Foundations of Robotics Final Project Report

Greta Perez-Haiek, Aadhav Sivakumar

December 19th, 2024

1 Introduction

The scope of this project includes mapping the kinematics of a 6-degree of freedom UFACTORY XARM robot using MATLAB 2024a, which could be later defined in simulation. A situation involving arming the XARM with a sword of previously defined length (with the tip of the blade as the end-effector) will be utilized as the focus. By allowing an avenue for individuals to directly control the angle of each joint, or find the angles given a coordinate, complete control of the robot (and it's weapon) can be simulated.

2 Forward Kinematics

Forward Kinematics is the process of calculating the location and orientation of the end effector of the robot given the angles at all of the individual joints. The Denavit-Hartenberg (DH) table is a simple way of describing a robot's links and joints. The DH Table of the XARM6 is made available from the reference documents on the product website for the XARM [1]. It is important to note that the theta values for the second and third joints have a pre-defined "off-set," a number provided for by UFACTORY to allow for the most efficient sense of control and accuracy.

Kinematics	theta (rad)	d (mm)	alpha (rad)	a (mm)	offset (rad)
Joint1	0	267	0	0	0
Joint2	0	0	-pi/2	0	T2_offset
Joint3	0	0	0	a2	T3_offset
Joint4	0	342.5	-pi/2	77.5	0
Joint5	0	0	pi/2	0	0
Joint6	0	97	-pi/2	76	0

Figure 1: DH parameters of the XARM 6 [1]

The Theta Column, while variable, is initially set as zero in the DH Parameters. This describes the "home configuration" of the robot. Theta and Alpha, as a variable, are listed in radians, while "A" and "D" are in meters.

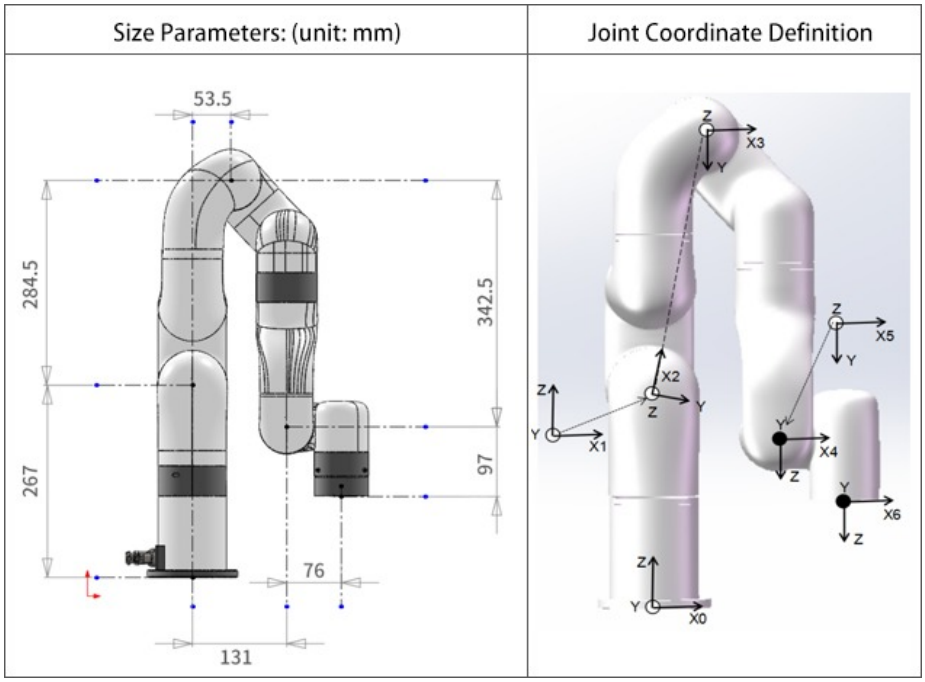

Figure 2: Home Configuration of the XARM 6

To test the Forward Kinematics Package (See Section 6), it is recommended to input zero for all of the theta angles and ensure that the program provides the correct x, y, and z configurations corresponding with the home positions. These values are 207 mm in the x direction, 0 mm in the y direction, and 112 mm in the z direction. There is also a 180 degree rotation in the x direction, which is represented in the obtained transformation matrix provided for by the Forward Kinematics Package.

```
1.0000         0         0    0.2070
     0   -1.0000         0         0
     0         0   -1.0000    0.1120
     0         0         0    1.0000
```

Figure 3: Resulting Transformation Matrix from Home Orientation

Once this was functioning, other theta values could be tested. By changing only the first theta associated with the swivel base of the robot, it was possible to verify that the transformation matrix was updating correctly in response to changes in the joint angle. This resulted in the robot moving in a circular pattern, allowing the end effector orientation to be quickly calculated and confirmed as an accurate representation. The process was repeated for each joint, with slight adjustments to verify that they had the intended effect on the end effector position and orientation. It is possible, if given all six thetas at the same time (representing the joint angles for all 6 joints), for the Forward Kinematics Package to provide an origin-to-end-effector transformation matrix.

