2021

Submitted on: Dec 19

Submitted to: Sir Khuwaja Fahad Iqbal

Robotics Project

Forward Kinematics of ROT3U

Rohail Ahmad Malik- 270952

Sameer Bin Khalid - 257215

Usman Shahid - 255526

Aqib Habib - 266993

Contents

Table of Figures

Forward Kinematics of ROT3U

Introduction:

ROT3U is a 6DOF Aluminum Robot Arm Kit with a Clamp Claw generally used for DIY projects or learning purpose. MG996R servos are used in 6 rotational joints so that joints can bear larger force. Forward kinematics is used to compute the position of the end-effector from specified values for the joint parameters.

We have used the Denavit–Hartenberg parameters (DH parameters) method to compute forward Kinematics of ROT3U robot.

DH parameters and Axes:

Axes Assignment:

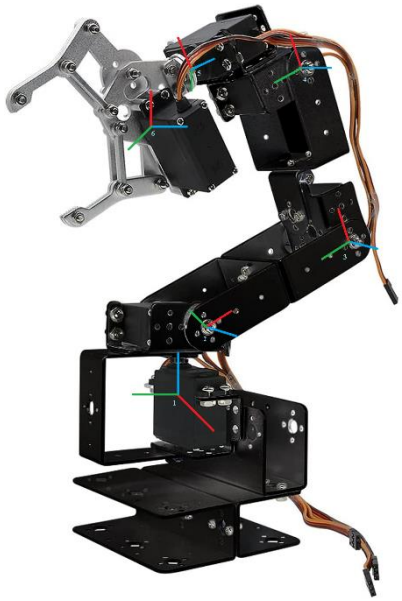

Figure 1 Axes of Rot3u

DH Table:

Table 1 DH Table of ROT3U

S#	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	90	0	0	θ_2
3	0	L_1	0	θ_3
4	0	L_2	0	θ_4
5	90	L_3	d_1	θ_5
6	-90	L_4	0	0

The distance a_1 has been taken as zero because there is no offset between the 2nd and third frame as seen in **Figure 7**.

Link Lengths and 3-D Model:

The lengths of DH table has been calculated through the 3-D model of the robot, the complete has been shown in **Figure 12** and the corresponding link lengths has been shown in **Figure 2, Figure 3, Figure 4, Figure 5, Figure 6,** and **Figure 7.**

