Warsaw University of Technology

Group Project

EMARO 2016-2018

Final Report

Ahmad Adee
Debaleena Misra
Fernando Nunez M

May 14, 2017

Contents

List of Figures

1 Kinematic Structure of Robot Manipulator

The robot to be modeled is a manipulator with 5 degree of freedom of type RRRRR. A simple 3D model desing is presented below.

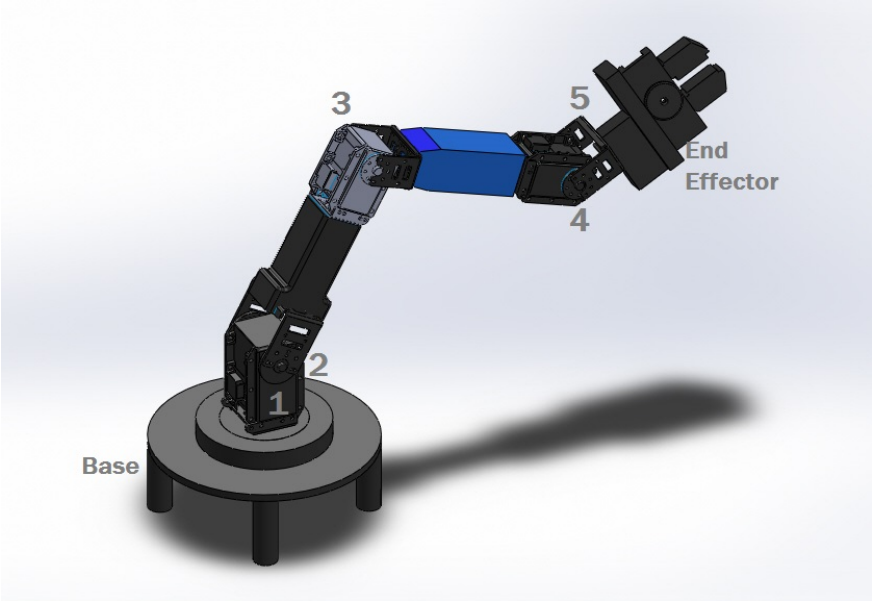


Figure 1: Simple 3D model of the manipulator

Five joints, the base of the manipulator and an end effector can be appreciated in the picture 1.

2 Denavit-Hartenberg Parameters

In order to solve the direct and inverse kinematic problems, the Denavit Hartenberg parameters are obtained. An image representing the joints of the manipulator with their different assigned axis and axis of rotation is shown in figure 2.

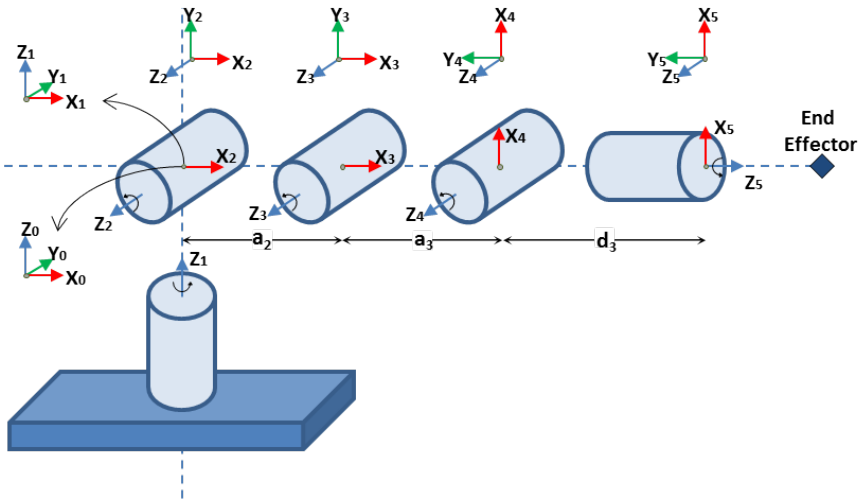


Figure 2: Picture of manipulator's links for the DH parameters calculation

This axis representation was used for the DH parameter derivation showed in table 1.

i	a_{i-1}	α_{i-1}	d_i	θ_i
0	0	0	0	θ_1
1	0	$\frac{\pi}{2}$	0	θ_2
2	a_2	0	0	θ_3
3	a_3	0	0	θ_4
4	0	$\frac{\pi}{2}$	d_5	θ_5

Table 1: DH parameters

3 Direct Kinematic Problem

The serial manipulator consists of a fixed base, a series of links connected by joints, and ending at a free end carrying the tool or the end-effector. By actuating the joints, one can position and orient the end-effector in a plane or in three-dimensional (3D) space to perform desired tasks with the end-effector. In the Direct Kinematics problem, given the link parameters and the joint variables, a_{i-1}, α_{i-1}, d_i, and $_i$, we seek to find the position of the last link in the fixed or reference coordinate system for a given set of joint angles. The serial manipulator geometries are already described using the well-known Denavit-Hartenberg (D-H) parameters in the earlier section.

Homogeneous matrix for each link is given by:

$$T_i^{i-1} = Rot(x_{i-1}, \alpha_{i-1})\ Trans(x_{i-1}, a_{i-1})\ Rot(z_i, \theta_i)\ Trans(z_i, d_i) \quad (1)$$

which defines the transformation of the frame (i) relative to frame $(i-1)$

$$T_i^{i-1} = \begin{bmatrix} C\theta_i & -S\theta_i & 0 & a_{i-1} \\ S\theta_i C\alpha_{i-1} & C\theta_i C\alpha_{i-1} & -S\alpha_{i-1} & -d_i S\alpha_{i-1} \\ S\theta_i S\alpha_{i-1} & C\theta_i S\alpha_{i-1} & C\alpha_{i-1} & d_i C\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

3.1 Transformations of individual links

Homogeneous matrix between links are presented bellow. They represent the pose of a link seen from point of view of the previous link. Combining them by multiplication results in the Final Transformation matrix which will give the pose of the end-effector seen from the base.

$$T_1^0 = \begin{bmatrix} C\theta_1 & -S\theta_1 & 0 & 0 \\ S\theta_1 & C\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_2^1 = \begin{bmatrix} C\theta_2 & -S\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ S\theta_2 & C\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^2 = \begin{bmatrix} C\theta_3 & -S\theta_3 & 0 & a_2 \\ S\theta_3 & C\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_4^3 = \begin{bmatrix} C\theta_4 & -S\theta_4 & 0 & a_3 \\ S\theta_4 & C\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_5^4 = \begin{bmatrix} C\theta_5 & -S\theta_5 & 0 & 0 \\ 0 & 0 & -1 & -d_5 \\ S\theta_5 & C\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.2 Final transformation

The homogeneous matrix that defines the position of the last link with respect to the base frame is obtained through the multiplication of the individual link matrices

$$T_5^0 = T_1^0 \; T_2^1 \; T_3^2 \; T_4^3 \; T_5^4 \tag{3}$$

The intermediate composition matrices are given below :

$$T_2^0 = \begin{bmatrix} C\theta_1 C\theta_2 & -C\theta_1 S\theta_2 & S\theta_1 & 0 \\ C\theta_2 S\theta_1 & -S\theta_1 S\theta_2 & -C\theta_1 & 0 \\ S\theta_2 & C\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^0 = \begin{bmatrix} C\theta_1 C\theta_{23} & -C\theta_1 S\theta_{23} & S\theta_1 & C\theta_1 C\theta_2 a_2 \\ C\theta_{23} S\theta_1 & -S\theta_1 S\theta_{23} & -C\theta_1 & C\theta_2 S\theta_1 a_2 \\ S\theta_{23} & C\theta_{23} & 0 & S\theta_2 a_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^0 = \begin{bmatrix} C\theta_1 C\theta_{234} & -C\theta_1 S\theta_{234} & S\theta_1 & C\theta_1(C\theta_2 a_2 + C\theta_{23} a_3) \\ C\theta_{234} S\theta_1 & -S\theta_1 S\theta_{234} & -C\theta_1 & S\theta_1(C\theta_2 a_2 + C\theta_{23} a_3) \\ S\theta_{234} & C\theta_{234} & 0 & S\theta_2 a_2 + S\theta_{23} a_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_5^0 = \begin{bmatrix} S\theta_1 S\theta_5 + C\theta_1 C\theta_5 C\theta_{234} & C\theta_5 S\theta_1 - C\theta_1 C\theta_{234} S\theta_5 & C\theta_1 S\theta_{234} & C\theta_1(C\theta_2 a_2 + C\theta_{23} a_3 + S\theta_{234} d_5) \\ C\theta_5 S\theta_1 C\theta_{234} - C\theta_1 S\theta_5 & -C\theta_1 C\theta_5 - S\theta_1 C\theta_{234} S\theta_5 & S\theta_1 S\theta_{234} & S\theta_1(C\theta_2 a_2 + C\theta_{23} a_3 + S\theta_{234} d_5) \\ C\theta_5 S\theta_{234} & -S\theta_5 S\theta_{234} & -C\theta_{234} & S\theta_2 a_2 - C\theta_{234} d_5 + S\theta_{23} a_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The effects of the base and end-effector of the manipulator has been excluded from the model so far. Introducing them into the kinematics would be trough

