 0 1]'; oH3 = [0 0 0];
RH4 = [ 1 0 0; 0 1 0; 0 0 1]'; oH4= [0 0 0];
%*****************************************************************
*****
%Ln attached to frame n
%*****************************************************************
*****
L0_o=L0_import; L1_o=L1_import;
LR2_o=LR2_import; LR3_o=LR3_import; LR4_o=LR4_import;
LR5_o=LR5_import; LR6_o=LR6_import;
LL2_o=LL2_import; LL3_o=LL3_import; LL4_o=LL4_import;
LL5_o=LL5_import; LL6_o=LL6_import;
LH2_o=LH2_import; LH3_o=LH3_import; LH4_o=LH4_import;

L0_o.vertices = (R0*L0_o.vertices'+o0')'; L1_o.vertices =
(R1*L1_o.vertices'+o1')';

LR2_o.vertices = (RR2*LR2_o.vertices'+oR2')';
LR3_o.vertices = (RR3*LR3_o.vertices'+oR3')';
LR4_o.vertices = (RR4*LR4_o.vertices'+oR4')';
LR5_o.vertices = (RR5*LR5_o.vertices'+oR5')';
LR6_o.vertices = (RR6*LR6_o.vertices'+oR6')';

LL2_o.vertices = (RL2*LL2_o.vertices'+oL2')';
```

```
LL3_o.vertices = (RL3*LL3_o.vertices'+oL3')';
LL4_o.vertices = (RL4*LL4_o.vertices'+oL4')';
LL5_o.vertices = (RL5*LL5_o.vertices'+oL5')';
LL6_o.vertices = (RL6*LL6_o.vertices'+oL6')';

LH2_o.vertices = (RH2*LH2_o.vertices'+oH2')';
LH3_o.vertices = (RH3*LH3_o.vertices'+oH3')';
LH4_o.vertices = (RH4*LH4_o.vertices'+oH4')';

%Copies Lno into Ln
L0=L0_o; L1=L1_o;
LR2=LR2_o; LR3=LR3_o; LR4=LR4_o; LR5=LR5_o; LR6=LR6_o;
LL2=LL2_o; LL3=LL3_o; LL4=LL4_o; LL5=LL5_o; LL6=LL6_o;
LH2=LH2_o; LH3=LH3_o; LH4=LH4_o;
```

Appendix E: *Plotting Joints in "Zero Position"*

```
%% Use Forward Kinematics to set to Zero Position
close all
figure(1)
theta1 = 115*dtr;
thetaR2 = 155*dtr;thetaR3 = 90*dtr;thetaR4 = 180*dtr;thetaR5
=100*dtr;thetaR6 = 0;
thetaL2 = 205*dtr;thetaL3 = -90*dtr;thetaL4 = 180*dtr;thetaL5 =
100*dtr;thetaL6 = 0;

T_1 = eval(T_R(:,:,1));L1.vertices =
(T_1(1:3,1:3)*L1_o.vertices'+T_1(1:3,4))';

T_R2 = eval(T_R(:,:,2));LR2.vertices =
(T_R2(1:3,1:3)*LR2_o.vertices'+T_R2(1:3,4))';
T_R3 = eval(T_R(:,:,3));LR3.vertices =
(T_R3(1:3,1:3)*LR3_o.vertices'+T_R3(1:3,4))';
T_R4 = eval(T_R(:,:,4));LR4.vertices =
(T_R4(1:3,1:3)*LR4_o.vertices'+T_R4(1:3,4))';
T_R5 = eval(T_R(:,:,5));LR5.vertices =
(T_R5(1:3,1:3)*LR5_o.vertices'+T_R5(1:3,4))';
T_R6 = eval(T_R(:,:,6));LR6.vertices =
(T_R6(1:3,1:3)*LR6_o.vertices'+T_R6(1:3,4))';

T_L2 = eval(T_L(:,:,2));LL2.vertices =
(T_L2(1:3,1:3)*LL2_o.vertices'+T_L2(1:3,4))';
```