2-D drawings of the brackets used in Rot3u robot can be seen in **APPENDIX I:**

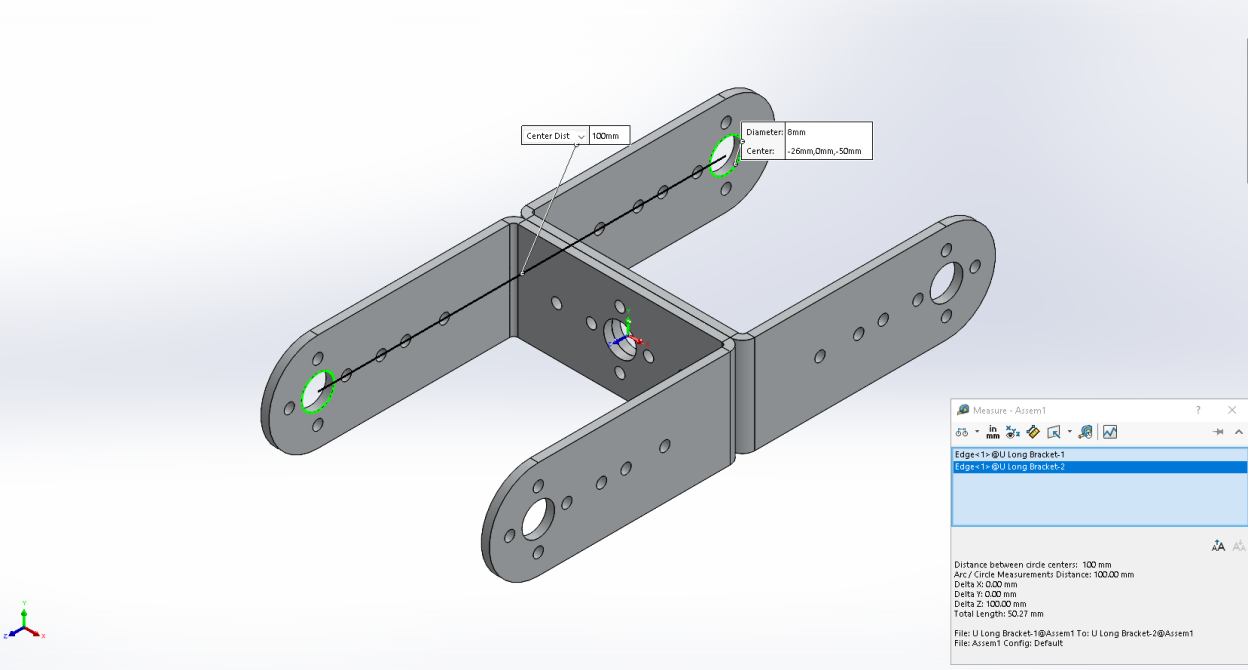

Figure 2 Link Length L_1

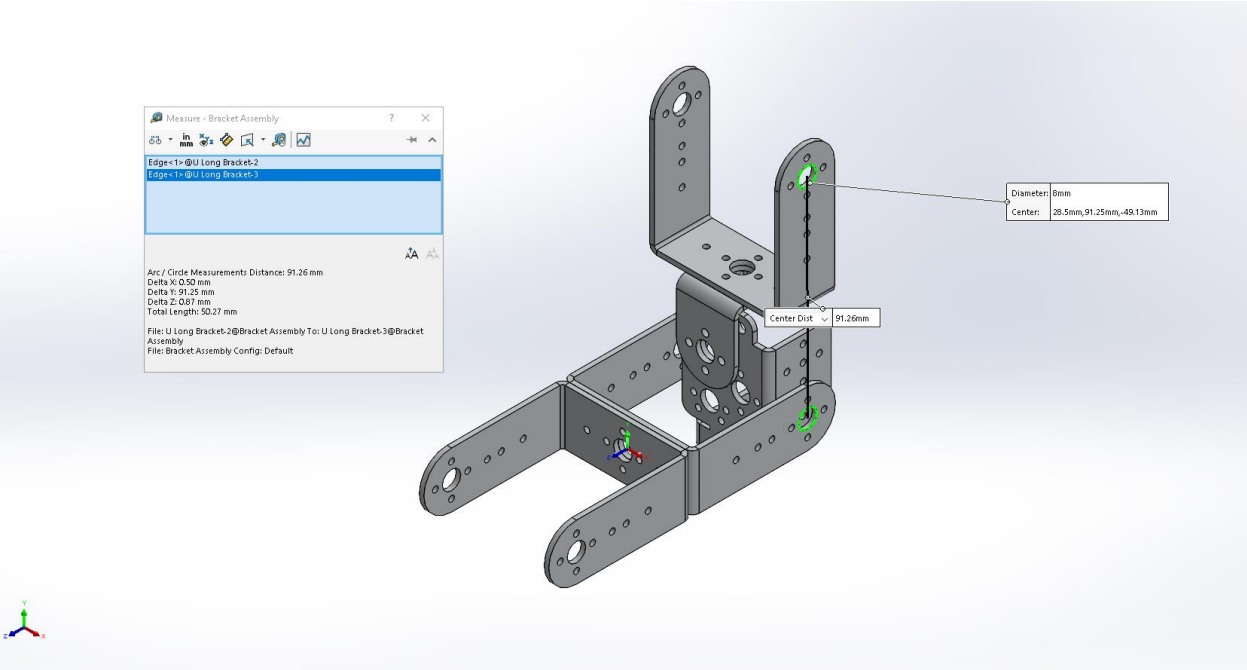

Figure 3 Link Length L_2

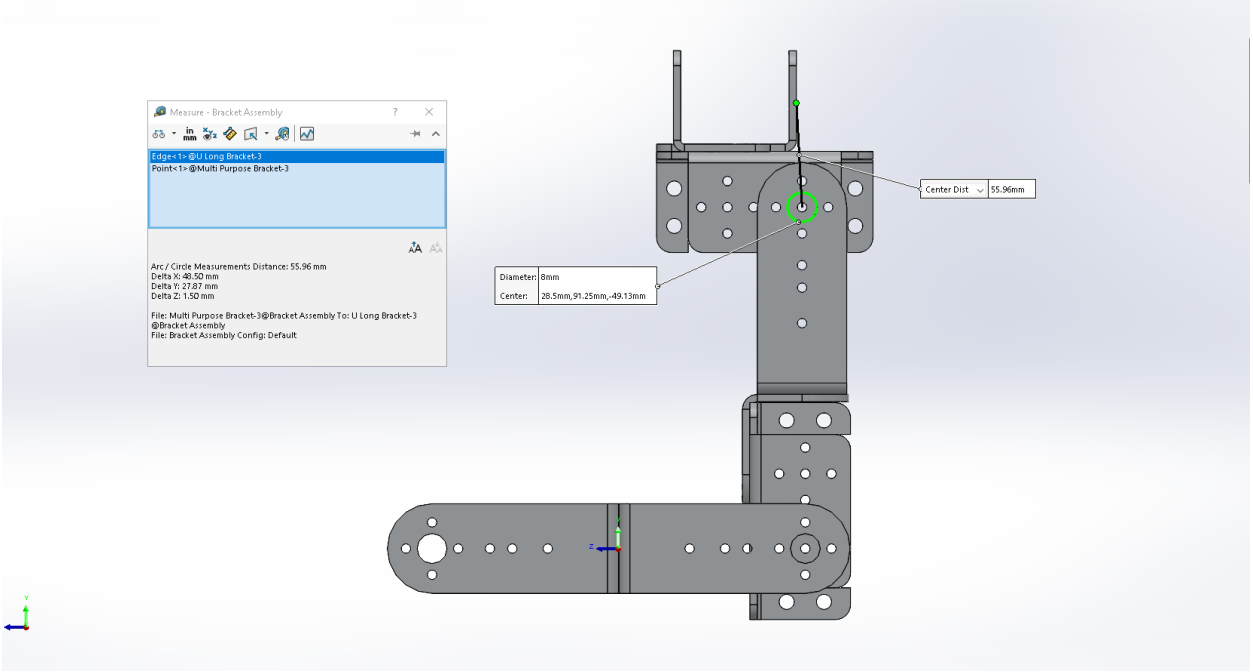

Figure 4 Link Length L_3

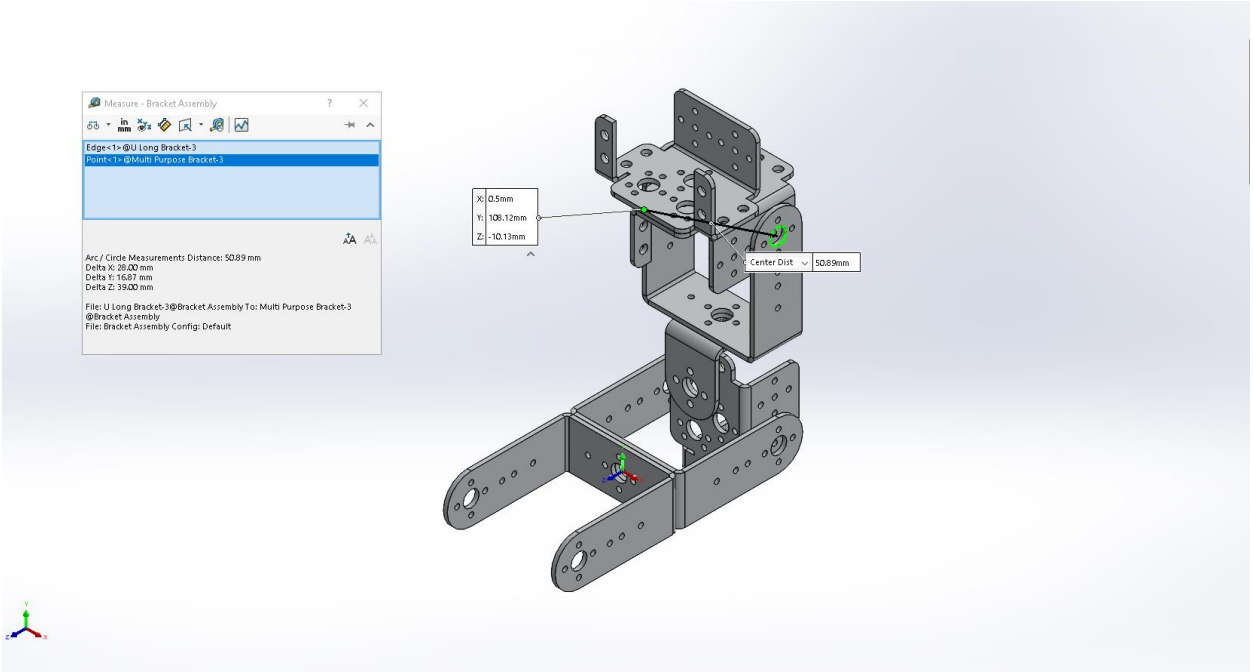

Figure 5 Offset Distance D_1

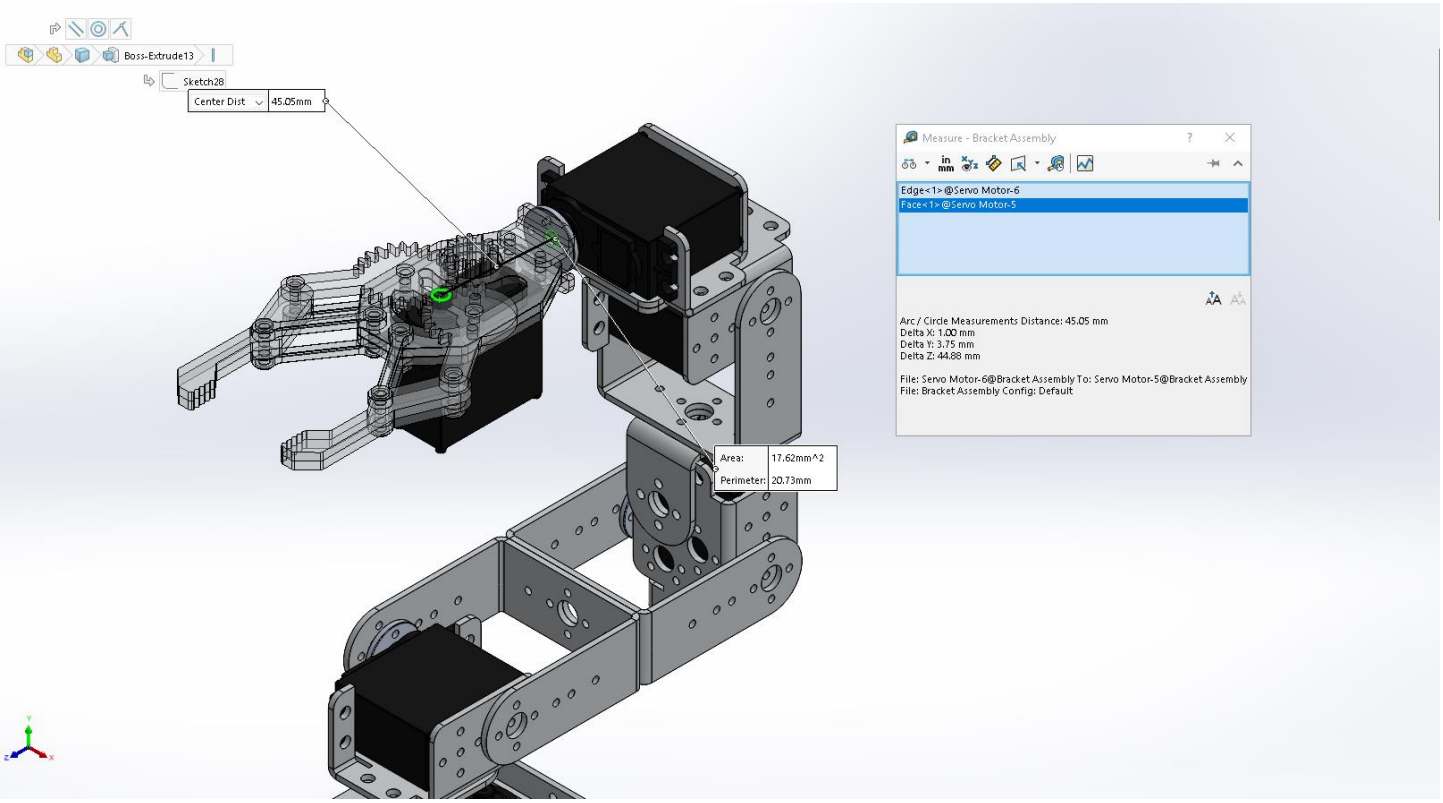

Figure 6 Link Length L_4

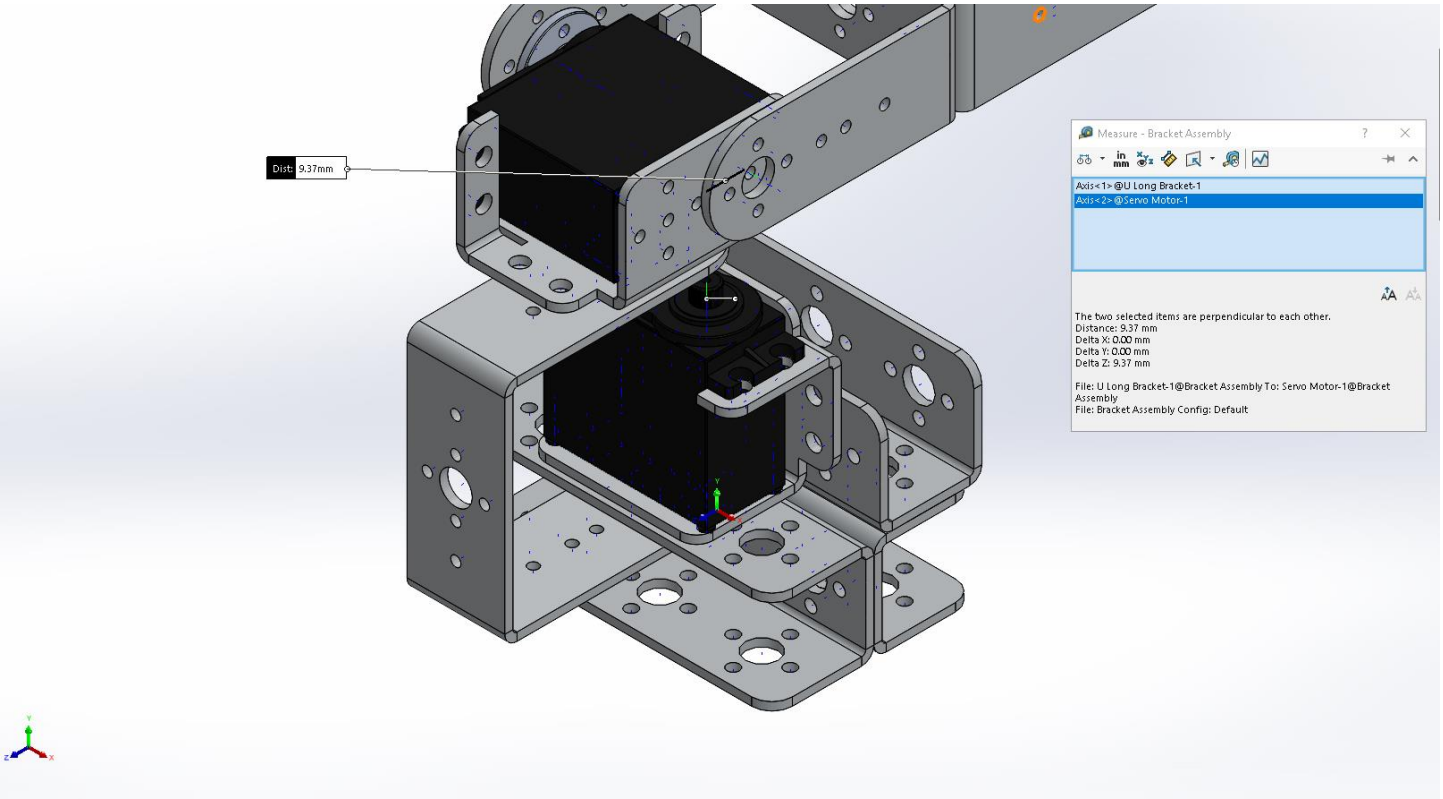

Figure 7 Link Length a_1

From **Figure 2** the length L_1 is 100 mm.

From **Figure 3** the length L_2 is 91.25mm.

From **Figure 4** length L_3 is 27.87 mm

From **Figure 6** the length L_4 = 3.75 mm

From **Figure 5** the length d_1 is 39 mm

Link Length	Length (mm)
L_1	100
L_2	91.25
L_3	27.87
L_4	3.75
D_1	39

Forward Kinematics:

To calculate the forward kinematics of the robot, a MATLAB code was constructed which can be seen in **APPENDIX II:**. Firstly, the symbolic representation of the forward kinematics is show in this section and outputs of MATLAB has been pasted here.

Finally, by inputting the links lengths obtained from the 3-D model, we derive the final pose of end effector for any joint angles.