the translation matrix for both of them, one at the beginning and the other at the end. The final homogeneous matrix from base $< 0, 0, 0 >$ coordinates to the end-effector would be:

$$T_e^b = T_0^b \; T_1^0 \; T_2^1 \; T_3^2 \; T_4^3 \; T_5^4 \; T_e^5 \tag{4}$$

where T_e^b represents the matrix that transform the coordinate frame from the base until the end-effector, T_0^b transform from base to joint 0, T_e^5 from joint 5 to end-effector. Therefore, the final transformation matrix would be:

$$T_e^b = T_0^b \; T_5^0 \; T_e^5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & b \\ 0 & 0 & 0 & 1 \end{bmatrix} \; T_5^0 \; \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & e \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

$$T_e^b = \begin{bmatrix} S\theta_1 S\theta_5 + C\theta_1 C\theta_5 C\theta_{234} & C\theta_5 S\theta_1 - C\theta_1 C\theta_{234} S\theta_5 & C\theta_1 S\theta_{234} & C\theta_1(C\theta_2 a_2 + C\theta_{23} a_3 + S\theta_{234}(d_5 + e)) \\ C\theta_5 S\theta_1 C\theta_{234} - C\theta_1 S\theta_5 & -C\theta_1 C\theta_5 - S\theta_1 C\theta_{234} S\theta_5 & S\theta_1 S\theta_{234} & S\theta_1(C\theta_2 a_2 + C\theta_{23} a_3 + S\theta_{234}(d_5 + e)) \\ C\theta_5 S\theta_{234} & -S\theta_5 S\theta_{234} & -C\theta_{234} & b + S\theta_2 a_2 - C\theta_{234}(d_5 + e) + S\theta_{23} a_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

where b and e represent the position in the Z axis of the base and end-effector respectively.

3.3 Forward Kinematics Validation

For the purpose of validation of our transformation matrices obtained for the forward kinematics, we use the Robotics Toolbox. This free open source toolbox provides many functions that are useful for the study and simulation of classical arm-type robotics [ref]. A serial link manipulator is created using the D-H parameters for every joint, and specifying the D-H convention used ('Modified' in our case). We use a sample joint space $q = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5] = [0, 0, 0, 0, 0]$ and use the numerical values for the offset variables: $a_2 = 14.7cm$, $a_3 = 14.9cm$ and $d_5 = 3cm$. Finally the fkine() function provides us the final matrix of transformation for the given manipulator model. We compare the results with the values obtained using the transformation matrices we obtained in the earlier section.

The MATLAB script for the validation is shown below:

```
%DH Parameters
a2=14.7; a3=14.9; d5=3;

base=11.8; b=base; tool=9.9;e=tool;
baseSize=[0 0 11.8 1]';
toolSize=[0 0 9.9 1]';

%Describing the link and joint structures
```

```
L1 = Link('d', 0, 'a', 0, 'alpha', 0, 'modified');
L2 = Link('d', 0, 'a', 0, 'alpha', pi/2, 'modified');
L3 = Link('d', 0, 'a', a2, 'alpha', 0, 'modified');
L4 = Link('d', 0, 'a', a3, 'alpha', 0, 'modified');
L5 = Link('d', d5, 'a', 0, 'alpha', pi/2, 'modified');

%Forming the serial link
bot = SerialLink([L1 L2 L3 L4 L5]);

%Adding bases and tool sizes
bot.base(:,4)=baseSize;
bot.tool(:,4)=toolSize;

%Initial angles set to zero
qz=[0 0 0 0 0];

%Forward Kinematics
T_RTB= fkine(bot,qz);

%Comparison with Matrix multiplication
theta1 = qz(1);
theta2 = qz(2);
theta3 = qz(3);
theta4 = qz(4);
theta5 = qz(5);

S1 = sin(theta1);
S2 = sin(theta2);
S3 = sin(theta3);
S4 = sin(theta4);
S5 = sin(theta5);
C1 = cos(theta1);
C2 = cos(theta2);
C3 = cos(theta3);
C4 = cos(theta4);
C5 = cos(theta5);

%% DKP MATRIXES
Tb0=[1   0   0  0;
     0   1   0  0;
     0   0   1  base;
     0   0   0  1];

T01=[C1 -S1  0  0;
     S1   C1 0  0;
      0   0  1  0;
```

```
55          0   0   0  1];
56
57  T12=[C2 -S2  0  0;
58       0   0  -1  0;
59       S2  C2  0  0;
60       0   0   0  1];
61
62  T23=[C3 -S3  0  a2;
63       S3  C3  0  0;
64        0   0  1  0;
65        0   0  0  1];
66
67  T34=[C4 -S4  0  a3;
68       S4  C4  0  0;
69       0   0  1  0;
70       0   0  0  1];
71
72  T45=[C5 -S5  0  0;
73       0   0 -1 -d5;
74       S5  C5  0  0;
75       0   0  0  1];
76
77   T5e=[1  0  0  0;
78       0  1  0  0;
79       0  0  1  tool;
80       0  0  0  1];
81
82  % MATRIX COMBINATION FOR DKP
83  T05=T01*T12*T23*T34*T45;
84
85  %Final DKP matrix from base to end-effector%
86  Tbe=Tb0*T01*T12*T23*T34*T45*T5e;
```

4 Inverse Kinematic Problem

In the inverse kinematics problem, for a given end-effector position and orientation, we seek to find the values of the joint variables that will realize it. We obtain a set of analytical equations to calculate the joint angles that would lead to reach the desired position of the end-effector with respect to the base frame.

Assume the end-effector is in a desired generic position given by

$$ {}_dT_e^b = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7} $$

This matrix is the generic form of the matrix T_e^b from 6, which can take numerical values for each one of the joint angles θ_i and constant values for the manipulator structure: a_2, a_3, d_5, b and e.

$$ {}_dT_e^b = T_e^b = T_0^b \; T_1^0 \; T_2^1 \; T_3^2 \; T_4^3 \; T_5^4 \; T_e^5 \tag{8} $$

To find the inverse kinematics of the first joint as a function of the known elements of T

$$ \left(T_0^b T_1^0\right)^{-1} {}_dT_e^b = T_2^1 \; T_3^2 \; T_4^3 \; T_5^4 \; T_e^5 \tag{9} $$

In the LHS we have,

$$ \left(T_0^b T_1^0\right)^{-1} {}_dT_e^b = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -b \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C_1 & -S_1 & 0 & 0 \\ S_1 & -C_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

which gives

$$ LHS = \begin{bmatrix} C_1 r_{11} + S_1 r_{21} & C_1 r_{12} + S_1 r_{22} & C_1 r_{13} + S_1 r_{23} & C_1 p_x + S_1 p_y \\ C_1 r_{21} - S_1 r_{11} & C_1 r_{22} - S_1 r_{12} & C_1 r_{23} - S_1 r_{13} & C_1 p_y - S_1 p_x \\ r_{31} & r_{32} & r_{33} & p_z - b \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10} $$

In the RHS we have

$$T_e^1 = T_2^1\ T_3^2\ T_4^3\ T_5^4\ T_e^5$$

The above matrix is:

$$T_e^1 = \begin{bmatrix} C_5C_{234} & -C_{234}S_5 & S_{234} & C_2a_2 + C_{23}a_3 + S_{234}(d_5 + e) \\ -S_5 & -C_5 & 0 & 0 \\ C_5S_{234} & -S_{234}S_5 & -C_{234} & S_2a_2 + S_{23}a_3 - C_{234}(d_5 + e) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