3 Inverse Kinematics

Inverse kinematics involves determining the specific joint angles required to achieve a desired end effector position and orientation. This process is significantly more challenging than Forward Kinematics because, while providing joint angles directly leads to a single solution for the end effector position, solving in reverse can result in potentially infinite joint configurations, or none at all.

To calculate the thetas given a desired x, y, and z position (relative to the frame of the Origin, which in this case, is the center of the bottom of the base of the robot), the symbolic representation of the positions contained inside the origin-to-end-effector transformation matrix is converted into an equation by equating it to the desired point, which is then input into a solver. The numerical solver VPASOLVE in MATLAB outputs all the theta values that satisfy the given equations. the Inverse Kinematics Package, however, selects a random one and outputs it. While the output thetas are indeed a solution, the resulting solution includes an arbitrary orientation.

Including Roll, Pitch, and Yaw orientation would allow VPASOLVE to locate a more accurate theta group, however, solving for all three angles simultaneously results in the solver taking an exorbitant amount of time due to the large number of symbolic variables being processed simultaneously. The current Inverse Kinematics Package, for this reason, excludes Roll, Pitch, and Yaw from calculations and demonstrations... but, if desired, it can be activated easily (See Section 6).

4 Visualization

To visualize the robot given all angular joint configurations, MATLAB 2024a provides Unified Robotic Description Format (URDF) manipulation, adjustment, and most importantly, visualization. The XARM6 URDF, which includes all the information needed to fully simulate the robot in various simulation environments, was obtained from PyBullet [2], an open source robotics real-time physics simulation project which specializes on creating and simulation various popular robotic models. Utilizing IMPORTROBOT would create the URDF

data as a "Rigid Body Tree," which would later be adjusted to accommodate an extenuous link... the sword. This sword's theta, unlike other links, will not be adjusted, as the sword will be attached firmly and vertically to the previous joint. By setting the joint position variables to the desired theta angles for the arm joints, the robot's displayed position is updated accordingly, and henceforth, displayed for the user [3]. Passing these theta values into the Visualization Package will print a display of the URDF in the desired position (See Image 4) where the X, Y, and Z units are in Meters.

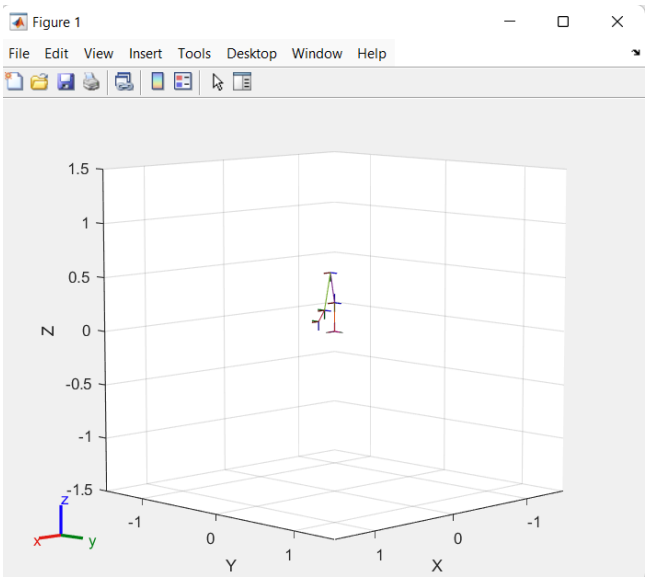

Figure 4: The XARM6 URDF Visualized

5 EXTRA CREDIT: Workspace Computation

To enhance the visualization aspect of the simulation project, the Workspace Package offers an opportunity to simulate and visualize the *reachable* and *dexterous* workspace with the help of the XARM URDF.

