Project Report

ENPM662: Introduction to Robot Modeling
Fall 2021
University of Maryland

Using Baxter for Farming in International Space Station

September 6, 2022

Neha Saini
UID:117556292

Abstract

Following is report for the final project of the course Introduction to Robot Modelling. The proposed project describes the use of robot Baxter to achieve a set of farming tasks at the International Space Station. The forward and inverse kinematics analysis on the left arm of Baxter followed by the formulation of dynamic analysis and there simulation in Gazebo are described.

1 Introduction

The Baxter robot was developed by Rethink Robotics in 2012. Later a newer version Baxter Intera 3 came out specifically for research purposes and is used today for the same in many laboratories. In one of the articles [3] Baxter is described as; a pair of chunky robot arms connected to a computer brain rides a motorised wheelchair over to a jar of peanut butter. The right gripper holds the jar still, while the left unscrews the top. The red arms reach out and grab it. The arms belong to Baxter, an industrial robot with a twist – it is designed to work right next to humans on factory floors. Figure 1, shows the picture of the Baxter Robot at Maryland Robotics Center, University of Maryland.

Figure 1: The Baxter Robot at Maryland Robotics Center.

2 Application

Due to the human like features Baxter is able to mimic many of the tasks which humans can perform. One such application which motivated me was to use Baxter for farming. The reasons for choosing this application are as follows:

- Baxter provides an opportunity to bridge Earth farming systems to space.
- The astronauts who specifically go for deep space missions for longer duration invest a lot of time in farming due to which they are not able to focus on other significantly important tasks. So in order for a better time management we can employ the robots as space farmers.
- Also in order to send food supplies on a regular basis to the International Space Station, a lot of fuel and time is required with high risk of failure which in turn leads to loss of resources. So creating a habitat in outer space is very beneficial.
- The use of Baxter as a farming robots will not only help in space farming but will also help with farming on earth in the conditions where it is difficult for humans to operate.

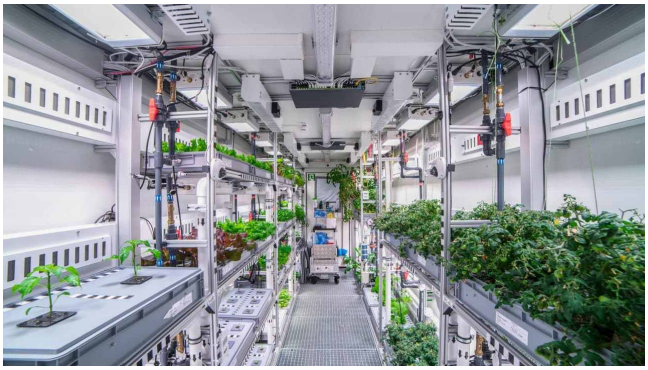

Figure 2: On going farming on ISS.

In my project I tried to use the left arm of the Baxter for executing a simple task of pouring water to the plants. Figure 2 shows an example of on going farming at ISS.

3 Robot Type

Baxter is a serial manipulator on a mobile platform with seven Degrees of Freedom revolute joints on each of its two arms, a two Degrees of Freedom head, and a torso with a mobile platform. The arms are actuated by Series Elastic Actuators which incorporate torsional springs and thus allow some compliance. This

feature along with various software features render the Baxter Robot, human-friendly and hence extremely popular for research activities. The servo motors also allow for inbuilt force, torque and position sensing at each joint.

Both 7-dof arms include angle position and joint torque sensing. For Cartesian sensors, there are three integrated cameras, plus sonar, accelerometers and range-finding sensors. Each Baxter arm has a temperature sensor, allowing human fingers to be detected for lead-through programming and other applications. The research version allows programming via a standard, opensource ROS API interface.

4 Degrees of Freedom and Dimensions

Baxter is about 3' tall (around 6' tall with stationary pedestal) and weighs 165 lbs. (306 lbs. including the pedestal). Baxter has a 103" 'wingspan' and a 32" x 36" pedestal base.

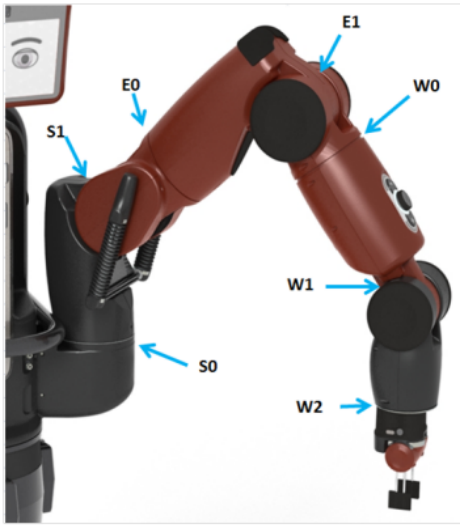

Figure 3: The left arm of Baxter.