4.1 Joint 1

Equating element $(2, 4)$ of both sides of the T_e^1 matrix

$$C_1p_y - S_1p_x = 0 \tag{12}$$

$$\implies C_1p_y = S_1p_x$$
$$\implies tan(\theta_1) = \frac{p_y}{p_x}$$

$$\implies \theta_1 = arctan\left(\frac{p_y}{p_x}\right) \tag{13}$$

4.2 Joint 5

Equating element $(2, 2)$ of both sides of the T_e^1 matrix:

$$C_1r_{22} - S_1r_{12} = -C_5 \tag{14}$$

$$C_5 = S_1r_{12} - C_1r_{22}$$

$$\theta_5 = Atan2\left(\pm\sqrt{1 - (S_1r_{12} - C_1r_{22})^2}, S_1r_{12} - C_1r_{22}\right) \tag{15}$$

4.3 Joint 3

Using T_e^1 element $(1,4)$: $C_2a_2 + C_{23}a_3 + S_{234}(d_5+e) = C_1p_x + S_1p_y$

Using T_e^1 element $(3,4)$: $S_2a_2 + S_{23}a_3 - C_{234}(d_5+e) = p_z - b$

Squaring both the above equations :

$$C_2^2a_2^2 + C_{23}^2a_3^2 + 2C_2a_2C_{23}a_3 = (C_1p_x + S_1p_y)^2 + (d_5+e)^2S_{234}^2 - 2(C_1p_x + S_1p_y)((d_5+e)S_{234})$$

$$S_2^2a_2^2 + S_{23}^2a_3^2 + 2S_2a_2S_{23}a_3 = (p_z-b)^2 + (d_5+e)^2C_{234}^2 + 2(p_z-b)(d_5+e)C_{234}$$

Adding the squared equations we obtain :

$a_2^2 + a_3^2 + 2a_2a_3C_3 = (p_z-b)^2 + (d_5+e)^2 + C_1^2p_x^2 + S_1^2p_y^2 + 2C_1S_1p_xp_y + 2(p_z - b)(d_5+e)C_{234} - 2p_x(d_5+e)C_1S_{234} - 2p_y(d_5+e)S_1S_{234}$

$\implies a_2^2 + a_3^2 + 2a_2a_3C_3 = (p_z-b)^2 + (d_5+e)^2 + C_1^2p_x^2 + S_1^2p_y^2 + 2C_1S_1p_xp_y + 2(p_z-b)(d_5+e)C_{234} - S_{234}(2p_x(d_5+e)C_1 + 2p_y(d_5+e)S_1)$

Substituting

T_e^1 element $(3,1)$: $C_5S_{234} = r_{31}$
T_e^1 element $(3,3)$: $C_{234} = -r_{33}$

This gives us :

$a_2^2 + a_3^2 + 2a_2a_3C_3 = (p_z-b)^2 + (d_5+e)^2 + C_1^2p_x^2 + S_1^2p_y^2 + 2C_1S_1p_xp_y + 2(p_z - b)(d_5+e)(-r_{33}) - (\frac{r_{31}}{C_5})(2p_x(d_5+e)C_1 + 2p_y(d_5+e)S_1)$

$\implies 2a_2a_3C_3 = (p_z-b)^2 + (d_5+e)^2 + C_1^2p_x^2 + S_1^2p_y^2 + 2C_1S_1p_xp_y + 2(p_z - b)(d_5+e)(-r_{33}) - (\frac{r_{31}}{C_5})(2p_x(d_5+e)C_1 + 2p_y(d_5+e)S_1) - a_2^2 - a_3^2$

Let the RHS be:

$$W = (p_z-b)^2 + (d_5+e)^2 + C_1^2p_x^2 + S_1^2p_y^2 + 2C_1S_1p_xp_y + 2(p_z-b)(d_5+e)(-r_{31}) - (\frac{r_{31}}{C_5})(2p_x(d_5+e)C_1 + 2p_y(d_5+e)S_1) - a_2^2 - a_3^2 \quad (16)$$

$$C_3 = \frac{W}{2a_2a_3}$$

$$\implies \theta_3 = Atan2\left(\pm\sqrt{1-(\frac{W}{2a_2a_3})^2}, \frac{W}{2a_2a_3}\right)$$

4.4 Joint 2

Using T_e^1 element $(3,4)$:

$$S_2a_2 + S_{23}a_3 - C_{234}(d_5+e) = (p_z - b) \tag{17}$$

$$\implies a_2s_2 + a_3(S_2C_3 + S_3C_2) = (p_z-b) + (d_5+e)C_{234}$$
$$\implies S_2(a_2 + a_3C_3) = (p_z-b) + (d_5+e)C_{234} - a_3S_3C_2$$

$$\implies S_2 = \frac{(p_z-b) + (d_5+e)C_{234} - a_3S_3C_2}{a_2 + a_3C_3} \tag{18}$$

Using T_e^1 element $(1,4)$:

$$C_2a_2 + C_{23}a_3 + S_{234}(d_5+e) = C_1p_x + S_1p_y \tag{19}$$

$$\implies a_2C_2 + a_3(C_2C_3 - S_2S_3) = C_1p_x + S_1p_y - S_{234}(d_5+e)$$
$$\implies C_2(a_2 + a_3C_3) = C_1p_x + S_1p_y - S_{234}(d_5+e) + a_3S_2S_3$$

Let $C_1p_x + S_1p_y - S_{234}(d_5+e) = M$. Substituting this in the above equation and using the value of S_2 we get:

$$C_2(a_2 + a_3C_3) = M + a_3S_3\frac{(p_z-b) + d_5C_{234} - a_3S_3C_2}{a_2 + a_3C_3}$$
$$\implies C_2(a_2 + a_3C_3) = M - \frac{a_3^2S_3^2C_2}{a_2 + a_3C_3} + \frac{a_3S_3((p_z-b) + (d_5+e)C_{234})}{a_2 + a_3C_3}$$

Substituting $N = \frac{a_3S_3((p_z-b) + (d_5+e)C_{234})}{a_2 + a_3C_3}$ and rewriting the above equation:

$$C_2(a_2 + a_3C_3) + \frac{a_3^2S_3^2C_2}{a_2 + a_3C_3} = M + N$$
$$\implies C_2\frac{(a_2 + a_3C_3)^2 + a_3^2S_3^2}{a_2 + a_3C_3} = M + N$$

Substituting $Q = \frac{(a_2 + a_3C_3)^2 + a_3^2S_3^2}{a_2 + a_3C_3}$ we can find C_2 as:

$$C_2 = \frac{M+N}{Q}$$

$$\implies \theta_2 = Atan2\left(\pm\sqrt{1 - (\frac{M+N}{Q})^2}, \frac{M+N}{Q}\right) \tag{20}$$

4.5 Joint 4

Using T_e^1 element $(3,3)$:

$$C_{234} = -r_{33} \tag{21}$$

$$\implies \cos(\theta_2 + \theta_3 + \theta_4) = -r_{33}$$

$$\implies \theta_2 + \theta_3 + \theta_4 = arctan\left(\frac{\pm\sqrt{1 - r_{33}^2}}{-r_{33}}\right)$$

$$\implies \theta_4 = arctan\left(\frac{\pm\sqrt{1 - r_{33}^2}}{-r_{33}}\right) - \theta_3 - \theta_2 \tag{22}$$

An important aspect of the inverse kinematic investigation, was to define the link limits imposed by the surrounding and the mechanical design of the manipulator itself. The exact limits of revolution were as below:

$$0 < \theta_1 < 304.8$$
$$0 < \theta_2 < 219.6$$
$$0 < \theta_3 < 227.8$$
$$0 < \theta_4 < 228.1$$
$$0 < \theta_5 < 300.7$$

4.6 Inverse Kinematics Validation

The objective in this section is to validate the analytical expressions obtained, to find the joint angles. We find the values for the position vector of the end-effector P_x, P_y and P_z from the final transformation matrix obtained in section

3.3 of Direct Kinematics Validation. These values are now used as input parameters for our inverse kinematic relations, and we obtain the joint angles as $q = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5]$. In our case, the inverse kinematics is validated correctly since we obtain the joint angles as $q = [0\,0\,0\,0\,0]$ which is the same as the angles given as input parameters to the Direct Kinematic Problem.