The reachable workspace includes all the points that the robot can reach, regardless of the end effector orientation. The area outside this workspace represents points the end effector can never reach, regardless of the joint angles. Trying to pass a theta value into an orientation that is not within the reachable workspace will result in a Warning being displayed to the user (See Section 6). Within the reachable workspace, there are also semi-dexterous solutions, where a point can be reached in multiple ways, as well as the fully dexterous workspace. The fully dexterous workspace consists of points the robot can reach from every possible angle, making it significantly smaller than the reachable workspace.

A voxel point was generated of the entire workspace of the XARM, and to better represent the workspace, a reachability index was assigned to each voxel-point, indicating how many different orientations the robot can achieve at that location. This index can be visualized using a color map: The more blue a voxel-point is, the more solutions it contains, while the more red a voxel-point is, the less solutions it contains. Deep blue voxel-points indicate the dexterous workspace: fully infinite solutions. Meanwhile, a deep red indicates a single solution. No points indicate no solutions (See Figure 5).

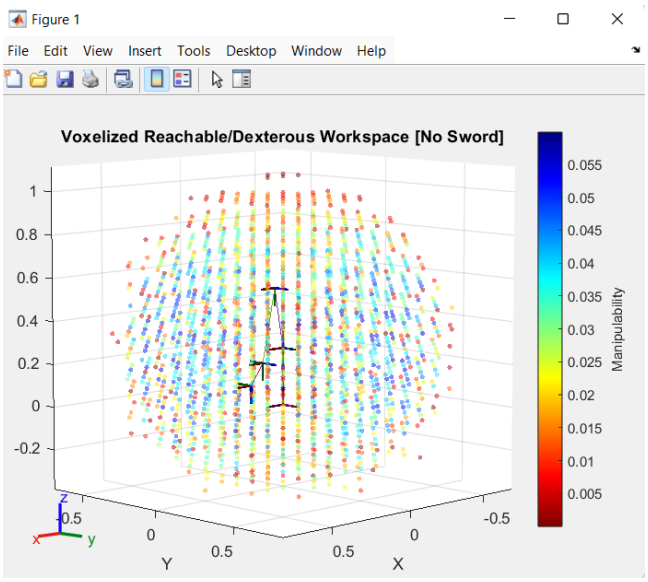

Figure 5: Workspace of the arm with Manipulability Plotting

In order to see the workspace better, a user can drag their mouse alongside the Voxel-Map Sphere to shift it's orientation. This way, a user can see whether the end-effector of the robot would be contained in their desirable workspace. In Figure 6, for example, the XARM was positioned using the Visualization Package using the following thetas: [0.2, 0.8, 3.0, 0.0, 0.2, 0.0] in degrees, where each corresponding theta number is associated with the corresponding joint number. When the voxel-map is oriented in a way that the end-effector aligns with a point, it can be seen that this specific theta orientation would generate a position within the reachable workspace, but not necessarily a dexterous one.

6 A Helpful How-To Guide

Our package contains 5 MATLAB files, and 1 URDF File: Main.m, Inverse_kinematics.m, Forward_kinematics.m, Visualization.m, and Workspace.m, and xarm6_robot.urdf.

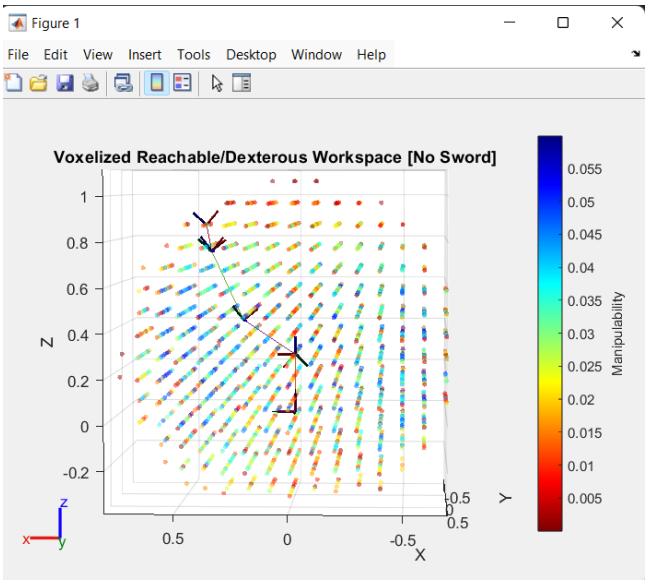

Figure 6: Visualizing Dexterity: An Example

You must download all of these files and make sure they are all contained in the same folder. Then, open up MATLAB 2024a, and open the Main.m file for editing in the programming editor of your choice.