General Transformation Matrices:

$$\text{dof} = 6$$

$$\text{a} = \begin{pmatrix} 0 & 0 & L_1 & L_2 & L_3 & L_4 \end{pmatrix}$$

$$\text{d} = \begin{pmatrix} 0 & 0 & 0 & 0 & D_1 & 0 \end{pmatrix}$$

$$\text{th} = \begin{pmatrix} \theta_1 & \theta_2 & \theta_3 & \theta_4 & \theta_5 & \theta_6 \end{pmatrix}$$

$${}^{0}_{1}T = \begin{pmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$${}^{1}_{2}T = \begin{pmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin(\theta_2) & \cos(\theta_2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$${}^{2}_{3}T = \begin{pmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & L_1 \\ \sin(\theta_3) & \cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$ {}^{3}_{4}T = \begin{pmatrix} \cos(\theta_4) & -\sin(\theta_4) & 0 & L_2 \\ \sin(\theta_4) & \cos(\theta_4) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

$$ {}^{4}_{5}T = \begin{pmatrix} \cos(\theta_5) & -\sin(\theta_5) & 0 & L_3 \\ 0 & 0 & -1 & -D_1 \\ \sin(\theta_5) & \cos(\theta_5) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

$$ {}^{5}_{6}T = \begin{pmatrix} \cos(\theta_6) & -\sin(\theta_6) & 0 & L_4 \\ 0 & 0 & 1 & 0 \\ -\sin(\theta_6) & -\cos(\theta_6) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

$$ {}^{0}_{6}T = {}^{0}_{1}T \times {}^{1}_{2}T \times {}^{2}_{3}T \times {}^{3}_{4}T \times {}^{4}_{5}T \times {}^{5}_{6}T $$

$$ {}^{\mathbf{0}}_{\mathbf{6}}\boldsymbol{T} = $$

$$ \begin{pmatrix} \cos(\theta_6)\,\sigma_3 - \sigma_1\cos(\theta_1)\sin(\theta_6) & -\sin(\theta_6)\,\sigma_3 - \sigma_1\cos(\theta_1)\cos(\theta_6) & \cos(\theta_5)\sin(\theta_1) - \sigma_5\cos(\theta_1)\sin(\theta_5) & L_4\sigma_3 + L_2\cos(\theta_2+\theta_3)\cos(\theta_1) + L_1\cos(\theta_1)\cos(\theta_2) + L_3\sigma_5\cos(\theta_1) + D_1\sigma_1\cos(\theta_1) \\ -\cos(\theta_6)\,\sigma_4 - \sigma_1\sin(\theta_1)\sin(\theta_6) & \sin(\theta_6)\,\sigma_4 - \sigma_1\cos(\theta_6)\sin(\theta_1) & -\cos(\theta_1)\cos(\theta_5) - \sigma_5\sin(\theta_1)\sin(\theta_5) & L_2\cos(\theta_2+\theta_3)\sin(\theta_1) + L_1\cos(\theta_2)\sin(\theta_1) - L_4\cos(\theta_1)\sin(\theta_5) + L_3\sigma_5\sin(\theta_1) + D_1\sigma_1\sin(\theta_1) + L_4\sigma_5\cos(\theta_5)\sin(\theta_1) \\ \sigma_5\sin(\theta_6) + \sigma_1\cos(\theta_5)\cos(\theta_6) & \sigma_5\cos(\theta_6) - \sigma_1\cos(\theta_5)\sin(\theta_6) & -\sigma_1\sin(\theta_5) & L_2\sin(\theta_2+\theta_3) + L_1\sin(\theta_2) + D_1\,(\sin(\theta_2+\theta_3)\sin(\theta_4) - \cos(\theta_2+\theta_3)\cos(\theta_4)) + L_3\sigma_2 + L_4\cos(\theta_5)\,\sigma_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

where

$$ \sigma_1 = \sin(\theta_2+\theta_3+\theta_4) $$

$$ \sigma_2 = \cos(\theta_2+\theta_3)\sin(\theta_4) + \sin(\theta_2+\theta_3)\cos(\theta_4) $$

$$ \sigma_3 = \sin(\theta_1)\sin(\theta_5) + \sigma_5\cos(\theta_1)\cos(\theta_5) $$

$$ \sigma_4 = \cos(\theta_1)\sin(\theta_5) - \sigma_5\cos(\theta_5)\sin(\theta_1) $$

$$ \sigma_5 = \cos(\theta_2+\theta_3+\theta_4) $$

$\theta_6 = 0$

Using Link Lengths:

dof = 6

a = $(0 \quad 0 \quad 100.0 \quad 91.25 \quad 27.87 \quad 3.75)$

d = $(0 \quad 0 \quad 0 \quad 0 \quad 39 \quad 0)$

th = $(\theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4 \quad \theta_5 \quad \theta_6)$

$$ {}^{0}_{1}T = \begin{pmatrix} \cos(\theta_1) & -1.0\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 1.0 \end{pmatrix} $$

$$ {}^{1}_{2}T = \begin{pmatrix} \cos(\theta_2) & -1.0\sin(\theta_2) & 0 & 0 \\ 0 & 0 & -1.0 & 0 \\ \sin(\theta_2) & \cos(\theta_2) & 0 & 0 \\ 0 & 0 & 0 & 1.0 \end{pmatrix} $$

$$ {}^{2}_{3}T = \begin{pmatrix} \cos(\theta_3) & -1.0\sin(\theta_3) & 0 & 100.0 \\ \sin(\theta_3) & \cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 1.0 \end{pmatrix} $$

$$ {}^{3}_{4}T = \begin{pmatrix} \cos(\theta_4) & -1.0\sin(\theta_4) & 0 & 91.25 \\ \sin(\theta_4) & \cos(\theta_4) & 0 & 0 \\ 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 1.0 \end{pmatrix} $$

$$ {}^{4}_{5}T = \begin{pmatrix} \cos(\theta_5) & -1.0\sin(\theta_5) & 0 & 27.87 \\ 0 & 0 & -1.0 & -39.0 \\ \sin(\theta_5) & \cos(\theta_5) & 0 & 0 \\ 0 & 0 & 0 & 1.0 \end{pmatrix} $$

$$ {}^{5}_{6}T = \begin{pmatrix} \cos(\theta_6) & -1.0\sin(\theta_6) & 0 & 3.75 \\ 0 & 0 & 1.0 & 0 \\ -1.0\sin(\theta_6) & -1.0\cos(\theta_6) & 0 & 0 \\ 0 & 0 & 0 & 1.0 \end{pmatrix} $$

$$ {}^{0}_{6}T = {}^{0}_{1}T \times {}^{1}_{2}T \times {}^{2}_{3}T \times {}^{3}_{4}T \times {}^{4}_{5}T \times {}^{5}_{6}T $$

$ {}^{0}_{6}T = $

$$ \begin{pmatrix} \cos(\theta_6)\,o_3 - 1.0\,o_1\cos(\theta_5)\sin(\theta_6) & -1.0\sin(\theta_6)\,o_3 - 1.0\,o_1\cos(\theta_5)\cos(\theta_6) & \cos(\theta_5)\sin(\theta_1) - 1.0\,o_4\cos(\theta_1)\sin(\theta_5) & 100.0\cos(\theta_1)\cos(\theta_5) + 3.75\sin(\theta_1)\sin(\theta_5) - 91.25\cos(\theta_1)\sin(\theta_5)\sin(\theta_5) + 3.75\,o_4\cos(\theta_1)\cos(\theta_5) + 27.87\cos(\theta_5+\theta_5)\cos(\theta_1)\cos(\theta_4) + 39.0\cos(\theta_5+\theta_5)\cos(\theta_1)\sin(\theta_4) + 39.0\sin(\theta_5+\theta_5)\cos(\theta_1)\cos(\theta_4) - 27.87\sin(\theta_5+\theta_5)\cos(\theta_1)\sin(\theta_4) + 91.25\cos(\theta_1)\cos(\theta_5)\cos(\theta_5) \\ -1.0\cos(\theta_6)\,o_2 - 1.0\,o_3\sin(\theta_1)\sin(\theta_6) & \sin(\theta_6)\,o_2 - 1.0\,o_3\,o_1\cos(\theta_6)\sin(\theta_1) & -1.0\cos(\theta_1)\cos(\theta_5) - 1.0\,o_4\sin(\theta_1)\sin(\theta_5) & 100.0\cos(\theta_5)\sin(\theta_1) - 3.75\cos(\theta_1)\sin(\theta_5) - 91.25\sin(\theta_1)\sin(\theta_5)\sin(\theta_5) + 3.75\,o_4\cos(\theta_5)\sin(\theta_1) + 27.87\cos(\theta_5+\theta_5)\cos(\theta_4)\sin(\theta_1) + 39.0\cos(\theta_5+\theta_5)\sin(\theta_1)\sin(\theta_4) + 39.0\sin(\theta_5+\theta_5)\cos(\theta_4)\sin(\theta_1) - 27.87\sin(\theta_5+\theta_5)\sin(\theta_1)\sin(\theta_4) + 91.25\cos(\theta_5)\cos(\theta_5)\sin(\theta_1) \\ o_4\sin(\theta_6) + o_1\cos(\theta_5)\cos(\theta_6) & o_4\cos(\theta_6) - 1.0\,o_1\cos(\theta_5)\sin(\theta_6) & -1.0\,o_1\sin(\theta_5) & 1.875\sin(\theta_5+\theta_5+\theta_4-1.0\,\theta_5) - 47.93\cos(\theta_5+\theta_5+\theta_4+0.6205) + 1.875\sin(\theta_5+\theta_5+\theta_4+\theta_5) + 91.25\sin(\theta_5+\theta_5) + 100.0\sin(\theta_5) \\ 0 & 0 & 0 & 1.0 \end{pmatrix} $$

where

$o_1 = \sin(\theta_5+\theta_5+\theta_4)$

$o_2 = \cos(\theta_1)\sin(\theta_5) - 1.0\,o_4\cos(\theta_5)\sin(\theta_1)$

$o_3 = \sin(\theta_1)\sin(\theta_5) + o_4\cos(\theta_1)\cos(\theta_5)$

$o_4 = \cos(\theta_5+\theta_5+\theta_4)$

$\theta_6 = 0$

Conclusion:

The forward kinematics of Rot3u robot has been calculated for arbitrary angles of θ but with numeric link lengths. The values joint angles have some restrictions which will be investigated during the programming phase of the robot. In future reports, the 3-D model can also be used to calculate the required position and joint angles of the robot to achieved a certain task. The information in this report will be used later to program the robot to achieve a certain a task such as playing tic-tac-toe or object sorting.

APPENDIX I:

MATLAB Code of Forward Kinematics:

```
% RobotiQ
% Matlab Code for Forward Kinematics of ROT3U
syms L1 L2 L3 L4 D1
dof = 6
a = sym([0 0 L1 L2 L3 L4])
d = sym([0 0 0 0 D1 0])
al = sym([0 pi/2 0 0 pi/2 -pi/2]);
th = sym('theta',[1,dof])
T = sym(zeros(4,4,dof));
Tf = sym(eye(4));
for i = 1:dof
    T(:,:,i) = [
      cos(th(i))             -sin(th(i))            0              a(i)      ;
sin(th(i))*cos(al(i))   cos(th(i))*cos(al(i))  -sin(al(i))   -sin(al(i))*d(i);
sin(th(i))*sin(al(i))   cos(th(i))*sin(al(i))   cos(al(i))    cos(al(i))*d(i);
          0                      0                 0               1      ];
    Tf = Tf*T(:,:,i);
end
T
simplify(Tf)
```

APPENDIX II:

2-D Drawings of Links:

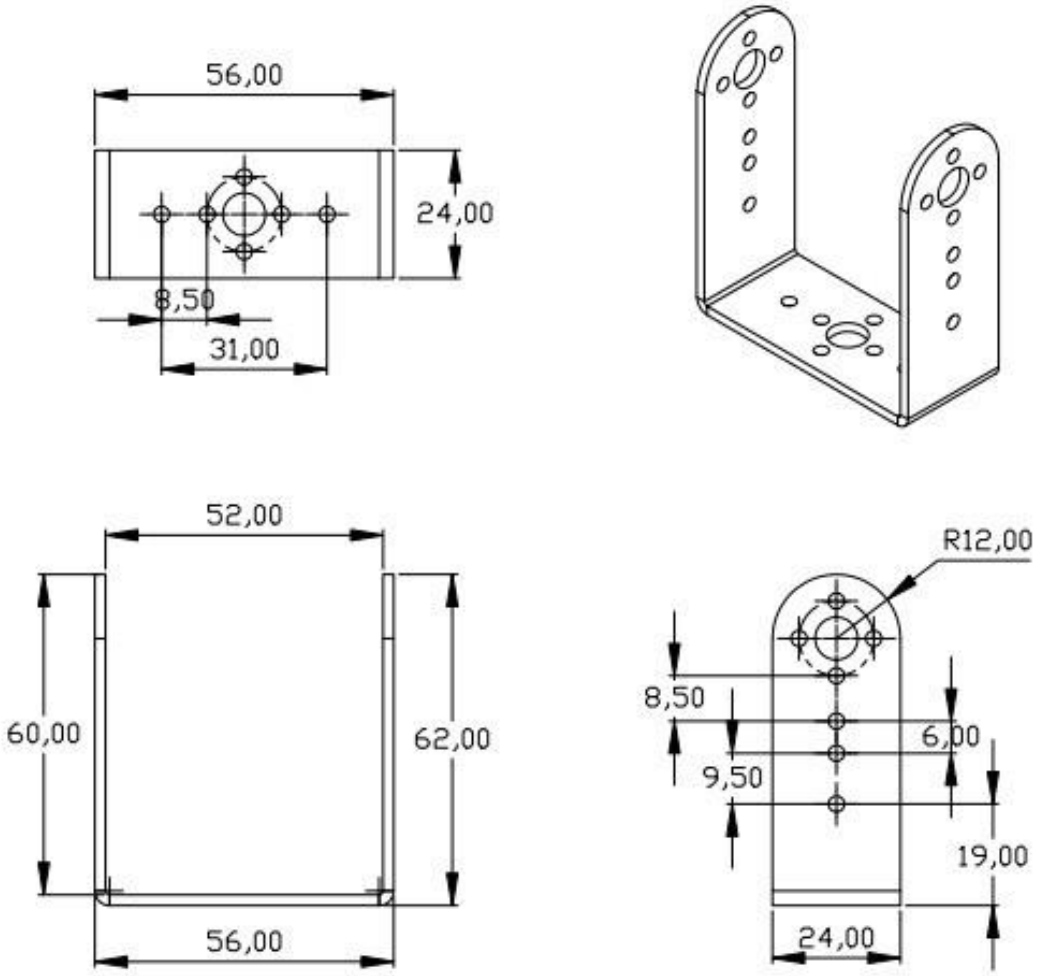

Figure 8 Drawing of Long-U Bracket

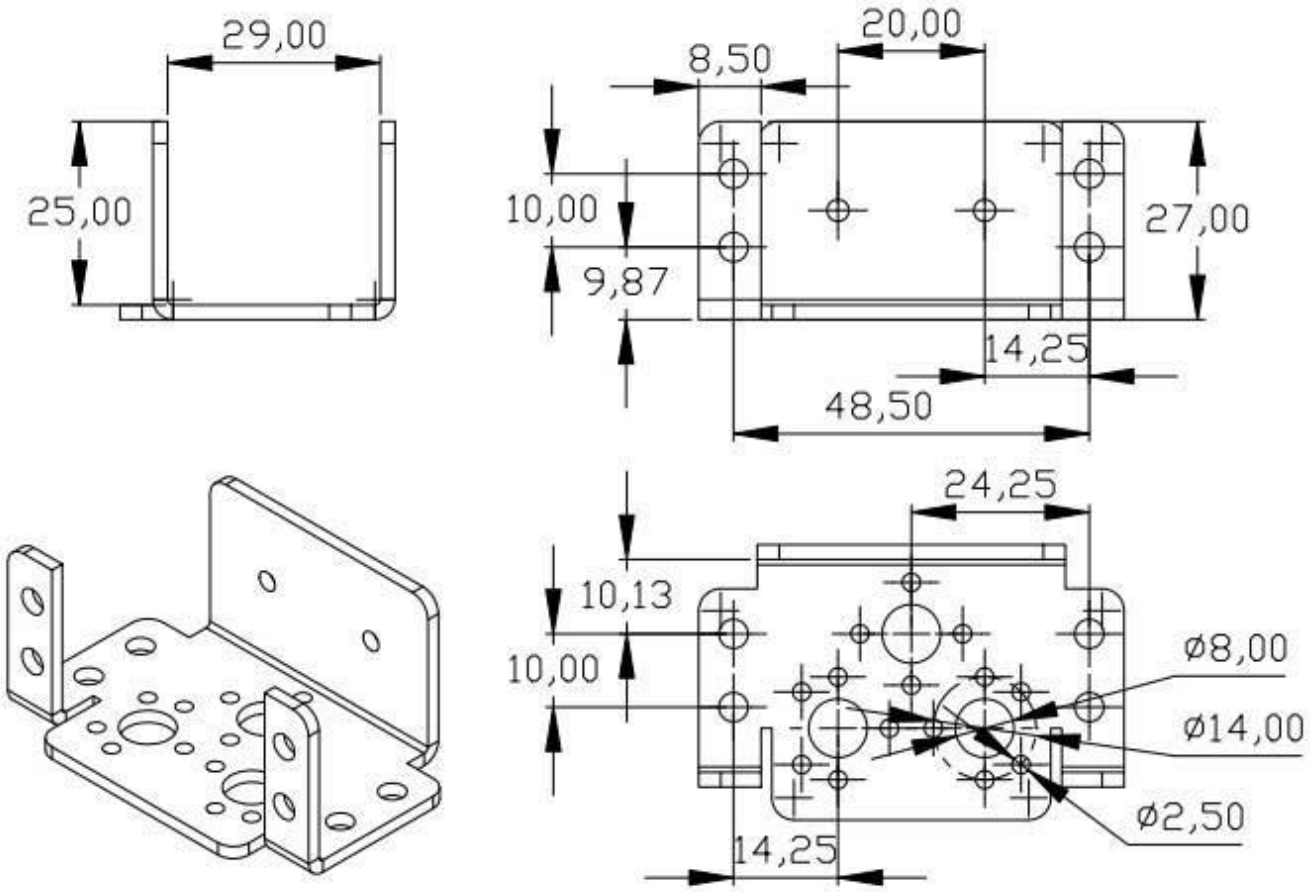

Figure 9 Drawing of Multiple Usage Bracket

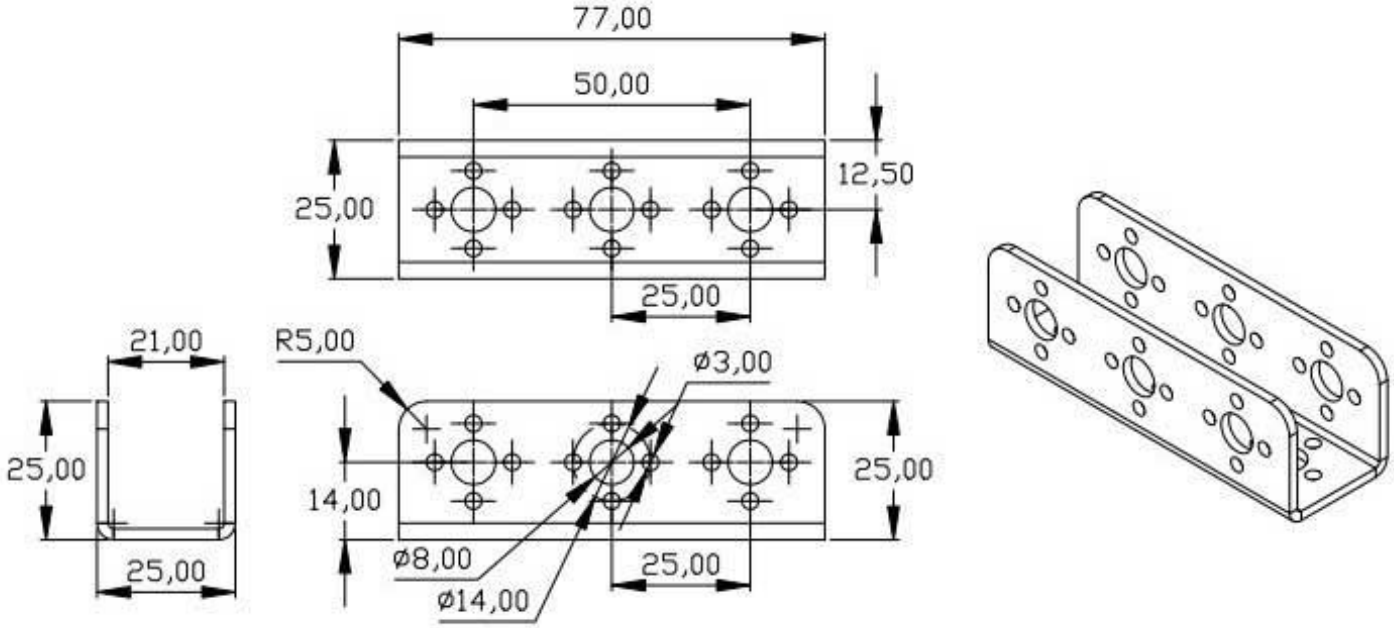

Figure 10 Drawing of Base Bracket

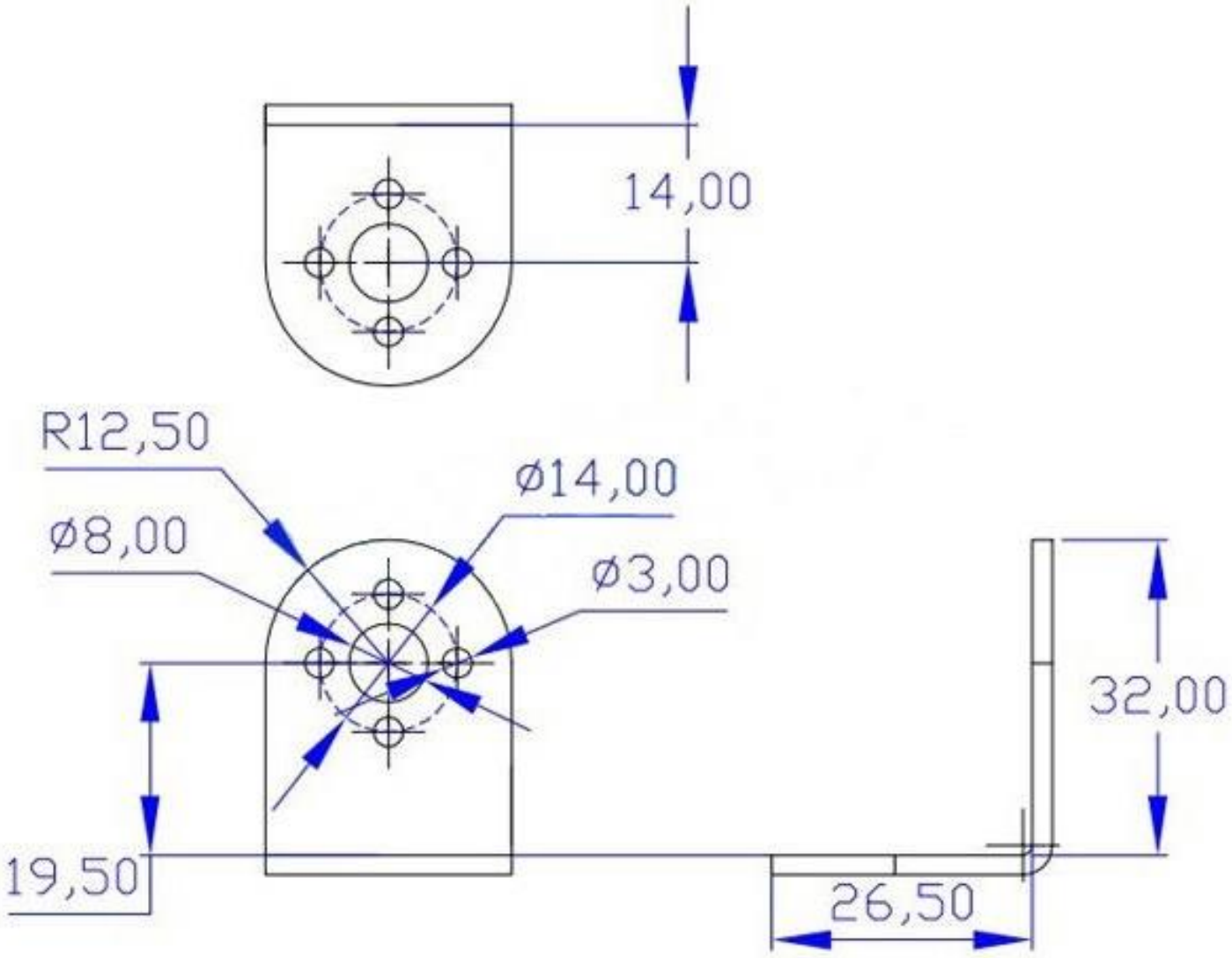

Figure 11 Drawing of Corner Bracket

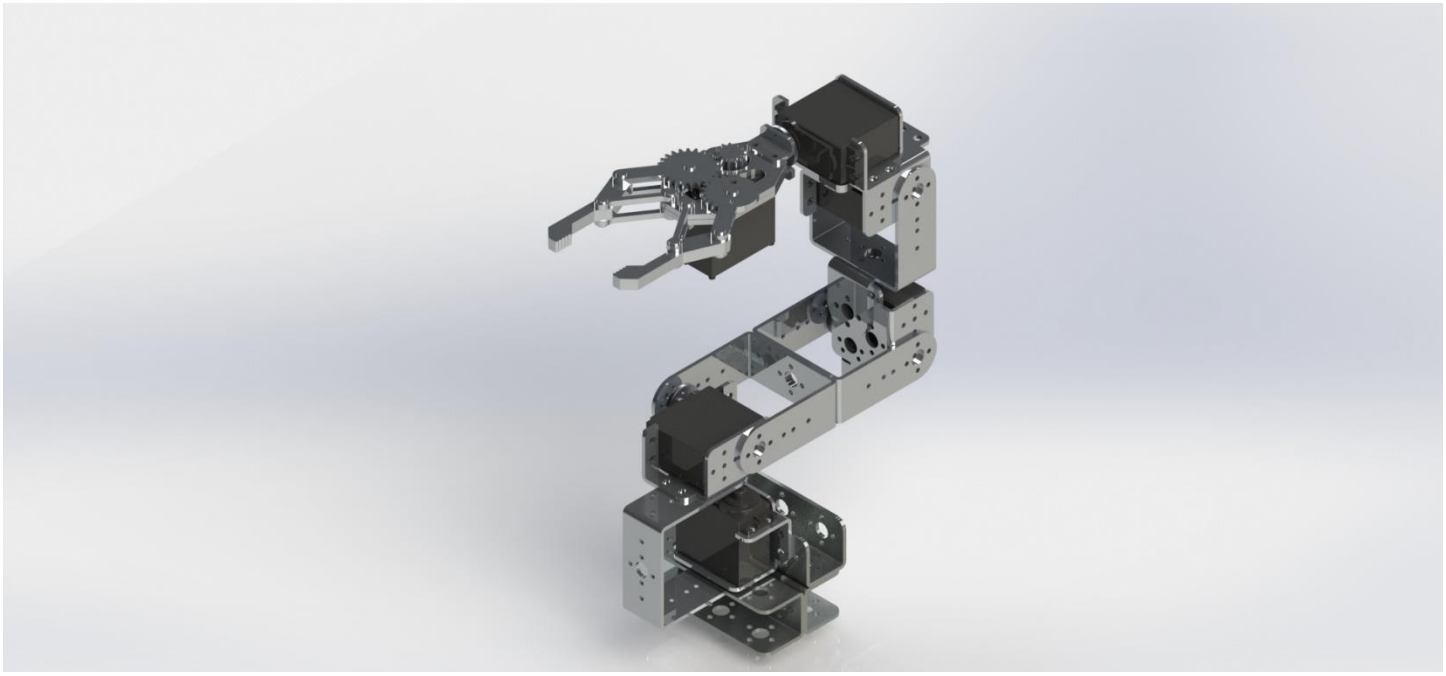

Figure 12 Complete Assembly of Robot

2021

Submitted on: Dec 28

Submitted to: Sir Khuwaja Fahad Iqbal

Robotics Project

Inverse Kinematics of ROT3U

Rohail Ahmad Malik- 270952

Sameer Bin Khalid - 257215

Usman Shahid - 255526

Aqib Habib - 266993

Contents

Table of Figures

Inverse Kinematics of ROT3U

In this report, the inverse kinematics of ROT3U has been calculated by using firstly a numerical method and secondly by geometric method. The improvements and progressions from previous report has also been mentioned.

Improvements from Submission 1:

1. We had taken link lengths and offsets of ROT3U from internet. A CAD assembly was created using standard brackets to calculate exact values.
2. However, in the pictures available on internet, we could not visualize the offset.
3. The actual robot available in lab had an offset on the joint. Therefore, corrected length of a1 = 31mm and in addition to this offset, other lengths have also been corrected.

DH Table:

Table 1 DH Table of ROT3U

S#	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	90	L_1	0	θ_2
3	0	L_2	0	θ_3
4	0	L_3	0	θ_4
5	90	L_4	d_1	θ_5
6	-90	L_5	0	0

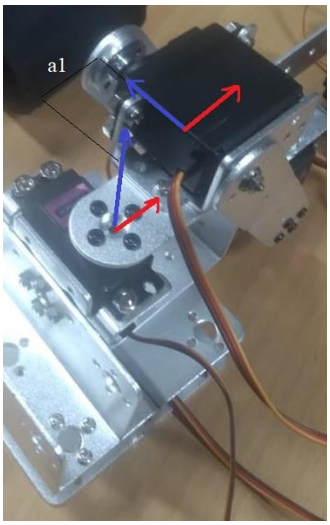

Figure 1 Link Offset (a_2)

Inverse Kinematics using Numerical Method:

A rigid multibody system consists of a set of rigid objects, called links, joined together by joints. To control the movement of a rigid multibody it is common to use inverse kinematics (IK). For IK, it is presumed that specified points, called "end effectors," on the links are assigned "target positions."

To solve the IK problem, we must find settings for the joint angles so that the resulting configuration of the multibody places each end effector at its target position. There are several methods for solving IK problems, coming originally from robotics applications. These include geometric method, cyclic coordinate descent methods, pseudoinverse methods, Jacobian transpose methods, the Levenberg-Marquardt damped least squares methods, quasi-Newton and conjugate gradient methods, and neural net and artificial intelligence methods.

We will use two methods to calculate Inverse Kinematics for the ROT3U robot.