The MATLAB script for the validation is shown below:

```
%Inverse Kinematics Validation
T=Tbe; %Final Transformation matrix from the DKP

%Analytical Equations for the joint angles

%Joint 1
theta1i = atan2(T(2,4),T(1,4));

%Joint 5
theta5i = atan2(sqrt(1-(sin(theta1i)*T(1,2) - cos(theta1i
    )*T(2,2))^2), (sin(theta1i)*T(1,2) - cos(theta1i)*T
    (2,2)));

%Joint 3
W= (T(3,4) - b)^2 + (d5 + e)^2 + (cos(theta1i)^2 * T(1,4)
    ^2) + (sin(theta1i)^2 *T(2,4)^2) + 2 * cos(theta1i)*
    sin(theta1i)* T(1,4)* T(2,4) + 2* (T(3,4)-b)*(d5 + e)
    *(-T(3,3)) - (T(3,1)/(cos(theta5i)))*(2*T(1,4)*(d5 + e
    )*cos(theta1i) + 2*T(2,4)*(d5 + e)*sin(theta1i)) - a2
    ^2 - a3^2;
Num=1-(W/(2*a2*a3))^2; Num(Num<eps)=0;
theta3i = atan2(sqrt(Num), W/(2*a2*a3));

%Joint 2
M=cos(theta1i)*T(1,4)+sin(theta1i)*T(2,4)-(T(3,1)/cos(
    theta5i))*(d5 + e);
N=(a3*sin(theta3i)*((T(3,4) - b)+(d5 + e)*(-T(3,3))))/(a2
    +a3*cos(theta3i));
Q=((a2+a3*cos(theta3i))^2+a3^2 * sin(theta3i)^2)/(a2+a3*
    cos(theta3i));
Num2=1 -((M+N)/Q)^2; Num2(Num2<eps)=0;
theta2i=atan2(sqrt(Num2), (M+N)/Q);

%Joint 4
theta4i = atan2(sqrt(1 - T(3,3)^2), -T(3,3)) - theta3i -
    theta2i;

```

```
27 %IKP output vector for joint angles
28 q = [theta1i*180/pi theta2i*180/pi theta3i*180/pi theta4i
       *180/pi theta5i*180/pi]
```

5 Visualization

For the process of visualization of the manipulator, Robotic Toolbox for MatLab was employed. It was desired to simulate and visualize different states of the manipulator:

- Setting joint angle values and compare the end-effector position through the homogeneous matrix for DKP.
- Setting an end-effector position and compare the joints values of the manipulator through the inverse kinematic equations

5.1 Manipulator design

The Robotic Toolbox provides a function for creating a robot manipulator with serial links through DenavitHartenberg parameters specification. Then it can plot the manipulator with different joint angles values in a 3D coordinate environment. Therefore, along the DH parameters along with the DH parameters, the base and tool size had to be specified:

- Base: Through measures with the real manipulator, it was settled that the first location was $11.8cm$ over the Z axis.
- Tool: From the last joint until the end of end-effector there are $8.9cm$ in the Z-axis of the last link coordinate of reference.

These distances will affect the direct and inverse kinematic problem due to not being counted in any case ans should be subtracted from the resulting position coordinates of the end-effector. For the distance between joints in the DH parameters, the following values were selected:

$$a_2 = 14.7cm \qquad a_3 = 14.9cm \qquad d_5 = 3cm$$

Image [3] is presented an plotting of the manipulator executed by Robotic Toolbox.

5.2 DKP Visualization Comprobation

Different values for the joint angles, were implemented in the toolbox manipulator visualization:

- **All joint angle equal to zero**. $q_z = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5] = [0, 0, 0, 0, 0]$
  By setting all angles to zero into the homogeneous transformation matrix T_e^b from equation (6), the position of the end-effector is obtained:

$$T_e^b(q_z) = \begin{bmatrix} 1 & 0 & 0 & a_2 + a_3 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & b - (d_5 + e) \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 29.6 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & -1.1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{23}$$

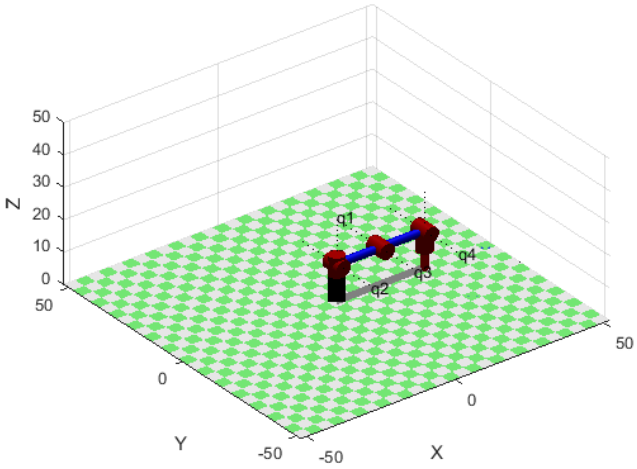
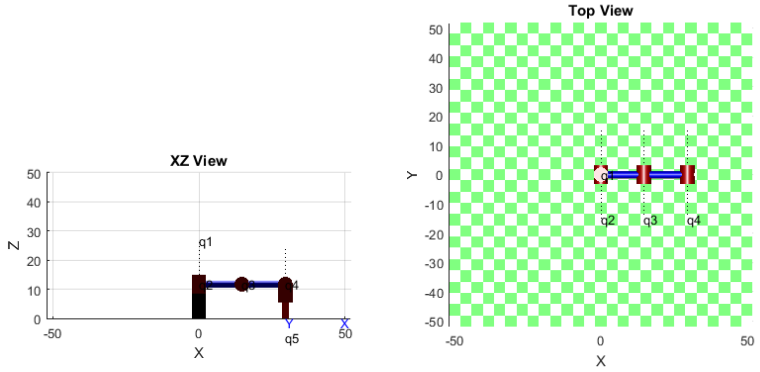


Figure 3: Manipulator plotting example

The translation position would be $< x, y, z >=< 29.6, 0, -1.1 >$ cm. The rotation too be analyze. This result can be compared with the plot provide by the robotic toolbox.

As it can be seen in the picture 4 presented below, the position of the manipulator is the same as the obtained in the calculations by the direct kinematic problem and the homogeneous transformation matrix.



(a) XZ View (b) Top View

Figure 4: Manipulator with all angles equal to zero

6 Trajectory

In this section the workspace, procedure for trajectory and some results will be shown.

6.1 Workspace

To follow a trajectory, the workspace for the robot had to be define. It was decided to work in the 9x9, $4cm^2$ each box, square table that is present in the laboratory. However, not all the coordinates in the plane are reachable as there are some angle value limits. Besides, we established that, for now, when reaching at object in the plane (near Z=0) the end-effector had to be straight pointing down, so it is easier and more feasible to grab and drop the object. This restriction reduced our workspace, eliminating some squares near the center of the plane.

A figure of the resulting workspace is shown below. The green squares represent a reachable position while the red ones an out of range position. The yellow ones are places that the manipulator in theory could reach but to accomplish it, some more work has to be done (increase a little bit of height of the point to reach). Therefore, yellow squares are declared as out of range, but on further work, they can be changed into green squares.

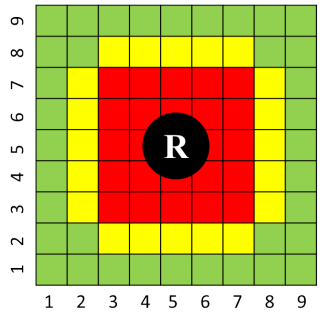


Figure 5: Workspace

The center of the plane is where the manipulator is placed. This workspace was tested in the simulation and some results will be shown later.

6.2 Algorithm

For the trajectory we implemented the results obtained from last sections: DKP and IKP, along with function for joint space trajectory generation from Peter

Corke's Robotic Toolbox. This function called jtraj, generates the trajectory from a initial set of joint values to final one, using a quintic (5th order) polynomial with default zero boundary conditions for velocity and acceleration.

The algorithm followed to generate the simulation of the trajectory is shown in the picture bellow.