In Main's "Forward Kinematics" Section, on line 15, you'd have the opportunity to choose your theta values. Feel free to modify the theta values (in radians) in the example array "fk_theta_val" to those that you see fit.

```
%% For Forward Kinematics...
%Specify your "i" thetas, for each "i" joint (from Joints 1, 2... 5, 6).
fk_theta_val = [0.2, 0.8, 3.0, 0.0, 0.2, 0.0]; %forward kinematics theta values

%CHOOSE YOUR WEAPON!!!! >:D
%(Choose the sword that you'd like the XARM to wield in honorable combat...
sorted by length in METERS)
%(You can also choose "no sword," but where is the fun in that?)

no_sword = 0; %fist-fighting, of Paleolithic origin, I presume.

hudiedao = 0.30; %Qing Dynasty Era Chinese Origin

sica = 0.35; %Ancient Roman/Dacian Origin

machete = 0.45; %Colonial Central American Origin
```

```
katana = 0.6; %Feudal Japanese Origin

tachi = 0.75; %Koto period Japanese Origin

ZhanmaDao = 0.95; %Song Dynasty Era Chinese origin

rapier = 1.05; %Medieval Spanish Origin

longsword = 1.15; %Rennaisance Era Germanic Origin

claymore = 1.25; %Late Medieval Scottish Origin

sword_of_choice = no_sword; %%INSERT the NAME of the sword of
your choosing
HERE! [i.e, machete]
```

Once you choose your theta values... now is time to choose your weapon of choice! There are plentiful of swords to choose from, varying in length and origin. On line 41, type in the name of the sword that you wish the XARM to wield. You could also choose "no sword" as well, if that is what you desire. This sword will be oriented vertically (similar to the way that the AAB Robot in Figure 7 is holding a Katana).

To modify the "Inverse Kinematics" section, feel free to adjust the Yaw, Pitch, and Roll variables (in radians), as well as an end-effector position (in meters). Please assume that the tip of the sword is the end-effector. If an invalid position is generated, the program will throw the following warning: *Desired end effector position and orientation out of workspace of robot! No solutions found: Returning [0×1 sym].*

```
%% For Inverse Kinematics...
%Specify the end effector position as x,y,z coordinates.
alpha = 0; %yaw
beta = 0; %pitch
gamma = 0; %roll
end_effector_info = [0.5,0.5,0.5, gamma, beta, alpha];
```

For now, the Inverse Kinematics Package does not accommodate for Roll, Pitch, and Yaw, and hence, assumes that the robot is spherical manipulator when conducting solving. If you have a couple of hours to spare, you could go into Inverse_Kinematics.m file and modify line 34 to include roll, pitch, and yaw. Simple type "roll, pitch, yaw" into the VPASOLVE call before executing Main.

```
%If roll, pitch, and yar are desired, add them to the [xval,
yval, zval] array below.
theta_val = vpasolve([xval, yval, zval],[t1,t2,t3,t4,t5,t6],
[[-pi, pi];[-pi, pi];[-pi, pi];[-pi, pi];[-pi, pi];[-pi, pi]]);
```

Figure 7: An AAB Robot holding a Katana Vertically

Lastly, if you'd like access to the "Visualization" section of the project, you can uncomment the visualization call on line 68. You can also uncomment the the workspace display for the reachable/dexterous workspace on line 70. A fun feature of the visualization is that it is possible to display both at the same time (like done in Figure 6) by uncommenting both lines, then running Main. This is a great way to see visually where your end-effector is within the reachable workspace. These simulation environments will be called according to the thetas set under the "Forward kinematics" section of Main, so be sure to adjust them accordingly if you desire a different position!

```
visualization(fk_theta_val, sword_of_choice) %visualize the
XARM6 (with real dimensions from the URDF file)
%% For Displaying the XARM Workspace... [EXTRA CREDIT]
workspace(sword_of_choice) % Displays the reachable/dextrous
workspace of an XARM as a Voxel-Point Map... without sword in hand.
```

When executing Main, all sections (Forward Kinematics, Backward Kinematics, Visualization, and Workspace) will run at the same time. It takes about 30 seconds to process entirely, so please be patient. Please refer to Code Documentation if you have any questions about the individual functions contained

in Main.m, Inverse_kinematics.m, Forward_kinematics.m, Visualization.m, and Workspace.m.