1. Jacobian Pseudoinverse Method
2. Geometric Method

Jacobian Pseudoinverse Method:

Let the transformation matrix of the robot from origin to end-effector be defined as:

$$\begin{bmatrix} \alpha_x & \beta_x & \gamma_x & x \\ \alpha_y & \beta_y & \gamma_y & y \\ \alpha_z & \beta_z & \gamma_z & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Each element of the matrix is a function of joint angles, given by

$$q = [\theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4 \quad \theta_5]^T$$

Thus, r, which is an array consisting of all functions from transformation matrix, is a function of q which describes the pose of end effector.

$$r(q) = [\alpha_y \quad \alpha_y \quad \alpha_y \quad \beta_x \quad \beta_y \quad \beta_z \quad \gamma_x \quad \gamma_y \quad \gamma_z \quad x \quad y \quad z]^T$$

The Jacobian of $r(q)$ is thus defined as:

$$J(q) = \frac{dr}{dq} \Rightarrow dq = J(q)^{-1} dr$$

$$\Delta q \approx J(q)^{-1} \Delta r$$

If t is the value of $r(q)$ given for some unknown q, then the error between t and position at any q is given as

$$e(q) = t - r(q) = \Delta r$$

Thus, for an iterative process:

$$q_{k+1} = q_k + J^{-1}(q_x)[t - r(q_k)]$$

If J is not square and hence not invertible, pseudo inverse of J is used:

$${J_{mod}}^{-1}(q_x) = J(q_x)^T [J(q_x) \cdot J(q_x)^T]^{-1}$$

The results of this method and code is available in **APPENDIX I**

Inverse Kinematics using Geometric Method:

For the calculation of inverse kinematics of ROT3U, 3-D calculator (Geogebra) and solidworks is used. The complete solidworks assembly is available in **APPENDIX II:** and the following figures demonstrate the usage of these software.

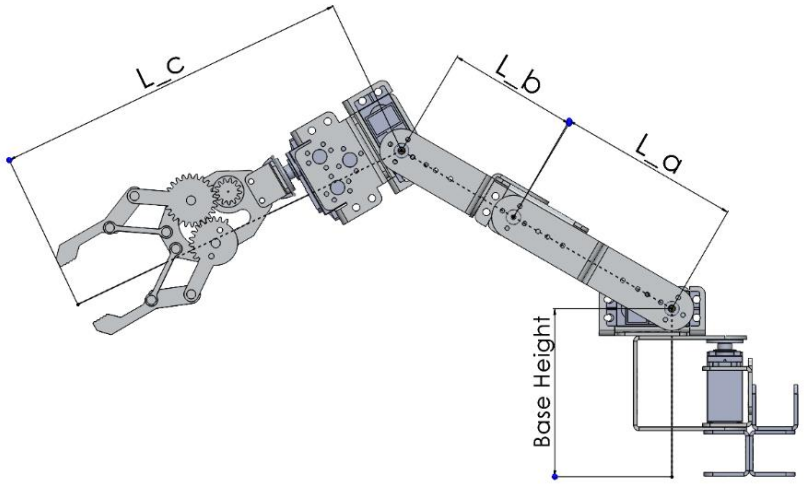

Figure 2: 3-D Model (SOLIDWORKS)

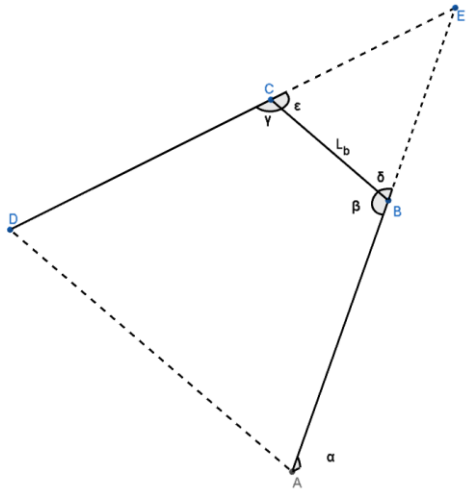

Figure 3: 2D Simplification (Geogebra)

The ROT3U robot can be simplified into 2-D geometry to calculate its inverse kinematics.

First, we will define some terminologies used in Error! Reference source not found.

Points:

- 'A' is the origin of our Cartesian system, so it has coordinates of (0,0,0)
- 'D' is the point that we have to reach i.e., the end effector pose.

AB, BC, CD (L_a , L_b , L_c) are the lengths of each arm segments.

The purpose is to find the angles, α, β, γ.

Calculation of Joint Angles:

The first step is to calculate the length “AD” and it can by achieved by using Pythagorean theorem.

$$AD = \sqrt{x^2 + y^2 + (z - Base\ Height)^2}$$

Where x, y, z is cartesian coordinates of our point D and the whole cartesian plane is at a offset of Base Height.