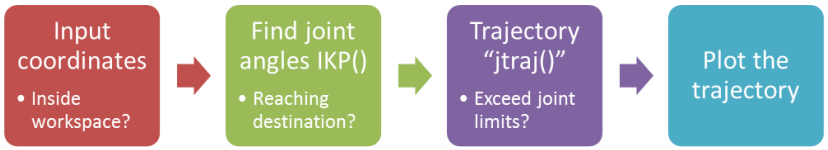


Figure 6: Trajectory Algorithm

- **Input Coordinates**: The destination coordinates is received as X, Y. It has to be in the range of 1 and 9. Then it is converted to the robot's reference (from -5 to 5) and into centimeters (where each square is $4cm^2$). e.g. if the destination coordinates entered are [5,1], in the robots reference floor they are taken as [0,-4].
- **Joint through IKP**: The joint values for the destination coordinates are obtained through the IKP function developed in the last chapter. Ideally, because of the presence of +- symmetry (e.g. elbow up and down position), 16 different solution can be obtained from the function developed. Here, it is worth mentioning that among those solution, not all can be implemented since theoretical solutions do not consider the fact that practically joint limits produce constraints in motion. Moreover, theoretical solutions also do not take in account the direction of motion of actuators. Thus, by only using the feasible solution from the resulting joint values, are introduced into the DKP function and the resulted transformation matrix is compared with the desired one.
- **Trajectory trough jtraj**: The joint trajectory between the initial position (which is pre-established) and the final position is obtained using jtraj function, which uses a quintic (5th order) polynomial with zero boundary condition for velocity and acceleration. Since there are no boundary condition defined for velocity and acceleration, controlling the speed of actuator is necessary. For this purpose, number of steps can be defined. Steps defined, divide the trajectory in numbers, starting from initial point to destination. e.g. if step=m, the trajectory will be divided into m and the matrix formed from this will be mxN while N is the DOF of manipulator. One other important aspect that must be checked is the physical

joints limitations. The matrix obtained in the mxN form is compared row wise, with a function defining joint limits so as to keep the track of joint values. If at any instant, the theoratical values exceed the joint limits, other solution from IKP must be considered and the whole process should be repeated.

- **Plot**: If the obtained trajectory is within the workspace, it is plotted.

6.3 Simulation

An example of the trajectory plotted is presented bellow. In this simulation, the manipulator went to the desired position $(1, 4)$ through the red line track. It can be seen how it accomplished the goal of reaching the desired position.

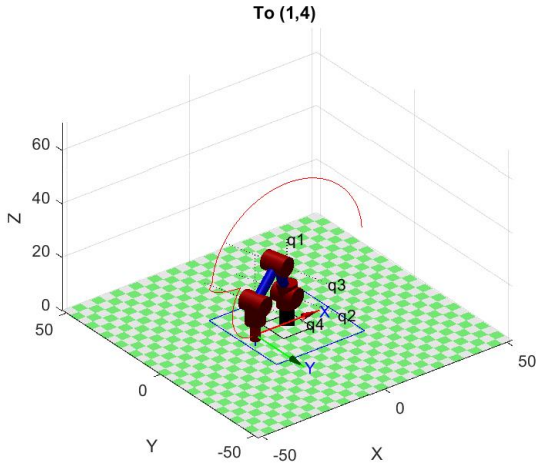


Figure 7: Trajectory Simulation

7 Connecting with the Manipulator

In this section the method and algorithm for connecting/using the manipulator will be explained in details.

7.1 DYNAMIXEL connection

Serial communication set-up between Matlab and the Dynamixel motors of the manipulator is essential for controlling the robot manipulator. Dynamixel is a line-up of high performance networked actuators for robotic applications. They are made by the Korean manufacturer Robotis. The dynamixel model with which our manipulator is build is the Dynamixel MX-28T Robot Actuator. Each motor is fully integrated with feedback functionality and programmability. In this subsection, we would see how the control of the motors is done using the Dynamixel libraries for Matlab:

- Estabilishing serial Communication with DYNAMIXEL motors:

```
1          loadlibrary('dynamixel', 'dynamixel.h');
2  %% INIZIALIZATION
3  res = calllib('dynamixel','dxl_initialize', DEFAULT_PORTNUM,
       DEFAULT_BAUDNUM);
4  if res
5      disp('Proper Inizialization');
6  else
7      error('Failed to initialize');
8  end
```

  Where:
  DEFAULT_PORTNUM - Port number by which the manipulator device is connected.
  DEFAULT_BAUDNUM - Set the value of Baud Rate with a value of 1Mbps.

- Setting the positions:
  1. Identify the current location of all the motors:

```
1  function PresentPos=readPositions()
2  %READS ALL THE VALUES OF THE MOTORS
3  id=[1 2 3 4 5 6];
4  P_PRESENT_POSITION = 36; % predefiend for read CONTROL
       TABLE
5
6  PresentPos1 = int32(calllib('dynamixel','dxl_read_word',id
       (1),P_PRESENT_POSITION));
7  PresentPos2 = int32(calllib('dynamixel','dxl_read_word',id
       (2),P_PRESENT_POSITION));
```

```
8  PresentPos3 = int32(calllib('dynamixel','dxl_read_word',id
       (3),P_PRESENT_POSITION));
9  PresentPos4 = int32(calllib('dynamixel','dxl_read_word',id
       (4),P_PRESENT_POSITION));
10 PresentPos5 = int32(calllib('dynamixel','dxl_read_word',id
       (5),P_PRESENT_POSITION));
11 PresentPos6 = int32(calllib('dynamixel','dxl_read_word',id
       (6),P_PRESENT_POSITION));
12
13 PresentPos = [PresentPos1 PresentPos2 PresentPos3
       PresentPos4 PresentPos5 PresentPos6];
14 end
```

The current location for all the motors is printed through terminal. It is important to highlight that the position read from the dynamixel are expressed on bits and a range from 0 to 4095 is available.

2. Move joints to desired positions:
   Having read the actual position of the manipulator we proceed to move the joints to a goal position. The function given next realizes this task by providing manually the ID of the manipulator and a goal position in bytes.

```
1                  function MoveTo(HomePos)
2  %GIVES THE VALUES OF THE MOTOR STORED IN HomePos (Goal
       position) TO THE REAL ROBOT
3  id=[1 2 3 4 5 6];
4  P_GOAL_POSITION = 30; % predifined for set CONTROL TABLE
5
6  for i=1: size(HomePos,1)-1
7            calllib('dynamixel','dxl_write_word',id(1),
                 P_GOAL_POSITION,HomePos(i,1));
8            calllib('dynamixel','dxl_write_word',id(2),
                 P_GOAL_POSITION,HomePos(i,2));
9            calllib('dynamixel','dxl_write_word',id(3),
                 P_GOAL_POSITION,HomePos(i,3));
10           calllib('dynamixel','dxl_write_word',id(4),
                 P_GOAL_POSITION,HomePos(i,4));
11           calllib('dynamixel','dxl_write_word',id(5),
                 P_GOAL_POSITION,HomePos(i,5));
12     end
13
14 end
```

7.2 Pick and Placement of a given object

In order to program the manipulator for the final pick and place task, the following considerations needed to be met:

- The manipulator must be set in a suitable initial position (Home Position) to avoid collision with the object and avoid singularity configurations
- The given object is a totem of height around 6.5cm.

The algorithm followed in our study is discussed below:

A) Picking of Totem from a given location:

1. Initialization of the Dynamixel libraries
2. Create robot for simulation in RTB in MATLAB
3. Read present position of the manipulator
4. If present position is not equal to home position, then move the manipulator to Home Position using the sub-routine given in step 5, else move to step 6.
5. Go-To home position sub-routine:
   - Define the trajectory for movement from current to Home Position
   - Check motor direction of movement
   - Check the mechanical joint limits are not exceeded
   - Define intermediate steps for slowing the operational speed of motors
   - Move to home position
   - Close end effecter gripper
6. Once home position is reached, next move to goal position using the same sub-routine as in step 5, for final destination location
7. Open end effecter Gripper and grab totem
8. Move back to Home position

B) Release of Totem at Goal position:

1. Follow the sub-routine in step 5 in part (a) to move from Home position to final goal location.
2. Open end effecter gripper and release totem.
3. End of process

It is worthwhile to observe that a step value is introduced in order control the speed of the manipulator and have a smooth the movement from an initial position to the goal position.

A picture for the flowchart to pick an object is presented bellow.

7.3 MATLAB Code

The fully functional code for this final task involves all the sub-parts introduced earlier in the work. The full code maybe found in the appendix included at the end of the document.