7 Code Documentation

7.1 "forward_kinematics.m"

7.1.1 Function: forward_kinematics(theta_val, sword)

Docstring: Calculates the full transformation matrix of the robot from the start frame to the end frame.
Given:
theta_val: 1 x 6 double array of theta values, where each theta is respectively associated with each joint. Modified by the user in Main. All Thetas must be in radians.
sword: A double value, in METERS, associated with the length of the sword of the user's choosing. This value was choose in Main.
Returns:
end_effector_positions: (4 x 4) x 7 double array of Transformation Matrices from ORIGIN to joint "i" for i = 1,2,3,4,5,6,7, given the the joint angles as the input, and sword dimensions. the Robot End-Effector transformation matrix is the last matrix in this array.
Description: The algorithm starts with adding the offset values from the "DH parameters" function call onto the second and third theta values. The transformation matrices are then iteratively constructed with the transformation_matrix function and multiplied with each other. The positional column is then extracted and truncated to be used later. Returns the resulting i_T_7 matrices in the form of an array.

7.1.2 Function: transformation_matrix(alpha, a, theta, d)

Docstrong: Returns the *traditional* DH convention transformation matrix given alpha, a, d, and theta. Since the documentation has given us the parameters according to this convention, this matrix will be used instead of the matrix taught in class. (Please forgive us Prof. Peng).
Given:
alpha - "alpha" double in RADIANS (angle around x)
a - "a" double in METERS (length of linkage in x-axis)
theta - "theta" double in RADIANS (angle around z)
d - "d" double in METERS (length of link in z-axis)
Returns:
matrix - (4 x 4) array of representations
Description: Using the elements of the DH table, this function creates a transformation matrix that maps one frame to the next. Each cell of the matrix adheres to the DH convention, which is not part of the class but is instead used by the XARM 6 and the rest of its family. This convention incorporates a

combination of cosines and sines of both alpha and theta in the rotation matrix portion of the homogeneous transformation matrix, while the offset and link length populate the rightmost positional column.

7.1.3 Function: xarm_parameters(sword)

Docstring: Returns the UFACTORY X-Arm Parameters as 3 arrays, associated with official documentation.
Given:
sword: A double value, in METERS, associated with the length of the sword of the user's choosing. This value was choose in Main.
Returns:
alpha - "alpha" (1 x 7) double array in RADIANS (angle around x)
a - "a" (1 x 7) double array in METERS (length of linkage in x-axis)
theta - "theta" (1 x 7) double array in RADIANS (angle around z)
d - "d" (1 x 7) doubles in METERS (length of link in z-axis)
Description: This function returns UFACTORY X-Arm parameters as three arrays, taking a specific sword as input. The parameters are stored in arrays of length seven, with the variable representing theta excluded since it is a controlled input provided by the user in all functions that utilize this one. The sword argument is placed in the last position of the array representing the corresponding parameter, as the sword does not affect the rotation of the end effector, only its position. This results in the arrays for the other parameters, such as d and alpha, ending with a value of 0, to accommodate for the sword.

7.2 "inverse_kinematics.m"

7.2.1 Function: inverse_kinematics(end_effector_coord, sword)

Docstring: Returns the "theta_val", which is an array of "i" joint theta values for i = 1,2,3,4,5,6,7, given the end-effector position. Note hat theta_7 is ALWAYS zero, because the linkage between the sword and the previous joint is fixed.
Given:
end_effector_coord : A (1 x 6) array of doubles, following the convention [x,y,z,gamma,beta,alpha], where the first three input variables are in METERS, and the last three are in RADIANS. These values are set by the user in Main.
sword: A double value, in METERS, associated with the length of the sword of the user's choosing. This value was choose in Main.
Returns
theta_val: 1 x 6 double array of theta values, where each theta is respectively associated with each joint. All Thetas are returned in radians.
Description: Calculates the theta values needed to achieve a specific end effector orientation and position (returns an empty vector if the target is unreachable). It sets up equations by equating the symbolic versions of quantities (x, y, z, yaw, pitch, roll) to their desired values. Then, it uses the VPASOLVE function

to find the appropriate thetas that satisfy these equations within the range of -pi to pi, which represents the full range of each joint.