```
T_L3 = eval(T_L(:,:,3));LL3.vertices =
(T_L3(1:3,1:3)*LL3_o.vertices'+T_L3(1:3,4))';
T_L4 = eval(T_L(:,:,4));LL4.vertices =
(T_L4(1:3,1:3)*LL4_o.vertices'+T_L4(1:3,4))';
T_L5 = eval(T_L(:,:,5));LL5.vertices =
(T_L5(1:3,1:3)*LL5_o.vertices'+T_L5(1:3,4))';
T_L6 = eval(T_L(:,:,6));LL6.vertices =
(T_L6(1:3,1:3)*LL6_o.vertices'+T_L6(1:3,4))';

patch(L0, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0, ...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
patch(L1, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0, ...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);

patch(LR2, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0, ...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
patch(LR3, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0, ...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
patch(LR4, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0, ...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
patch(LR5, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0, ...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
patch(LR6, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0, ...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
%{
patch(LL2, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0, ...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
patch(LL3, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0, ...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
patch(LL4, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0, ...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
patch(LL5, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0, ...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
patch(LL6, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0, ...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
%}
axis equal
light('Position', [-300 300 300] , 'Style', 'Infinite');
xlabel('x'), ylabel('y'), zlabel('z')
```

Appendix F: *Animation With Forward Kinematics*

```
%% Animate

n_R = 2;
t = 0.001;
inc = 1;
sgn2 = 1;sgn3 = 1;sgn4 = 1;sgn5 = -1;sgn = 1;
while(1)
      figure(2)
      theta1 = theta1 + 15*pi/180;

      if (thetaR2 <= 130*dtr)||(thetaR2 >= 160*dtr)
      sgn2 = -sgn2;
      end
      thetaR2 = thetaR2 + inc*sgn2*dtr;
      if (thetaR3 <= 0*dtr)||(thetaR3 >= 90*dtr)
      sgn3 = -sgn3;
      end
      thetaR3 = thetaR3 + inc*sgn3*dtr;
      if (thetaR4 <= 70*dtr)||(thetaR4 >= 160*dtr)
      sgn4 = -sgn4;
      end
      thetaR4 = thetaR4 + inc*sgn4*dtr;
      if (thetaR5 <= 100*dtr)||(thetaR5 >= 160*dtr)
      sgn5 = -sgn5;
      end

      T_1 = eval(T_R(:,:,1));L1.vertices =
(T_1(1:3,1:3)*L1_o.vertices'+T_1(1:3,4))';
      T_R2 = eval(T_R(:,:,2));LR2.vertices =
(T_R2(1:3,1:3)*LR2_o.vertices'+T_R2(1:3,4))';
      T_R3 = eval(T_R(:,:,3));LR3.vertices =
(T_R3(1:3,1:3)*LR3_o.vertices'+T_R3(1:3,4))';
      T_R4 = eval(T_R(:,:,4));LR4.vertices =
(T_R4(1:3,1:3)*LR4_o.vertices'+T_R4(1:3,4))';
      T_R5 = eval(T_R(:,:,5));LR5.vertices =
(T_R5(1:3,1:3)*LR5_o.vertices'+T_R5(1:3,4))';
      T_R6 = eval(T_R(:,:,6));LR6.vertices =
(T_R6(1:3,1:3)*LR6_o.vertices'+T_R6(1:3,4))';
```

```
      %{
      T_L2 = eval(T_L(:,:,2));LL2.vertices =
(T_L2(1:3,1:3)*LL2_o.vertices'+T_L2(1:3,4))';
      T_L3 = eval(T_L(:,:,3));LL3.vertices =
(T_L3(1:3,1:3)*LL3_o.vertices'+T_L3(1:3,4))';
      T_L4 = eval(T_L(:,:,4));LL4.vertices =
(T_L4(1:3,1:3)*LL4_o.vertices'+T_L4(1:3,4))';
      T_L5 = eval(T_L(:,:,5));LL5.vertices =
(T_L5(1:3,1:3)*LL5_o.vertices'+T_L5(1:3,4))';
      T_L6 = eval(T_L(:,:,6));LL6.vertices =
(T_L6(1:3,1:3)*LL6_o.vertices'+T_L6(1:3,4))';
      %}
      clf(2)
      patch(L0, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0,
...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
      patch(L1, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0,
...
            'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
      patch(LR2, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0,
...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
      patch(LR3, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0,
...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
      patch(LR4, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0,
...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
      patch(LR5, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0,
...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
      patch(LR6, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0,
...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
      %{
      patch(LL2, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0,
...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
      patch(LL3, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0,
...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
      patch(LL4, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0,
...
```

```
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
      patch(LL5, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0,
...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
      patch(LL6, 'FaceAlpha', 1.0, 'EdgeColor', 'None','EdgeAlpha',1.0,
...
      'FaceColor', 0.7*[1,.5,0], 'DiffuseStrength', 1.0);
      %}
      axis equal
      light('Position', [-300 300 300] , 'Style', 'Infinite');
      xlabel('x'), ylabel('y'), zlabel('z')

      pause(t)
end
```

Appendix G: *Inverse Kinematics Solution for Right Arm*

```
%% Inverse
%Given T0_6
r11 = 0; r12 = -1; r13 = 0;
r21 = -0.9848; r22 = 0; r23 = -0.1736;
r31 = 0.1736; r32 = 0; r33 = -0.9848;
x = 546.547; y = 209.638; z = 816.6308;
T_spec = [r11 r12 r13 x;
          r21 r22 r23 y;
          r31 r32 r33 z;
          0 0 0 1];