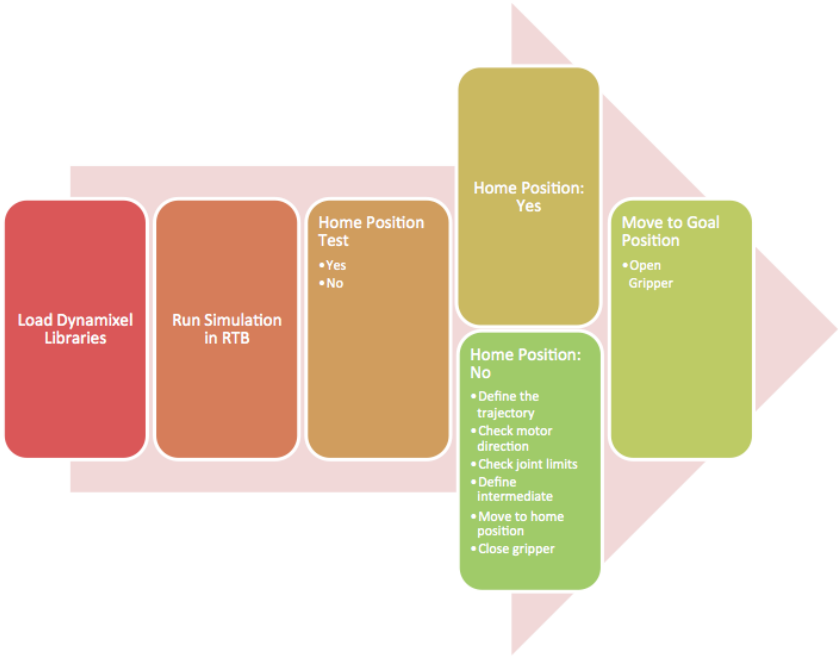


Figure 8: Flowchart for PICK action

8 Conclusion

Programming of the robot manipulator for the assigned pick and place task involved several intermediary stages. The first and most fundamental step in the process was defining the kinematic structure of the manipulator, which impacts the overall result and performance of the system. The manipulator model was described by obtaining the link transformation matrices from each joint using the Denavit-Hartenberg (DH) notations. Then, the Inverse Kinematics (IIK) was formulated to find configurations for the manipulator. Frame assignment and selection of the DH parameters directly influence both the forward and inverse kinematics, and thus this step was a crucial one in our task.

The Matlab robotic toolbox (RTB) was used for simulations and visualization purposes and was very useful to observe and identify favorable configurations and paths for the trajectory planning for the robot manipulator. It helped provide a better understanding of the system behaviour and analysis of the system constrains. To find admissible solutions along a path, the workspace and physical limitations of the manipulator were also considered. It was also important to recognize and avoid possible singularities present while computing the angles for a desired position. The Dynamixel MX-28T Robot Actuator presented comfortable handling and control of the manipulator since it has an easy interface with Matlab.

In the method adopted in our work, the manipulator was always started from a suitable home position that avoided any singularity possibilities. Using the inverse kinematics calculations, the current spatial location of the totem was converted to required angles for the manipulator. The dynamixel connection is then used to set the manipulator in the right orientation and the end effector proceeds to grab the totem. The manipulator then traverses back via a suitable trajectory first to the home position and then to release the totem in its final desired location.

The effectiveness of our analysis was verified by comparing simulation and experimental results. Overall results confirm the efficiency of the implemented methods for performing pick-and-place operations with a 6-DOF manipulator.

A MATLAB Code

A.1 Main Function

```
% CONTROL TABLE:
% http://support.robotis.com/en/product/actuator/dynamixel/mx_series/mx-28at_ar.htm
% .../mx-64at_ar.htm

id=[1 2 3 4 5 6];
P_PRESENT_POSITION = 36; % predefiend for read CONTROL TABLE
P_GOAL_POSITION = 30; % predifined for set CONTROL TABLE
DEFAULT_PORTNUM = 6;  %COM3
DEFAULT_BAUDNUM = 1;  %1Mbps

myStep= 8; % number of steps controls the velocity of actuator
Display= false;
toPlot=false;
left=1;
loadlibrary('dynamixel', 'dynamixel.h'); % load dynamixel library
%% INIZIALIZATION of dynamixel
res = calllib('dynamixel','dxl_initialize',DEFAULT_PORTNUM, DEFAULT_BAUDNUM);
if res
    disp('Proper Inizialization');
else
    error('Failed to initialize');
end
%% Creating Bot
 createBot();
%% FIND TRAJ TO HOME
 PresentPos = readPositions()
 qtemp= [0 1.0472 -0.5236 0.5236 0]; %Home Position, chosen by finding suitable angles offline
 HomePos = fromTraj2Man(qtemp,PresentPos, myStep); % convert simulation to manioulator using fromTraj2Man function
 checkMotLim(HomePos);
 checkMotStep(HomePos, myStep);
%% GO HOME
 MoveTo(HomePos);

 %% FIND TRAJ TO SOMEPLACE A
 actionGripper('Open');
 PresentPos = readPositions()
 MapCoord = [2 2]; %X and Y to pick
 qtemp = oneTraj2(MapCoord(1),MapCoord(2), toPlot);
 HomePos = fromTraj2Man(qtemp,PresentPos, myStep);
 checkMotLim(HomePos); %check if actuators are within limits
 checkMotStep(HomePos, myStep); % Check between two consective steps so as to move the actuators slowly
 %% GO TO THAT PLACE & PICK
 MoveTo(HomePos);
%%
 actionGripper('close'); %send action close to gripper
```

```
46 %% FIND TRAJ TO HOME
47  PresentPos = readPositions()
48
49  if (MapCoord(1) <3 && MapCoord(1)>0 && MapCoord(2) <8 && MapCoord
        (2)>2) %% Have to go to LEFT
50      left=-1;
51  else
52      left=1;
53  end
54  Ahmadtraj=PresentPos(2):left*myStep:PresentPos(2)+ left*300; %lift
        the manipulator by joint 3 so that it doesnt crash with the
       floor after picking operation
55  for i=1:size(Ahmadtraj,2)
56      calllib('dynamixel','dxl_write_word',id(2),P_GOAL_POSITION,
            Ahmadtraj(i));
57  end
58  PresentPos = readPositions()
59
60  qtemp= [0 1.0472 -0.5236 0.5236 0]; %HOME POSITION
61  HomePos = fromTraj2Man(qtemp,PresentPos, myStep); %home position
       in manipulator
62  % check actuator limits and step
63  checkMotLim(HomePos);
64  checkMotStep(HomePos, myStep);
65
66  %% GO TO HOME
67  MoveTo(HomePos);
68  %% TRAJ TO DROP
69  PresentPos = readPositions()
70  MapCoord = [9 9]; %X and Y to go as destination
71  qtemp = oneTraj2(MapCoord(1),MapCoord(2), toPlot);
72  HomePos = fromTraj2Man(qtemp,PresentPos, myStep);
73  checkMotLim(HomePos);
74  checkMotStep(HomePos, myStep);
75  %% MOVE to DROP
76   MoveTo(HomePos);
77
78  %% GO TO THAT PLACE & PICK
79  actionGripper('Open');
80
81  %%
82   PresentPos = readPositions()
83   %%
84   left=1;
85  Ahmadtraj=PresentPos(2):left*myStep:PresentPos(2)+ left*300;
86  for i=1:size(Ahmadtraj,2)
87      calllib('dynamixel','dxl_write_word',id(2),P_GOAL_POSITION,
            Ahmadtraj(i));
88  end
89  PresentPos = readPositions()
90
91  qtemp= [0 1.0472 -0.5236 0.5236 0]; %HOME POSITION
92  HomePos = fromTraj2Man(qtemp,PresentPos, myStep);
93  checkMotLim(HomePos);
94  checkMotStep(HomePos, myStep);
95
96  %%
```

```
97      MoveTo(HomePos);
98
99  %calllib('dynamixel','dxl_terminate');
```

A.2 Robot Creation in Simulation

```
1                   function createBot()
2   %CREATES BOT WITH OUR PARAMS
3   run('C:\Users\Usuario\Documents\MATLAB\robot-9.10\rvctools\
        startup_rvc.m')
4   % clc;clearvars;close all;
5
6   %% Vars
7
8   % JOINT LIMITS
9   JLim1 =[-5.3198/2, 5.3198/2 ];                    % First Joint
        Limits
10  JLim2 =[-3.8319/2 + pi/2,  3.8319/2 + pi/2];      % Second Joint
        Limits
11  JLim3 =[-3.9761/2, 3.9761/2];                     % Third Joint
        Limits
12  JLim4 =[-3.9807/2 + pi/2, 3.9807/2 + pi/2];       % Forth Joint Limis
13  JLim5 =[-5.2477/2, 5.2477/2];                     % Fifth Joint
        Limits
14  JLim= [JLim1;JLim2;JLim3;JLim4;JLim5]; % Joint Limitation Matrix
15
16  % MANIPULATOR & ENVIRONMENT STRUCTURE
17  a2=14.7;
18  a3=14.9;
19  d5=3;
20  baseSize=[0 0 11.8 1]';
21  toolSize=[0 0 8.9 1]';
22  workspace=[-50,50,-50,50,0,70]; %W = [xmn, xmx ymn ymx zmn zmx]
        defining floor in the simulation environment
23  tokenHeight=6;
24
25  % DH PARAMS
26  L1 = Link('d', 0, 'a', 0, 'alpha', 0, 'modified', 'qlim', JLim1);
27  L2 = Link('d', 0, 'a', 0, 'alpha', pi/2, 'modified', 'qlim', JLim2)
        ;
28  L3 = Link('d', 0, 'a', a2, 'alpha', 0, 'modified', 'qlim', JLim3 );
29  L4 = Link('d', 0, 'a', a3, 'alpha', 0, 'modified', 'qlim', JLim4);
30  L5 = Link('d', d5, 'a', 0, 'alpha', pi/2, 'modified', 'qlim', JLim5
        );
31
32  % BUILD ROBOT
33  bot = SerialLink([L1 L2 L3 L4 L5]);                % Join Links
34  bot.base(:,4)=baseSize;                            % Setting Base
35  bot.tool(:,4)=toolSize;                            % Setting End
        Effector
36  mask = [1 1 1 1 1 0];
37  disp('Bot created');
38  end
```