7.3 "visualization.m"

7.3.1 Function: visualization(theta_val, sword_length)

Docstring: Given an array of theta values in the form "theta_val", which is an array of "i" joint theta values for i = 1,2,3,4,5,6, displays the URDF of an XARM6 in the specified position. Functions with MATLAB version 2024a.
Given:
sword: A double value, in METERS, associated with the length of the sword of the user's choosing. This value was choose in Main.
theta_val: 1 x 6 double array of theta values, where each theta is respectively associated with each joint. Modified by the user in Main. All Thetas must be in radians.
Description: Displays the URDF of an XARM6 in the specified position given the theta values and the sword length. The IMPORTROBOT function in MATLAB allows you to load a URDF file, which contains all the data required to recreate the XARM 6 robot. The home configuration can then be adjusted using the specified theta values, altering the positions of each joint and plotting the robot's frames in space.

7.4 "workspace.m"

7.4.1 Function: workspace(sword_length)

Docstring: Displays the reachable/dextrous workspace as a Voxel-Map where the more blue the voxel-point is, the more solutions are associated, but the more red the voxel-point is, the less solutions are associated. Dark red indicates a single solution. Bright Blue indicates infinite solutions. Functions with MATLAB 2024a.
Given:
sword: A double value, in METERS, associated with the length of the sword of the user's choosing. This value was choose in Main.
Description: Displays the reachable/dexterous workspace as a Voxel-Map, with redder values indicating areas with fewer solutions and bluer values representing more dexterous regions. This is accomplished using the MATLAB 2024a built-in "manipulabilityIndex" function, which takes a robot defined with a URDF and the joint theta values. The results are then fed into the MATLAB 2024a "showWorkspaceAnalysis" function, which visualizes the workspace and robot using a voxel map.

8 References

[1] "Kinematic and Dynamic Parameters of UFACTORY xArm Series." UFACTORY Help Center, help.ufactory.cc/en/articles/4330809-kinematic-and-dynamic-parameters-of-ufactory-xarm-series. Accessed 19 Dec. 2024.

[2] https://pybullet.org/wordpress/

[3] https://www.mathworks.com/help/robotics/ref/importrobot.html

9 Appendix

9.0.1 Main.m

```
%% Main Executable Program for XARM Kinematics and Display

import forward_kinematics.*
import inverse_kinematics.*
import visualization.*
import workspace.*

%Note on things to improve:
% - Workspace & Visualization... incorporate the sword...?
% - inverse kinematics... clarify integration of end effector matrix
% - forward kinematics... origin to end? or link i-1 to link i?

%% For Forward Kinematics...
%Specify your "i" thetas, for each "i" joint (from Joints 1, 2... 5, 6).
fk_theta_val = [0.2, 0.8, 3.0, 0.0, 0.2, 0.0]; %forward kinematics theta values

%CHOOSE YOUR WEAPON!!!! >:D
%(Choose the sword that you'd like the XARM to wield in
honorable combat... sorted by length in METERS)
%(You can also choose "no sword," but where is the fun in that?)

no_sword = 0; %fist-fighting, of Paleolithic origin, I presume.

hudiedao = 0.30; %Qing Dynasty Era Chinese Origin

sica = 0.35; %Ancient Roman/Dacian Origin

machete = 0.45; %Colonial Central American Origin

katana = 0.6; %Feudal Japanese Origin

tachi = 0.75; %Koto period Japanese Origin

ZhanmaDao = 0.95; %Song Dynasty Era Chinese origin
```

```
rapier = 1.05; %Medieval Spanish Origin

longsword = 1.15; %Rennaisance Era Germanic Origin

claymore = 1.25; %Late Medieval Scottish Origin

sword_of_choice = no_sword; %%INSERT the NAME of the sword
of your choosing HERE! [i.e, machete]

disp("Executing Forward Kinematics, please stand by...")
trans_matrixies = forward_kinematics(fk_theta_val,
sword_of_choice); %1-7 joint positions and orientation!
end_effector_matrix = cell2mat(trans_matrixies(7)); %grabs last
trans. matrix for "tip of sword" position and orientation!
disp("Given the Thetas and the sword choice, the Final
Transformation Matrix is...")
disp(end_effector_matrix) %displays the End-Effector Posititon
and Orientation (origin to end-effector) Matrix.

%% For Inverse Kinematics...
%Specify the end effector position as x,y,z coordinates.
alpha = 0; %yaw
beta = 0; %pitch
gamma = 0; %roll
end_effector_info = [0.5,0.5,0.5, gamma, beta, alpha];

disp("Executing Inverse Kinematics, please stand by...")
%ik_theta_val = inverse_kinematics(end_effector_info,
sword_of_choice); %inverse kinematics theta values
disp("Given the End Effector Coordinates, and the given
Sword, the associated Thetas are...")
disp(ik_theta_val)

%% For Displaying Robot Positioning... without the sword.
%Specify your "i" thetas, for each "i" joint (from Joints 1, 2... 5, 6).
%Uncomment the visualization line if you'd like the visualization
%Uncomment the workspace display for the reachable/dexterous workspace
%Uncomment both if you'd like both at the same time! This is a great way to
%see visually where your end-effector is within the reachable workspace...

visualization(fk_theta_val, sword_of_choice) %visualize the
XARM6 (with real dimensions from the URDF file)
%% For Displaying the XARM Workspace... [EXTRA CREDIT]
workspace(sword_of_choice) % Displays the reachable/dextrous
workspace of an XARM as a Voxel-Point Map... without sword in hand.
```