%End effector position
p_ee = [x;y;z];

%Wrist center position
p_c = p_ee - T_spec(1:3,3)*d6;
 xp = p_c(1);
 yp = p_c(2);
 zp = p_c(3);

%Joint 4 is wrist center
eqn1 = xp == T(1,4,4);
eqn2 = yp == T(2,4,4);
eqn3 = zp == T(3,4,4);

%Since eqn3 has only theta3, we can solve for it
```

```
theta3_1 = acos(-zp/d4)*180/pi;
theta3_2 = -acos(-zp/d4)*180/pi;

 if (theta3_1<=105)&&(theta3_1>=-105)&&(imag(theta3_1)==0)
      theta3 = theta3_1;
 elseif (theta3_2<=105)&&(theta3_2>=-105)&&(imag(theta3_2)==0)
      theta3 = theta3_2;
 else
      error('Position/Orientation not reachable or error in Inverse
Kinematics.');
 end

%Square and add eqn1 and eqn2 to leave only theta2
eqn4 = T(1,4,4);
eqn5 = T(2,4,4);
eqn6 = simplify(eqn4^2+eqn5^2);
eqn7 = xp^2+yp^2 == eqn6;

%Apply CST #6 to find theta2
a=2*a1*d3;
b=2*a1*a2+2*a1*d4*sin(theta3);
c=xp^2+yp^2-a1^2-d3^2+d4^2*cos(theta3)^2-2*a2*d4*sin(theta3);
t1 = (2*a+sqrt(4*a^2-4*(c+b)*(c-b)))/(2*(c+b));
t2 = (2*a-sqrt(4*a^2-4*(c+b)*(c-b)))/(2*(c+b));

%Theta2 has two answers since we used the quadratic formula
theta2_1 = 2*atan(t1)*180/pi;
theta2_2 = 2*atan(t2)*180/pi;

 if (theta2_1<=170)&&(theta2_1>=115)&&(imag(theta2_1)==0)
      theta2 = theta2_1;
 elseif (theta2_2<=170)&&(theta2_2>=115)&&(imag(theta2_2)==0)
      theta2 = theta2_2;
 else
      error('Position/Orientation not reachable or error in Inverse
Kinematics.');
 end

%Apply CST #6 to find theta1
a=-a2*sin(theta2)+d3*cos(theta2)-d4*sin(theta2)*sin(theta3);
b=a2*cos(theta2)+d3*sin(theta2)+a1+d4*sin(theta3)*cos(theta2);
c=xp;
t1 = (2*a+sqrt(4*a^2-4*(c+b)*(c-b)))/(2*(c+b));
```

```
t2 = (2*a-sqrt(4*a^2-4*(c+b)*(c-b)))/(2*(c+b));

theta1_1 = 2*atan(t1)*180/pi;
theta1_2 = 2*atan(t2)*180/pi;

 if (theta1_1<=145)&&(theta1_1>=85)&&(imag(theta1_1)==0)
      theta1 = theta1_1;
 elseif (theta1_2<=145)&&(theta1_2>=85)&&(imag(theta1_2)==0)
      theta1 = theta1_2;
 else
      error('Position/Orientation not reachable or error in Inverse
Kinematics.');
 end
```

Appendix H: *Inverse Kinematics Solution for Right Wrist*

```
%Solving for the wrist
R0_3 = T(1:3,1:3,3);
R0_6 = T_spec(1:3,1:3);
R3_6knowns = simplify(transpose(R0_3)*R0_6)
T3_6 = simplify(Ti(:,:,4)*Ti(:,:,5)*Ti(:,:,6));
R3_6FK = T3_6(1:3,1:3)
eqn8 = R3_6knowns(3,3) == R3_6FK(3,3)

%This implies theta5 is:
theta5_1 = acos(-R3_6knowns(3,3));
theta5_1 = eval(theta5_1)*180/pi;
theta5_2 = -acos(-R3_6knowns(3,3));
theta5_2 = eval(theta5_2)*180/pi;

 if (theta5_1<=145)&&(theta5_1>=60)&&(imag(theta5_1)==0)
      theta5 = theta5_1;
 elseif (theta5_2<=145)&&(theta5_2>=60)&&(imag(theta5_2)==0)
      theta5 = theta5_2;
 else
      error('Position/Orientation not reachable or error in Inverse
Kinematics.');
 end