A.3 Simulation

```
1          function QTi=oneTraj2(i,j,boolean)
2
3  % MANIPULATOR & ENVIRONMENT STRUCTURE
4  %% Vars
5
6  % JOINT LIMITS
7  JLim1 =[-5.3198/2, 5.3198/2 ];                    % First Joint
       Limits
8  JLim2 =[-3.8319/2 + pi/2,  3.8319/2 + pi/2];     % Second Joint
       Limits
9  JLim3 =[-3.9761/2, 3.9761/2];                     % Third Joint
       Limits
10 JLim4 =[-3.9807/2 + pi/2, 3.9807/2 + pi/2];      % Forth Joint Limis
11 JLim5 =[-5.2477/2, 5.2477/2];                     % Fifth Joint
       Limits
12 JLim= [JLim1;JLim2;JLim3;JLim4;JLim5]; % Joint Limitation Matrix
13
14 % MANIPULATOR & ENVIRONMENT STRUCTURE
15 a2=14.7;
16 a3=14.9;
17 d5=3;
18 baseSize=[0 0 11.8 1]'; % defining base size as it is not included
       in DH paramters
19 toolSize=[0 0 8.9 1]'; % defining tool size as it is not included
       in DH paramters
20 workspace=[-50,50,-50,50,0,70]; %W = [xmn, xmx ymn ymx zmn zmx]
21 tokenHeight=6.5;
22
23 % DH PARAMS
24 L1 = Link('d', 0, 'a', 0, 'alpha', 0, 'modified', 'qlim', JLim1); %
       Defining link 1 as per Robotics toolbox requirements
25 L2 = Link('d', 0, 'a', 0, 'alpha', pi/2, 'modified', 'qlim', JLim2)
       ; % Defining link 2 as per Robotics toolbox requirements
26 L3 = Link('d', 0, 'a', a2, 'alpha', 0, 'modified', 'qlim', JLim3 );
       % Defining link 3 as per Robotics toolbox requirements
27 L4 = Link('d', 0, 'a', a3, 'alpha', 0, 'modified', 'qlim', JLim4);
      % Defining link 4 as per Robotics toolbox requirements
28 L5 = Link('d', d5, 'a', 0, 'alpha', pi/2, 'modified', 'qlim', JLim5
       ); % Defining link 5 as per Robotics toolbox requirements
29
30 % BUILD ROBOT
31 bot = SerialLink([L1 L2 L3 L4 L5]);                % Join Links
32 bot.base(:,4)=baseSize;                            % Setting Base
33 bot.tool(:,4)=toolSize;                            % Setting End
       Effector
34
35          MapCoord = [i j];              % Coord in the map floor
36          Dest = MapCoord - 5;           % Destination: [X , Y]
                coordinate
37          Pf=[Dest(1), Dest(2), tokenHeight/4]';     % Setting the P.
                Pz= (Token High / 2)/4 -> for future cm
38          Rf=[1   0   0;             % Rotation matrix to end the
                trajectory with the end-effector pointing down
39              0   -1   0;
40              0    0  -1];
```

```
41
42          Tf = [Rf Pf*4; 0 0 0 1];        % Building the Hom. Trasnf.
                Matrix... Mult. by 4 cuz converting
43                                          %   from coordinate to cm's
44          %vars
45          qf=[];
46          qt=[];
47          time=1:1:51;
48
49          qi=[0 1.0472 -0.5236 0.5236 0];                  % Initial
                position
50          ql=[0 -1.0472+pi (-1)*(-0.5236) -0.5236+pi 0];  % Left side
                checkpoint
51          % Check if LEFT
52          if (MapCoord(1) <3 && MapCoord(1)>0 && MapCoord(2) <8 &&
                MapCoord(2)>2) %% Have to go to LEFT
53              Tf(3,4)=5.2;
54              qf=myInv2L(Tf,JLim);                         %% IKP for
                    qf between checkp and final
55              Tqf = DKP(qf);                               %% DKP to
                    checl final position
56              qc=qf;
57              qc(4)=0;
58              qt=[jtraj(qi,ql,time); jtraj(ql,qc,time); jtraj(qc,qf,
                    time)];  %% Define traject for this part
59          else
60          qf=myInv2(Tf);     %Inverse from transformation using
                defined inverse function
61          qf(5)=0; %CHECK that should not change angle
62          Tqf=bot.fkine(qf); % FKP using RTB
63          qc=qf;
64          qc(4)=pi;
65          qt=[jtraj(qi,qc,time);jtraj(qc,qf,time)];
66          end
67
68          mytitle = (['To (' num2str(MapCoord(1)) ',' num2str(
                MapCoord(2)) ')']);
69          if CheckJointLimits3(qt,size(qt,1),JLim) == 1 || CheckTPos(
                Tqf,Tf)         % Check if ikine func gave good angles
70              error(['Error on ' mytitle]);
71  %             continue;
72          end
73
74          QTi=qt;
75          if(boolean)
76              FerPlot2(bot,qt,3);
77              bot.fkine(bot.getpos())
78          else
79              disp('No simulation plot');
80          end
81
82          disp('Action');
83
84  end
```

A.4 Reading Current Positions of Actuator

```
                function PresentPos=readPositions()
%READS ALL THE VALUES OF THE MOTORS
id=[1 2 3 4 5 6];
P_PRESENT_POSITION = 36; % predefiend for read CONTROL TABLE

PresentPos1 = int32(calllib('dynamixel','dxl_read_word',id(1),
    P_PRESENT_POSITION)); % Read position of actuator 1
PresentPos2 = int32(calllib('dynamixel','dxl_read_word',id(2),
    P_PRESENT_POSITION)); % Read position of actuator 2
PresentPos3 = int32(calllib('dynamixel','dxl_read_word',id(3),
    P_PRESENT_POSITION)); % Read position of actuator 3
PresentPos4 = int32(calllib('dynamixel','dxl_read_word',id(4),
    P_PRESENT_POSITION)); % Read position of actuator 4
PresentPos5 = int32(calllib('dynamixel','dxl_read_word',id(5),
    P_PRESENT_POSITION)); % Read position of actuator 5
PresentPos6 = int32(calllib('dynamixel','dxl_read_word',id(6),
    P_PRESENT_POSITION)); % Read position of actuator 6 (gripper)

PresentPos = [PresentPos1 PresentPos2 PresentPos3 PresentPos4
    PresentPos5 PresentPos6]; % Return present position


end
```