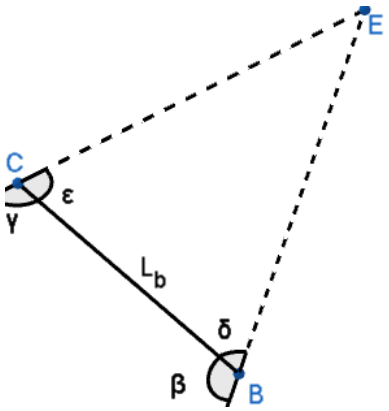

Figure 4 Triangle BEC

We make a small approximation that Length “BE” is equal to “CE” and hence $\delta = \epsilon\ and\ \gamma = \beta$

Then,

$$BEcos(\alpha) = \frac{BC}{2} \rightarrow BE = \frac{BC}{2\cos(\delta)}$$

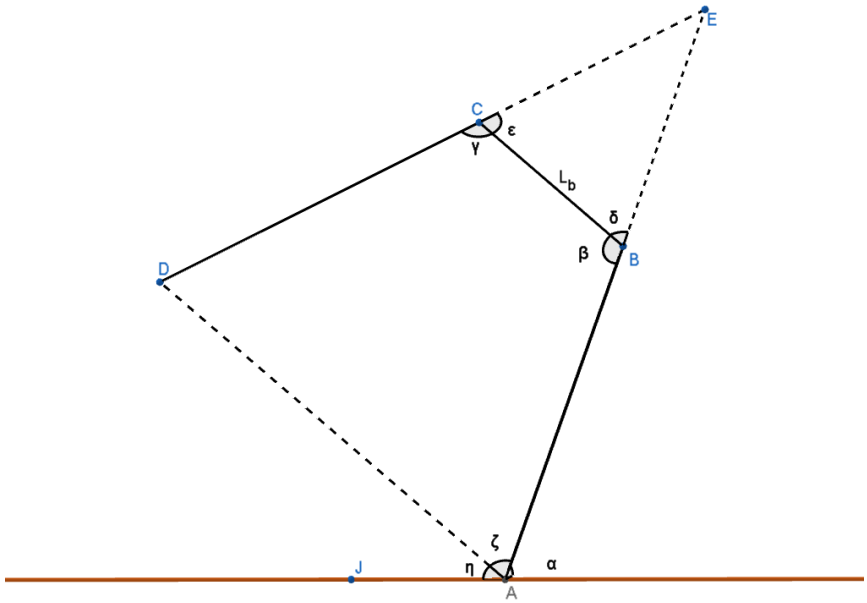

Figure 5 Triangle AED

We use law of cosines on triangle AED.

$$AD^2 = (AB + BE)^2 + (DC + CE)^2 - 2(AB + BE)(DC + CE)\cos(\phi)$$

And

$$BE = CE = \frac{BC}{2\cos(\delta)} \; and \; \phi = 180 - 2\delta$$

Substituting these variables, we get,

$$AD^2 = \left(AB + \frac{BC}{2\cos(\delta)}\right)^2 + \left(DC + \frac{BC}{2\cos(\delta)}\right)^2 - 2(AB + BE)(DC + CE)\cos(180 - 2\delta)$$

After simplifying above relation, we get,

$$4AB * CDcos(\delta)^2 + 2BC(AB + CD)\cos(\delta) + (AB^2 + BC^2 + CD^2 - AD^2 - 2AB * CD) = 0$$

Notice that the above equation is in quadratic form if we substitute $cos\delta = X$

Using quadratic formula

$$X = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\cos(\delta) = \frac{-BC * (AB + CD) \pm \sqrt{\left(BC(AB + CD)\right)^2 - 4AB * CD(AB^2 + BC^2 + CD^2 - AD^2 - 2AB * CD)}}{4AB * CD}$$

$$\delta = \cos^{-1}\left(\frac{-BC * (AB + CD) \pm \sqrt{\left(BC(AB + CD)\right)^2 - 4AB * CD(AB^2 + BC^2 + CD^2 - AD^2 - 2AB * CD)}}{4AB * CD}\right)$$

We again use law of cosines on triangle ABD and BCD

$$BD^2 = AB^2 + AD^2 - 2AB * ADcos(\tau)$$

$$BD^2 = BC^2 + CD^2 - 2BC * CDcos(\beta)$$

$$AB^2 + AD^2 - 2AB * ADcos(\tau) = BC^2 + CD^2 - 2BC * CDcos(\beta)$$

$$\tau = \cos^{-1}\left(\frac{BC^2 + CD^2 - 2BC * CDcos(\beta) - AD^2 - AB^2}{2AB * AD}\right)$$

$$\eta = \cos^{-1}\left(\frac{AK}{AD}\right)$$

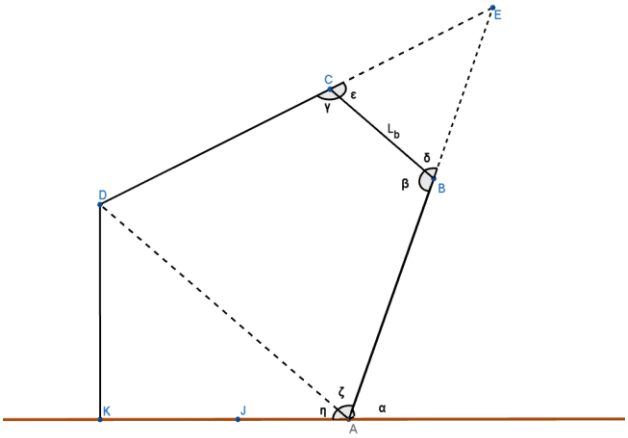

Figure 6 Calculation of Joint Angle η

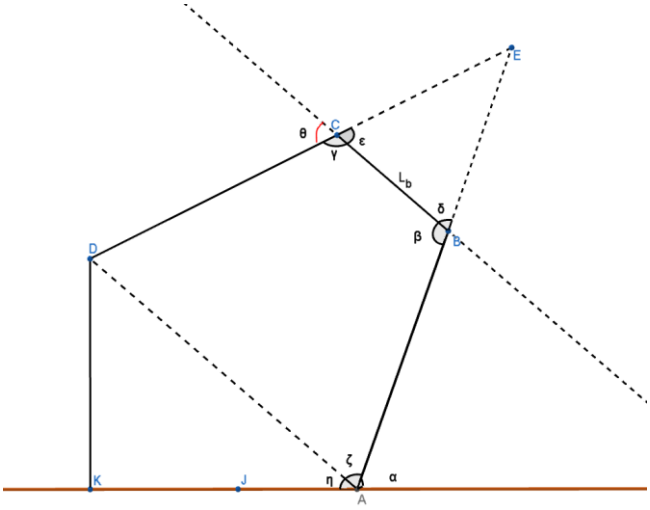

Figure 7 Joint Angle θ

Now the joint angles are:

Joint Angle 1:

$$\alpha = 180 - \tau - \eta \tag{1}$$

Joint Angle 2:

$$\delta = \cos^{-1}\left(\frac{-BC * (AB + CD) \pm \sqrt{\left(BC(AB + CD)\right)^2 - 4AB * CD(AB^2 + BC^2 + CD^2 - AD^2 - 2AB * CD)}}{4AB * CD}\right) \tag{2}$$

Joint Angle 3:

$$\theta = 180 - \beta \tag{3}$$

Conclusion:

The inverse kinematics of the robot is calculated from the known forward kinematics of previous report. Two methods of inverse kinematics have been used, the first is a complicated numerical solution and the second is the simplified geometric solution of planar joint angles. Using software like solidworks and GeoGebra, the process of solving for inverse kinematics was made easy. In the Arduino programming, two methods of controlling the robot will be used. The first will be controlling each servo using an IR remote and the second will be putting x , y , z coordinates of the end effector and joint angles will be optimized according to that.

APPENDIX I

MATLAB Code of Jacobian Pseudoinverse Method:

```
% Inverse Kinematics of a 6 DOF Robot
% The function for_kin.m is required

dof = 6 % Degrees of freedom
digits(3) % Accuracy of 'vpa'

% DH Paramters
a = sym([0 31 103 103 28 7]) % a
d = sym([0 0 0 0 64 0]) % d
al = sym([0 pi/2 0 0 pi/2 -pi/2]) % alpha
th = sym('theta',[1,dof]); % theta
th(1,6)=0

% Given theta for providing input to inverse kinematics
th_g = sym([pi/3 pi/8 pi/4 pi/6 pi/9 0])

% Robot's General Transformation Matrix & functions
Tm = for_kin(dof,a,al,d,th)
r = [Tm(1,1); Tm(2,1); Tm(3,1);
     Tm(1,2); Tm(2,2); Tm(3,2);
     Tm(1,3); Tm(2,3); Tm(3,3);
     Tm(1,4); Tm(2,4); Tm(3,4)];

% Robot's Given Transformation Matrix & functions
Tg = for_kin(dof,a,al,d,th_g)
t = [Tg(1,1); Tg(2,1); Tg(3,1);
     Tg(1,2); Tg(2,2); Tg(3,2);
     Tg(1,3); Tg(2,3); Tg(3,3);
     Tg(1,4); Tg(2,4); Tg(3,4)];

% Joint angle variables
q = sym('theta',[5,1]);
% Jacobian Matrix
J = simplify(vpa(jacobian(r,q)));
% Initial guess
qn = [pi/2; pi/2; pi/2; pi/2; pi/2];
% Desired accuracy
acc = 0.001;
```

```
% Iterations
for k = 1:2
    rn = simplify(vpa(subs(r, q, qn)));
    e = t - rn; % Error
    Jn = simplify(vpa(subs(J, q, qn)));
    qn_old = qn;
    qn = simplify(vpa(qn +  ((Jn'/(Jn*Jn'))*0.05*e)));
    if (norm(abs(qn-qn_old)) < acc)
        break
    end
end
qn
% Joint angle values
```

MATLAB Function for Forward Kinematics:

```
function TrM = for_kin(dof,a,al,d,th)
% This function performs the forward kinematics of
% a robot using its Denavit-Hartenberg parameters.
T = sym(zeros(4,4,dof));
Tf = sym(eye(4));

for i = 1:dof
    T(:,:,i) = [
      cos(th(i))             -sin(th(i))            0              a(i)      ;
sin(th(i))*cos(al(i))   cos(th(i))*cos(al(i))  -sin(al(i))   -sin(al(i))*d(i);
sin(th(i))*sin(al(i))   cos(th(i))*sin(al(i))   cos(al(i))    cos(al(i))*d(i);
          0                     0                 0                1      ];
    Tf = Tf*T(:,:,i);
end
TrM = simplify(vpa(Tf));
```

APPENDIX II:

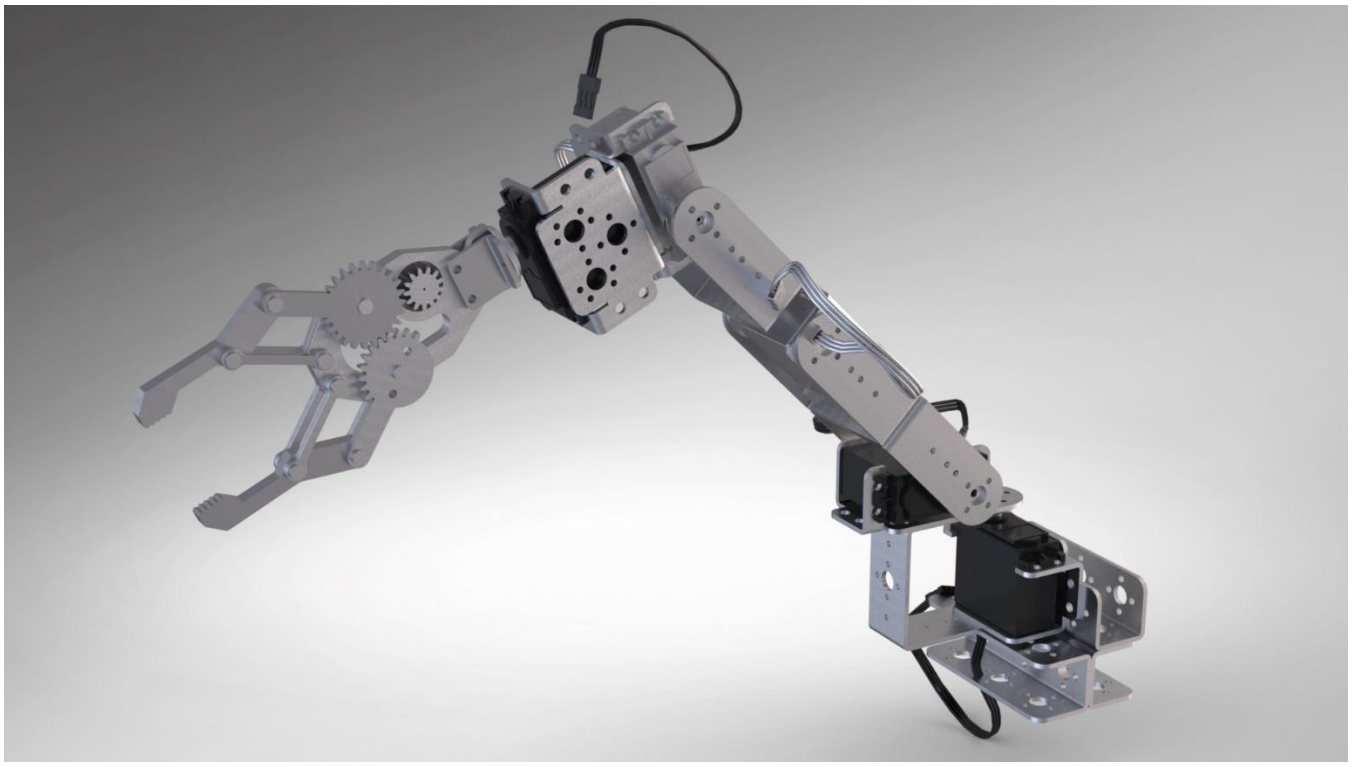

Figure 8 Complete Assembly of Robot

2021

Submitted on: Jan 6

Submitted to: Sir Khuwaja Fahad Iqbal

Robotics Project

Arduino Programming of ROT3U

Rohail Ahmad Malik- 270952

Sameer Bin Khalid - 257215

Usman Shahid - 255526

Aqib Habib - 266993

Table of Contents

Table of Figures

Abstract:

This paper introduces a chess-playing robotic system that is designed to autonomously play board games against human opponents. The control of the robotic arm manipulator is addressed in terms of speed and position control. It was implemented using Arduino Microcontroller.

Introduction:

On May 11, 1997, IBM's Deep Blue computer defeated one of the best human chess player Garry Kasparov. However, when saying a computer is not entirely true. In reality, the computer only calculated the movements and a human still assisted him for executing the movement on the chessboard. Board games are rich issues for human robot cooperation research because these games have an intermediate and an easily adjustable degree of structure. Perception of the chessboard and game pieces, observation of the human, game state and thinking about the game, and manipulation of the chess pieces while playing with the human opponent was involved. Development on physical chessboard game playing systems opens the way for more general human-robot interactions systems that considered less structure. For example, this type of work could lead in the end to a manipulator capable of helping engineering as a field assistant that performs manipulation tasks in different field environments.

Chosen Application

The robotics gripper can be used for many applications such for the entertainment industry or for more practical purposes such as for the assembly industries.

We have used our robot to play chess against a human player using the applications of the inverse kinematics and algorithms from artificial intelligence such as alpha beta-pruning and minimax.

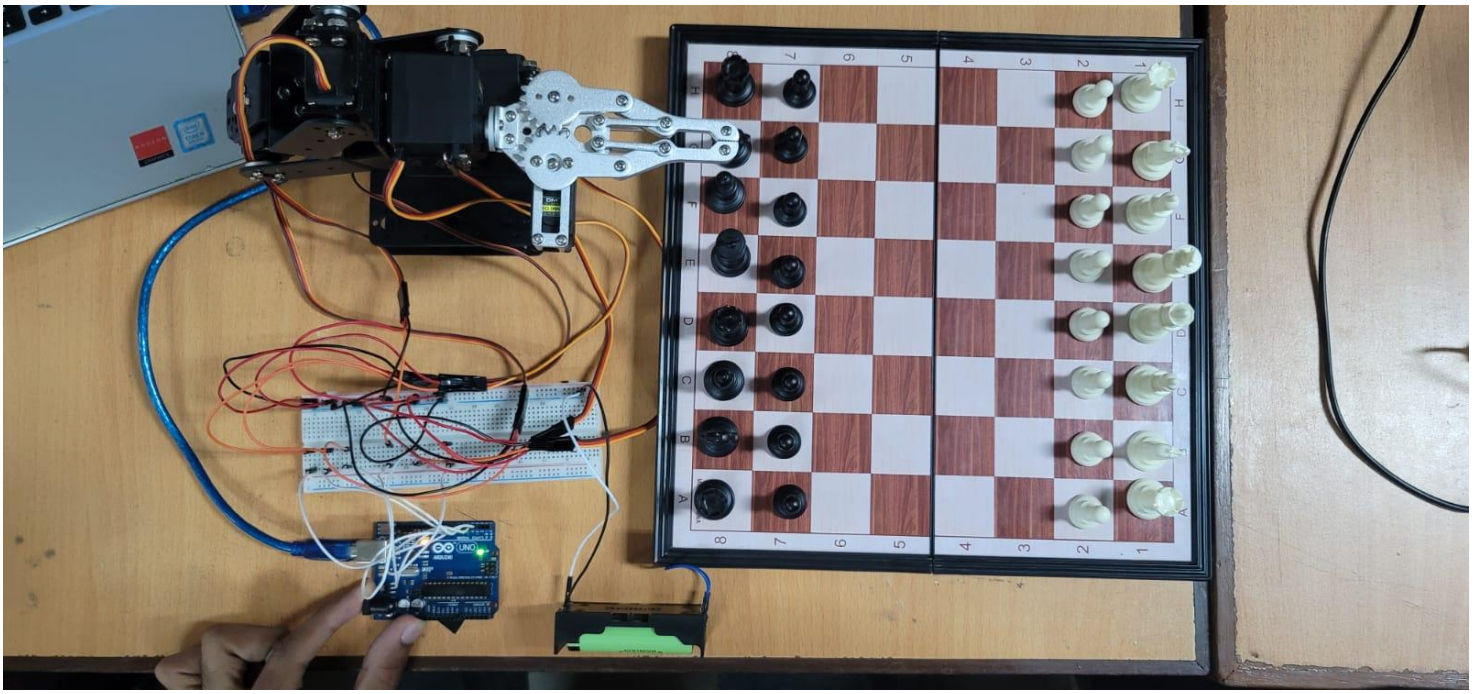

Figure 1 ROT3U and Chess Board

Analytical Solution to Inverse Kinematics

For the robotic arm to operate (i.e., pick up and place the chess pieces), the joint angles are required to achieve the desired pose. In particular, the position of the end-effector relative to the base frame is of primary concern since the gripper needs to be only in a plane parallel to the ground to pick and place the chess pieces. Hence, the orientation is not of much concern and the inverse kinematics only require the desired coordinates of the end-effector.

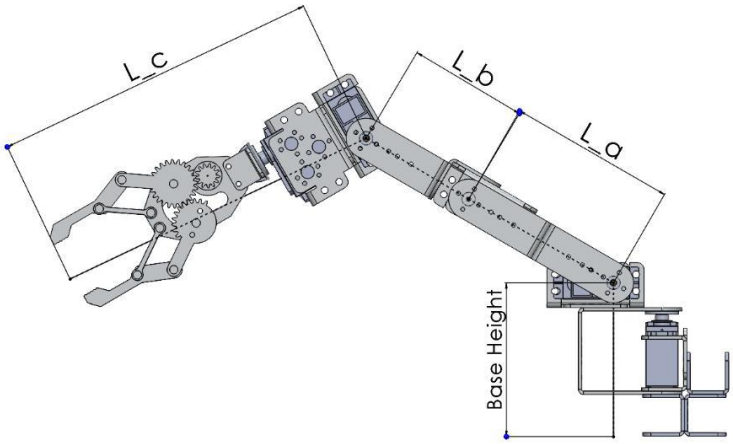

Figure 2: Side view of the Robot

Note that the last two servo motors do not affect the position of the end effector (the last motor controls the gripper while the second to last motor only rotates it about the axis of symmetry of the gripper). Hence, for inverse kinematics only involving the position of the end effector, only first four servo motors (and thus joint angles $\theta_1, \theta_2, \theta_3$ and θ_4) need to be found. Therefore, the robotic arm can be simplified from 6 DOF to 4 DOF, as shown in Figure 2. The lengths l_a, l_b and l_c are also defined as shown in the same figure.

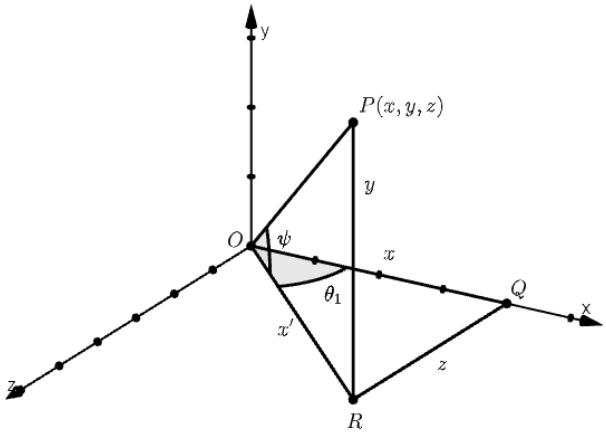

Figure 3: Position of point P(x ,y ,z) in the workspace

Consider a point $P(x, y, z)$ in the workspace of the robot as shown in Figure 3. The position vector of point P, OP can be resolved into rectangular components along the axes, as shown in the figure. Let the projection of OR on the zx-plane be x'. Then applying the Pythagorean principle for triangle ΔOQR gives the following values:

$$x' = \sqrt{x^2 + z^2}$$

$$\theta_1 = tan^{-1}\frac{z}{x}$$

Where θ_1 is the azimuthal angle between the xy-plane and the plane of ΔORP. In the case of the robot, the rotation corresponding to this angle will be performed by the first motor, placing the rest of the robot into the plane of ΔORP. Let the elevation angle of OP be ψ. Then, using the trigonometric ratios on ΔORP:

$$\psi = tan^{-1}\frac{y}{x'}$$

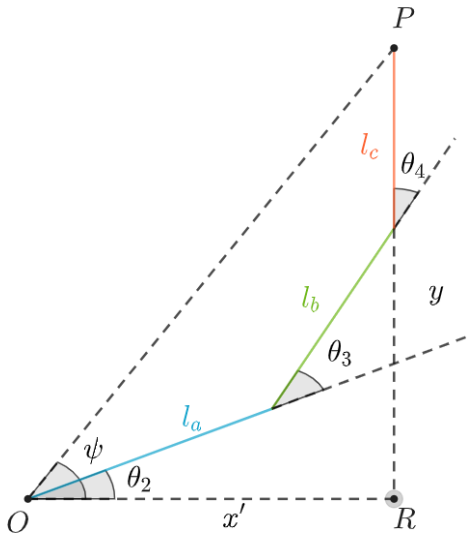

Figure 4: Planar simplification in the plane of ORP

Now consider the robotic manipulator in plane of ΔORP as shown in Figure 4. From the geometry in the figure, it can be noted that:

$$x' = l_a\, cos\,\theta_2 + l_b\, cos(\theta_2 + \theta_3) + l_c\, cos(\theta_2 + \theta_3 + \theta_4)\ 1$$

$$y = \mathrm{l_a}\sin\theta_2 + \mathrm{l_b}\sin(\theta_2 + \theta_3) + \mathrm{l_c}\sin(\theta_2 + \theta_3 + \theta_4)$$