%Using CST #5 to solve for theta4
theta4 =
atan2(R3_6knowns(2,3)/sin(theta5),R3_6knowns(1,3)/sin(theta5))
theta4 = eval(theta4)
```

```
%Using CST #5 to solve for theta6
theta6 =
atan2(-R3_6knowns(3,2)/sin(theta5),R3_6knowns(3,1)/sin(theta5))
theta6 = eval(theta6)
```

***Appendix I** "stlread" Function*

```
function varargout = stlread(file)
% STLREAD imports geometry from an STL file into MATLAB.
%    FV = STLREAD(FILENAME) imports triangular faces from the binary
STL file
%    indicated by FILENAME, and returns the patch struct FV, with
fields 'faces'
%    and 'vertices'.
%
%    [F,V] = STLREAD(FILENAME) returns the faces F and vertices V
separately.
%
%    [F,V,N] = STLREAD(FILENAME) also returns the face normal vectors.
%
%    The faces and vertices are arranged in the format used by the
PATCH plot
%    object.

%    Eric C. Johnson, 11-Dec-2008
%    Copyright 1999-2008 The MathWorks, Inc.

    if ~exist(file,'file')
    error(['File ''%s'' not found. If the file is not on MATLAB''s
path' ...
                ', be sure to specify the full path to the file.'],
file);
    end

    fid = fopen(file,'r');
    if ~isempty(ferror(fid))
    error(lasterror); %#ok
    end

    M = fread(fid,inf,'uint8=>uint8');
    fclose(fid);
```

```
      if( isbinary(M) )
      [f,v,n] = stlbinary(M);
      else
      [f,v,n] = stlascii(M); % Not currently supported
      end

      varargout = cell(1,nargout);
      switch nargout
      case 2
            varargout{1} = f;
            varargout{2} = v;
      case 3
            varargout{1} = f;
            varargout{2} = v;
            varargout{3} = n;
      otherwise
            varargout{1} = struct('faces',f,'vertices',v);
      end

end

function [F,V,N] = stlbinary(M)

      F = [];
      V = [];
      N = [];

      if length(M) < 84
      error('MATLAB:stlread:incorrectFormat', ...
            'Incomplete header information in binary STL file.');
      end

      % Bytes 81-84 are an unsigned 32-bit integer specifying the
number of faces
      % that follow.
      numFaces = typecast(M(81:84),'uint32');
      %numFaces = double(numFaces);
      if numFaces == 0
      warning('MATLAB:stlread:nodata','No data in STL file.');
      return
      end
```

```
      T = M(85:end);
      F = NaN(numFaces,3);
      V = NaN(3*numFaces,3);
      N = NaN(numFaces,3);

      numRead = 0;
      while numRead < numFaces
      % Each facet is 50 bytes
      %  - Three single precision values specifying the face normal
vector
      %  - Three single precision values specifying the first vertex
(XYZ)
      %  - Three single precision values specifying the second vertex
(XYZ)
      %  - Three single precision values specifying the third vertex
(XYZ)
      %  - Two unused bytes
      i1    = 50 * numRead + 1;
      i2    = i1 + 50 - 1;
      facet = T(i1:i2)';

      n  = typecast(facet(1:12),'single');
      v1 = typecast(facet(13:24),'single');
      v2 = typecast(facet(25:36),'single');
      v3 = typecast(facet(37:48),'single');

      n = double(n);
      v = double([v1; v2; v3]);

      % Figure out where to fit these new vertices, and the face, in
the
      % larger F and V collections.
      fInd  = numRead + 1;
      vInd1 = 3 * (fInd - 1) + 1;
      vInd2 = vInd1 + 3 - 1;

      V(vInd1:vInd2,:) = v;
      F(fInd,:)        = vInd1:vInd2;
      N(fInd,:)        = n;

      numRead = numRead + 1;
      end
```

```
end

function [F,V,N] = stlascii(A) %#ok
      warning('MATLAB:stlread:ascii','ASCII STL files currently not
supported.');
      F = [];
      V = [];
      N = [];
end

function tf = isbinary(A)
% ISBINARY determines if an STL file is binary or ASCII.

      % Look for the string 'endsolid' near the end of the file
      if isempty(A) || length(A) < 16
      error('MATLAB:stlread:incorrectFormat', ...
            'File does not appear to be an ASCII or binary STL file.');
      end

      % Read final 16 characters of M
      i2  = length(A);
      i1  = i2 - 15;
      str = char( A(i1:i2)' );

      k = strfind(lower(str), 'endsolid');
      if ~isempty(k)
      tf = false; % ASCII
      else
      tf = true;  % Binary
      end
end
```