A.5 Trajectory to Manipulator Conversion:

```
                function HomePos=fromTraj2Man(q,PresentPos, mystep)
%% CREATES A MATRIX OF VALUES OF THE MOTORS VALUES.. BY TAKING THE
    TRAJECTORY ANGLES FROM SIMULATION AND THEN SMOOTHING
%% THE TRANSITION BY THE STEPSIZE
initialqs=int32(PresentPos);
GoalPos1=PresentPos(1);
GoalPos2=PresentPos(2);
GoalPos3=PresentPos(3);
GoalPos4=PresentPos(4);
GoalPos5=PresentPos(5);
qtraj=int32(q*4096/(2*pi));     %Integers

for i= 1: 1: size(qtraj,1)
    % conversion from simulation (matlab) joints to manipulator
        joints
    qtraj(i,1)=qtraj(i,1)+4096/2;
    qtraj(i,2)=qtraj(i,2)+4096/4;
    qtraj(i,3)=4096/2 - qtraj(i,3);
    qtraj(i,4)=qtraj(i,4)+4096/4;
    qtraj(i,5)=qtraj(i,5)+4096/2;

    %all this is done to check that the motors are going in the
        right
    %direction.. using a correct step (positive or negative if
        going up or
    %down)
    if ( int32(qtraj(i,1))-initialqs(1)==0)
```

```
24          temp1=initialqs(1);
25      else
26          temp1= initialqs(1): sign(int32(qtraj(i,1))-initialqs(1))*
                mystep :int32(qtraj(i,1));
27          if (temp1==initialqs(1)) temp1=[];  end
28      end
29      GoalPos1 = [GoalPos1 temp1];
30
31      if ( int32(qtraj(i,2))-initialqs(2)==0)
32          temp2=initialqs(2);
33      else
34      temp2= initialqs(2): sign(int32(qtraj(i,2))-initialqs(2))*
            mystep :int32(qtraj(i,2));
35      if (temp2==initialqs(2)) temp2=[];  end
36      end
37      GoalPos2 = [GoalPos2 temp2];
38
39      if ( int32(qtraj(i,3))-initialqs(3)==0)
40          temp3=initialqs(3);
41      else
42      temp3= initialqs(3): sign(int32(qtraj(i,3))-initialqs(3))*
            mystep :int32(qtraj(i,3));
43      if (temp3==initialqs(3)) temp3=[];  end
44      end
45      GoalPos3 = [GoalPos3 temp3];
46
47
48      if ( int32(qtraj(i,4))-initialqs(4)==0)
49          temp4=initialqs(4);
50      else
51      temp4= initialqs(4): sign(int32(qtraj(i,4))-initialqs(4))*(
            mystep*2) :int32(qtraj(i,4));
52      if (temp4==initialqs(4)) temp4=[];  end
53      end
54      GoalPos4 = [GoalPos4 temp4];
55
56       if ( int32(qtraj(i,5))-initialqs(5)==0)
57          temp5=initialqs(5);
58      else
59      temp5= initialqs(5): sign(int32(qtraj(i,5))-initialqs(5))*
            mystep :int32(qtraj(i,5));
60      if (temp5==initialqs(5)) temp5=[];  end
61       end
62      GoalPos5 = [GoalPos5 temp5];
63
64      initialqs = [GoalPos1(end) GoalPos2(end) GoalPos3(end) GoalPos4
            (end) GoalPos5(end)];
65  end
66
67  biggestMat = max([size(GoalPos1,2),size(GoalPos2,2),size(GoalPos3
        ,2), size(GoalPos4,2), size(GoalPos5,2)]);
68  HomePos=ones(biggestMat,5);
69  %%
70  HomePos(:,1)=GoalPos1(end);
71  HomePos(:,2)=GoalPos2(end);
72  HomePos(:,3)=GoalPos3(end);
73  HomePos(:,4)=GoalPos4(end);
```

```
74 HomePos(:,5)=GoalPos5(end);
75 %%
76 HomePos(1:size(GoalPos1,2),1)=GoalPos1';
77 HomePos(1:size(GoalPos2,2),2)=GoalPos2';
78 HomePos(1:size(GoalPos3,2),3)=GoalPos3';
79 HomePos(1:size(GoalPos4,2),4)=GoalPos4';
80 HomePos(1:size(GoalPos5,2),5)=GoalPos5';
81 end
```

A.6 Actuator Limits

```
1                 function checkMotLim(HomePos)
2 %CHECHKING IF THE TRAJECTORY WE ARE GOING INSIDE THE BOUNDERIES OF
      THE MOTOR VALUES
3 %MEASURED BEFORE
4 for i= 1: size(HomePos,1)
5     if     ( HomePos(i,1) < 299    || HomePos(i,1)> 3767) % check if
          the actuator physical lower and upper limits exceed (
          calculated in offline mode)
6         error(['Exceeding Joint Limit: ' int2str(1) ' at row '
              int2str(i)]);
7     elseif( HomePos(i,2) < 825    || HomePos(i,2)> 3323)
8         error(['Exceeding Joint Limit:' int2str(2) ' at row '
              int2str(i)]);
9     elseif( HomePos(i,3) < 737    || HomePos(i,3)> 3329)
10        error(['Exceeding Joint Limit:' int2str(3) ' at row '
              int2str(i)]);
11    elseif( HomePos(i,4) < 748    || HomePos(i,4)> 3343)
12        error(['Exceeding Joint Limit:' int2str(4) ' at row '
              int2str(i)]);
13    elseif( HomePos(i,5) < 341    || HomePos(i,5)> 3762)
14        error(['Exceeding Joint Limit:' int2str(5) ' at row '
              int2str(i)]);
15    end
16 end
17 disp('Joint Limits for Motors: OK');
18 end
```

A.7 Step Size

```
1                 function checkMotStep(HomePos, myStep)
2 %CHECKS THE STEP SIZE OF 2 CONSECUTIVE VALUES IN THE TRAJECTORY..
      SO IT
3 %DOESNT MAKE FAST/BIG TRANSITION THAT COULD BREAK THE ROBoT
4 for i= 1: size(HomePos,1)-1
5     if      ( abs(HomePos(i,1) - HomePos(i+1,1)) > myStep)
6         error(['Exceeding Joint Step' int2str(1) ' at row ' int2str(
              i) ',' int2str(1+1)]);
7     elseif ( abs(HomePos(i,2) - HomePos(i+1,2)) > myStep)
8         error(['Exceeding Joint Step' int2str(2) ' at row ' int2str(
              i) ',' int2str(i+1)]);
9     elseif ( abs(HomePos(i,3) - HomePos(i+1,3)) > myStep)
10        error(['Exceeding Joint Step' int2str(3) ' at row ' int2str(
              i) ',' int2str(i+1)]);
```

```
11      elseif ( abs(HomePos(i,4) - HomePos(i+1,4)) > 2*myStep) %THIS
            IS INTENDED
12          error(['Exceeding Joint Step' int2str(4) ' at row ' int2str(
                i) ',' int2str(i+1)]);
13      elseif ( abs(HomePos(i,5) - HomePos(i+1,5)) > myStep)
14          error(['Exceeding Joint Step' int2str(5) ' at row ' int2str(
                i) ',' int2str(i+1)]);
15
16      end
17  end
18  disp('Joint Step change for Motors: OK');
19  end
```

A.8 Task Execution Call

```
1                   %% function to execute the actuators to new
                        position
2   function MoveTo(HomePos)
3   %GIVES THE VALUES OF THE MOTOR STORED IN HomePos TO THE REAL ROBOT
4   id=[1 2 3 4 5 6];
5   P_GOAL_POSITION = 30; % predifined for set CONTROL TABLE
6
7   for i=1: size(HomePos,1)-1
8           calllib('dynamixel','dxl_write_word',id(1),P_GOAL_POSITION
                ,HomePos(i,1)); % move actuator 1 from home position
                to goal
9           calllib('dynamixel','dxl_write_word',id(2),P_GOAL_POSITION
                ,HomePos(i,2)); % move actuator 2 from home position
                to goal
10          calllib('dynamixel','dxl_write_word',id(3),P_GOAL_POSITION
                ,HomePos(i,3)); % move actuator 3 from home position
                to goal
11          calllib('dynamixel','dxl_write_word',id(4),P_GOAL_POSITION
                ,HomePos(i,4)); % move actuator 4 from home position
                to goal
12          calllib('dynamixel','dxl_write_word',id(5),P_GOAL_POSITION
                ,HomePos(i,5)); % move actuator 5 from home position
                to goal
13      end
14
15  end
```

A.9 Gripper Action

```
1
2                   function actionGripper(action)
3       id=[1 2 3 4 5 6];
4       P_PRESENT_POSITION = 36; % predefiend for read CONTROL TABLE
5       P_GOAL_POSITION = 30; % predifined for set CONTROL TABLE
6       myStep=5;
7       action = lower(action);
8       if      (strcmp( action, 'open')|| strcmp( action,'drop'))
9           value=512;  %Value to drop or open the dripper
```

```
10      elseif (strcmp( action, 'close')|| strcmp( action,'pick'))
11          value=0;  %Value to drop or open the dripper
12      else
13          error(['Undefined action: ' action]);
14      end
15
16      % Read position
17      PresentPos6 = int32(calllib('dynamixel','dxl_read_word',id(6),
            P_PRESENT_POSITION));
18      GoalPos6= PresentPos6: sign(value-PresentPos6)*myStep :value; %
            set goal position
19      if(isempty(GoalPos6))
20          GoalPos6=PresentPos6;
21      end;
22      GoalPos6=GoalPos6';
23      for i=1: size(GoalPos6,1)
24          % Send to Position
25           calllib('dynamixel','dxl_write_word',id(6),P_GOAL_POSITION
                ,GoalPos6(i));
26      end
27
28  end
```