9.0.2 Inverse_Kinematics.m

```
%% The Backwards Kinematics of a UFACTORY X-Arm (6 DOF)...
%Not accounting for end orientation... update needed!

function theta_val = inverse_kinematics(end_effector_coord, sword)
    import forward_kinematics.*
    %% Main: Calculate the inverse kinematics of UFACTORY X-Arm
    (6 DOF)
    %Inverse Kinematics: Returns the "theta_val", which is an
    array of "i" joint
    %theta values for i = {1,2,3,4,5,6,7}, given the end-
    effector position.
    %NOTE That theta_7 is ALWAYS zero!!

    syms t1 t2 t3 t4 t5 t6
    theta_val = [t1, t2, t3, t4, t5, t6];
    end_effector_positions = forward_kinematics(theta_val,
    sword); %passes symbolic values into FK...
    MegaMatrix = end_effector_positions{6}; %grabs symbolic end-
    effector transformation matrix!

    %this is where the x, y, and z values are set
    xval=MegaMatrix(1,4)==end_effector_coord(1);
    yval=MegaMatrix(2,4)==end_effector_coord(2);
    zval=MegaMatrix(3,4)==end_effector_coord(3);

    %this is where the roll, pitch, and yaw values are set
    pitch = atan2(-1*MegaMatrix(3,1), sqrt(MegaMatrix(1,1)^2 +
    MegaMatrix(2,1)^2)) == end_effector_coord(5);
    %check if cos(pitch) = zero, if so, roll and yaw are arbitrary:
    cospitch = cos(atan2(-1*MegaMatrix(3,1),
    sqrt(MegaMatrix(1,1)^2 + MegaMatrix(2,1)^2)));
    if cospitch == 0
         yaw = end_effector_coord(4);
         roll = end_effector_coord(6);
     else
     yaw = atan2(MegaMatrix(2,1)/cospitch,
     MegaMatrix(1,1)/cospitch) == end_effector_coord(4);
     roll = atan2(MegaMatrix(3,2)/cospitch,
     MegaMatrix(3,3)/cospitch) == end_effector_coord(6);
     end

    %If roll, pitch, and yar are desired, add them to the [xval,
    yval, zval] array below.
    theta_val = vpasolve([xval, yval, zval],[t1,t2,t3,t4,t5,t6]
```

```
    ,[[-pi, pi];[-pi, pi];[-pi, pi];[-pi, pi];[-pi, pi];[-pi, pi]]);

    for i = 1:length(theta_val)
        if ~isa(theta_val(i), 'double')
            warning('Desired end effector position and
            orientation out of workspace of robot! No solutions
            found: Returning [0×1 sym].');
        end
    end
end
```