The angle ψ is the sum of all three planar joint angles:

$$\psi = \theta_2 + \theta_3 + \theta_4$$

Substitution of the above equation in the previous two equations, followed by algebraic manipulation yields:

$$x' - l_c\, cos\,\psi = l_a\, cos\,\theta_2 + l_b\, cos(\theta_2 + \theta_3)$$

$$\mathrm{y} - l_c\, sin\,\psi = l_a\, sin\,\theta_2 + l_b\, sin(\theta_2 + \theta_3)$$

Let $x' - l_c\, cos\,\psi = x''$ and $y - l_c\, sin\,\psi = y''$ in the above equations. It thus follows that:

$$x'' = l_a\, cos\,\theta_2 + l_b\, cos(\theta_2 + \theta_3)$$

$$y'' = l_a\, sin\,\theta_2 + l_b\, sin(\theta_2 + \theta_3)$$

The equations above are the governing equations for the inverse kinematics of a DOF planar manipulator. Solving both equations simultaneously results in the expressions of joint angles θ_3 and θ_2 as follows:

$$\theta_3 = cos^{-1}\frac{x''^2 + y''^2 - l_a^2 - l_b^2}{2l_a l_b}$$

$$\theta_2 = tan^{-1}\frac{y''}{x''} - tan^{-1}\frac{l_b\, sin\,\theta_3}{l_a + l_b cos\theta_3}$$

The fourth joint angle can be found as follows:

$$\theta_4 = \psi - (\theta_2 + \theta_3)$$

Arduino Programming:

Before we can begin our actual programming of the robot, we need to calibrate the servo motors and robot itself since it is a very old robot.

ROT3U Calibration:

During the calibration of the robot, we had to disassemble the whole brackets and servos. The first step is manually resting the angles to 0 of each servo by using Arduino.

After resetting the servos, a home position was set for the robot as shown in **Figure 1** which will then be used as the starting position of the robot.

A function has been created to automatically the calibrated position once the code starts running and has been mentioned in

APPENDIX I.

Programming:

To move the robot to desired x, y, z position, a function has been written which calculates the joint angles of the robot corresponding to the give coordinates. This function will be used to move the chess pieces accordingly when interfaced with the algorithms of artificial intelligence.

Initially, the two set of x, y, z coordinates have been bound with two high and low buttons to check the program. The buttons have the required x, y, z coordinates set, and the robot ideally should follow and hold these coordinates until next button is pressed.

In the real program, the positions of each box of chess will be bind with an IR controller for easy control of the robot.

The complete set of code has been mentioned in

APPENDIX I**.**

Future Prospects:

1. Image recognition algorithm to detect the chess pieces on the chessboard. This would help identify the opponent's move. Alternatively Smart Chess Board can also be used to detect the human opponent's move.
2. A gripper for the robot specialized for picking and dropping the chess pieces.
3. A local min-max search algorithm can be implemented to increase chess engine strength.
4. A trajectory tracking control scheme, considering the robot dynamic model in order to perform effective trajectory tracking capabilities like welding, cutting, and speed-controlled tasks.

Conclusion

In this project, we present a Robotic Manipulator Arm, that is capable of playing chess using inverse kinematics. To make it intelligent, we can make it such that it first uses a computer vision system to continuously monitor the chess board and track the movements on the board and then apply different algorithms like alpha beta pruning or mini max algorithm in order to find the best possible solution and finally, execute it. We applied a lot of our previous as well as newly found applied knowledge regarding robotics. The code implements both forward and inverse kinematics on MATLAB and Arduino.

APPENDIX I

Arduino Code:

```
#include <Servo.h>
#include <math.h>
// Constant Robot Lengths
const float Base_Length= .096; // Length in Meters
const float Humerus_Length= .1055; //Length in Meters
const float Ulna_Length= .098; //Length in Meters
const float Gripper_Length= .168; //Length in Meters
//Variables for calculation of thetas
float theta;
float x_pp;
float y_pp;
float theta_1;
float theta_3;
float theta_2;
float theta_4;
float x_p;
//
float x_home; //define home x position
float y_home; //define home y position
float z_home; //define home z position
//
float theta_1_home=0;  //home theta_1 position
float theta_2_home=0;  //home theta_2 position
float theta_3_home=180; //home theta_3 position
float theta_4_home=0; //home theta_4 position
float theta_5_home=0; //home theta_5 position
float theta_6_home=60; //home theta_6 position
//home theta_4 position
//
float theta_1_write=theta_1_home;  //theta 1 to write to servos
float theta_2_write=theta_2_home; //theta 2 to write to servos
float theta_3_write=theta_3_home; //theta 3 to write to servos
float theta_4_write=theta_4_home; //theta 4 to write to servos
//
// Servo positions for Button 1
float x_button_1=0.2;
float y_button_1=0;
float z_button_1=-0.03;
// Servo positions for Button 2
float x_button_2=0.2;
float y_button_2=0.1;
```

```
float z_button_2=-0.03;
//Home Angles for Joints
Servo servo_1; // Base Servo
Servo servo_2; // Humerus Servo
Servo servo_3; // Ulna Servo
Servo servo_4; // Gripper Servo
Servo servo_5; // End Effector Rotation Servo
Servo servo_6; // End Effector Arm Servo
bool buttonState_1=0;
bool buttonState_2=0;
int buttonPin_1; //set accordingly
int buttonPin_2; //set accordingly
void setup() {
servo_1.attach(2); // set pins of servo
servo_2.attach(3); // set pins of servo
servo_3.attach(4); // set pins of servo
servo_4.attach(5); // set pins of servo
servo_5.attach(6); // set pins of servo
servo_6.attach(7); // set pins of servo
Serial.begin(9600);
calibrate();
// calibrate();
}
void loop() {
buttonState_1 = digitalRead(buttonPin_1);
buttonState_2 = digitalRead(buttonPin_2);
  // check if the pushbutton sis pressed. If it is, the buttonState is HIGH:
  if (buttonState_1 == HIGH) {


    // Move the Gripper to Position 1
    inv_kin(x_button_1,y_button_1,z_button_1); // gripper_pos is a function taking three variables x,y,z and calculates automatically the
joint angles of the servo motors
    // which will thenbe used to move the gripper to position 1
    while(theta_1_write!=theta_1) { //Theta 1 will be the angle of base and it will move from home theta to new calculate theta
     servo_1.write(theta_1_write);
    if (theta_1_write>theta_1)
    theta_1_write+=1 ;
    else
    theta_1_write-=1;
    }
     while(theta_2_write!=theta_2) { //Theta 2 will be the angle of base and it will move from home theta to new calculate theta
    servo_2.write(theta_2_write);
    if (theta_2_write>theta_2)
    theta_2_write+=1 ;
    else
    theta_2_write-=1;
    }
     while(theta_3_write!=theta_3) { //Theta 3 will be the angle of base and it will move from home theta to new calcualte theta
```

```
    servo_3.write(theta_3_write);
    if (theta_3_write>theta_3)
    theta_3_write+=1 ;
    else
    theta_3_write-=1;
     }
    while(theta_4_write!=theta_4) { //Theta 4 will be the angle of base and it will move from home theta to new calculate theta
    servo_4.write(theta_4_write);
    if (theta_4_write>theta_4)
    theta_4_write+=1 ;
    }
  }
   //these four loops are for button 1
  else {
    // gripper_pose(x_home,y_home,z_home) // either define home function or home coordinates should be known
    // Gripper at Home Position
  }
// This is the end of function for button 1
//
  if (buttonState_2 == HIGH) {
    // Move the Gripper to Position 1
    inv_kin(x_button_2,y_button_2,z_button_2); // gripper_pos is a function taking three variables x,y,z and calculates automatically the
joint angles of the servo motors
    // which will thenbe used to move the gripper to position 1
    while(theta_1_write!=theta_1) { //Theta 1 will be the angle of base and it will move from home theta to new calcualte theta
    servo_1.write(theta_1_write);
    if (theta_1_write>theta_1)
    theta_1_write+=1 ;
    else
    theta_1_write-=1;
    }
     while(theta_2_write!=theta_2) { //Theta 2 will be the angle of base and it will move from home theta to new calcualte theta
    servo_2.write(theta_2_write);
    if (theta_2_write>theta_2)
    theta_2_write+=1 ;
    else
    theta_2_write-=1;
    }
     while(theta_3_write!=theta_3) { //Theta 3 will be the angle of base and it will move from home theta to new calcualte theta
    servo_3.write(theta_3_write);
    if (theta_3_write>theta_3)
    theta_3_write+=1 ;
    else
    theta_3_write-=1;
     }
    while(theta_4_write != theta_4) { //Theta 4 will be the angle of base and it will move from home theta to new calcualte theta
    servo_4.write(theta_4_write);
    if (theta_4_write>theta_4)
```

```
    theta_4_write+=1 ;
    else
    theta_4_write-=1;
    }
  }
   //these four loops are for button 2
  else {
    // gripper_pose(x_home,y_home,z_home) // either define home function or home coordinates should be known
    // Gripper at Home Position
  }
}
// This is the end of function for button 1
//
// Function to bring the robot to its home position
void calibrate(){
servo_1.write(theta_1_home);
delay(500);
servo_2.write(theta_2_home);
delay(500);
servo_3.write(theta_3_home);
delay(500);
servo_4.write(theta_4_home);
delay(500);
servo_5.write(theta_5_home);
delay(500);
servo_6.write(theta_6_home);
 }
//Function to calculate the joint angles from inverse kinematics
void inv_kin(float x,float y, float z) {
float e = pow(x,2)+pow(z,2);
x_p = sqrt(e);
float d=y/x_p;
theta = atan(d);
x_pp = x_p - Gripper_Length*cos(theta);
y_pp = y-Gripper_Length*sin(theta);
theta_1 = atan(z/x);
theta_2 = atan(y_pp/x_pp)-atan((Ulna_Length*sin(theta_3))/(Humerus_Length+Ulna_Length*cos(theta_3)));
theta_3 = acos((pow(x_pp,2)+pow(y_pp,2)-pow(Ulna_Length,2)-pow(Ulna_Length,2)/(2*Humerus_Length*Ulna_Length)));
theta_4 = theta - theta_2 - theta_3;
// Joint angles
theta_1*=(180/3.1419);
theta_2*=(180/3.1419);
theta_3*=(180/3.1419);
theta_4*=(180/3.1419);
float q[1][4] = {theta_1,theta_2,theta_3,theta_4};
 }
```