9.0.3 Forward_Kinematics.m

```
%% The Forward Kinematics of a UFACTORY X-Arm (6 DOF)...

function end_effector_positions = forward_kinematics(theta_val, sword)
    %% Main: Calculate the forward kinematics of UFACTORY X-Arm (6 DOF)
    % Forward Kinematics: Returns the "end_effector_positions",
    % which is an array of transformation matrixies from ORIGIN to joint
    % "i" for i = {1,2,3,4,5,6,7}, given the the joint angles as the input,
    % and sword dimensions.
    % Note: the Robot End-Effector transformation matrix is the last matrix
    %in the "end_effector_positions" array.

    %According to the XARM Documentation, there are offsets for
    Theta 2 and Theta 3 that
    %We should be aware of... (in radians)

    theta_val(2) = theta_val(2) - 1.3849179;
    theta_val(3) = theta_val(3) + 1.3849179;

    theta_val(7) = 0; %The last theta is ALWAYS zero!

    [a_val, d_val, alpha_val] = xarm_parameters(sword);
    transform_array = cell(1, length(theta_val));

    %Calculates the transformation matrices for each joint
    for i = 1:length(theta_val)
        transform_array{i} = transformation_matrix(alpha_val(i),
        a_val(i), theta_val(i), d_val(i));
    end

    %Calculates each joint's "end effector position" by chaining
    %transformations together....
    end_effector_positions = cell(1, length(theta_val));
    end_effector_positions{1} = transform_array{1};
```

```
    for i = 2:length(transform_array)
        end_effector_positions{i} = end_effector_positions{i-1}
        * transform_array{i};
    end

end

function matrix = transformation_matrix(alpha, a, theta, d)
    %% This is the traditional DH convention transformation
    matrix...
    %Since the documentation has given us the parameters
    according to this convention,
    %This matrix will be used instead of the matrix taught in
    class. (Please forgive us Prof. Peng)
    matrix = [
        cos(theta), -sin(theta), 0, a;
        sin(theta)*cos(alpha), cos(theta)*cos(alpha), -sin(alpha), -d*sin(alpha);
        sin(theta)*sin(alpha), cos(theta)*sin(alpha), cos(alpha), d*cos(alpha);
        0, 0, 0, 1];
end

function [a_val, d_val, alpha_val] = xarm_parameters(sword)
    %% This function returns the UFACTORY X-Arm Parameters as 3 arrays

    %a_val : array of "a" parameters in METERS (length of linkage in x-axis)
    a_val = [0.0, 0.0, 0.28948866, 0.0775, 0.0, 0.076, sword];

    %d_val : array of "d" parameters in METERS (length of link in z-axis)
    d_val = [0.267, 0.0, 0.0, 0.3425, 0.0, 0.097, 0];

    %alpha_val : array of "alpha parameters in RADIANS (angle around x)
    alpha_val = [0, -pi/2, 0.0, -pi/2, pi/2, -pi/2, 0];
end
```

9.0.4 Visualization.m

```
%% Visualizing the Kinematics of a UFACTORY X-Arm (6 DOF)...

function visualization(theta_val, sword_length)
    %% Given an array of theta values in the form "theta_val",
    % which is an array of "i" joint theta values for i = {1,2,3,4,5,6},
    %displays the URDF of an XARM6 in the specified position.

    %imports the URDF of a XARM6
    robot = importrobot('xarm6_robot.urdf');
```

```
    sword = rigidBody('sword');
    hilt = rigidBodyJoint('hilt','revolute');
    sword.Joint = hilt;

    matrix = [
        1, 0, 0, sword_length;
        0, 1, 0, 0;
        0, 0, 1, 0;
        0, 0, 0, 1];

    setFixedTransform(hilt,matrix);
    addBody(robot,sword,"link6")

    config = homeConfiguration(robot); %Configures each joint...
    config(1).JointPosition = theta_val(1);
    config(2).JointPosition = theta_val(2);
    config(3).JointPosition = theta_val(3);
    config(4).JointPosition = theta_val(4);
    config(5).JointPosition = theta_val(5);
    config(6).JointPosition = theta_val(6);

    show(robot,config); %Displays a 3D visualization!
end
```

9.0.5 Workspace.m

```
%% Visualizing the Workspace of a UFACTORY X-Arm (6 DOF)...
function workspace(sword_length)
    %% Displays the reachable/dextrous workspace as a Voxel-Map
    %where the more blue the voxel-point is, the more solutions are associated,
    %but the more red the voxel-point is, the less solutions are associated.
    %Dark red indicates a single solution.
    %Bright Blue indicates infinite solutions.

    robot = importrobot('xarm6_robot.urdf'); %rigid body tree...

    sword = rigidBody('sword');
    hilt = rigidBodyJoint('hilt','fixed');
    sword.Joint = hilt;

    matrix = [
        1, 0, 0, sword_length;
        0, 1, 0, 0;
        0, 0, 1, 0;
        0, 0, 0, 1];
```

```
    setFixedTransform(hilt,matrix);
    addBody(robot,sword,"link6")

    ee = "sword";
    rng default
    [workspace,configs] = generateRobotWorkspace(robot,{},ee,IgnoreSelfCollision="on");
    mIdx = manipulabilityIndex(robot,configs,ee);
    hold on
    showWorkspaceAnalysis(workspace,mIdx,Voxelize=true)
    axis auto
    title("Voxelized Reachable/Dexterous Workspace")
    hold off
end
```