The Figure 3 shows the left arm of the Baxter Robot. Each 7-dof arm has a 2-dof (offset-U-joint) shoulder joint, a 2-dof (offset-U-joint) elbow joint, and a 3-dof (offset-S-joint) wrist joint. The Baxter 7-dof arms neither have a spherical shoulder nor a spherical wrist so it has 3 offsets due to which no 3 consecutive frames meet at the same origin. Now according to Pieper's principle, if a 6-dof serial robot has 3 consecutive coordinate frames meeting at the same origin, then an analytical solution is guaranteed to exist for the coupled nonlinear inverse pose kinematics problem. But this does not occur for Baxter, so I suspect no analytical solutions exist for Inverse Pose Kinematics.[4][2]

5 CAD Model

The image of the CAD model designed in Solidworks for the left arm is provided in Figure 4.

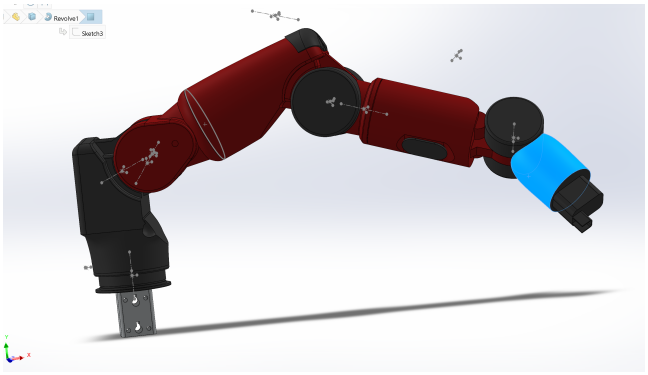

Figure 4: The CAD model for left arm of Baxter.

6 Denavit–Hartenberg Parameters

The Cartesian coordinate frame assignment diagram for the seven degree of freedom left arm in its home position is shown in Figure 5.

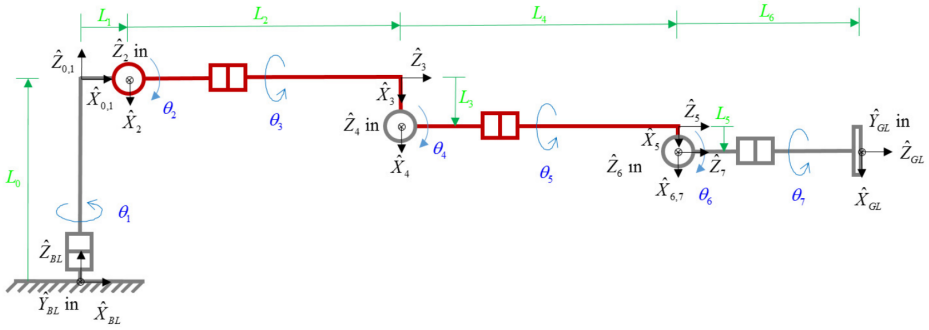

Figure 5: Seven-dof Left Arm Kinematic Diagram with Coordinate Frames.

The DH parameters based on the link assignment using Craig's Convention [1] is shown in Table 1.

Based on the DH parameters we will calculate the kinematic analysis.

Table 1: The DH Parameter of Baxter Left arm.

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	-90	L_1	0	$\theta_2 + 90$
3	90	0	L_2	θ_3
4	-90	L_3	0	θ_4
5	90	0	L_4	θ_5
6	-90	L_5	0	θ_6
7	90	0	0	θ_7

7 Forward Kinematics

To obtain the seven neighboring homogeneous transformation matrices as a function of the joint angles(theta) for the 7-dof left arm, we substitute each row of the DH parameters to equations of homogeneous transformation matrix.

Now we substitute these seven neighboring homogeneous transformation matrices into the homogeneous transform equation 1 to derive the active-joints FPK result:

$$ {}^{0}_{7}T = {}^{0}_{1}T * {}^{1}_{2}T * {}^{2}_{3}T * {}^{3}_{4}T * {}^{4}_{5}T * {}^{5}_{6}T * {}^{6}_{7}T \tag{1} $$

Hence we get ${}^{0}_{7}T$ as:

$$ {}^{0}_{7}T = \begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ R_{21} & R_{22} & R_{23} & R_{24} \\ R_{31} & R_{32} & R_{33} & R_{34} \\ R_{41} & R_{42} & R_{43} & R_{44} \end{bmatrix} $$

where

$$ \begin{aligned} R_{11} = {} & ((((-s(1)*s(3) - s(2)*c(1)*c(3))*c(4) - s(4)*c(1)*c(2))*c(5) + \\ & (-s(1)*c(3) + s(2)*s(3)*c(1))*s(5))*c(6) - ((-s(1)*s(3) - s(2)*c(1)* \\ & c(3))*s(4) + c(1)*c(2)*c(4))*s(6))*c(7) + (-((-s(1)*s(3) - s(2)*c(1)* \\ & c(3))*c(4) - s(4)*c(1)*c(2))*s(5) + (-s(1)*c(3) + s(2)*s(3)*c(1))*c(5))*s(7) \end{aligned} $$

$$ \begin{aligned} R_{12} = {} & -((((-s(1)*s(3) - s(2)*c(1)*c(3))*c(4) - s(4)*c(1)*c(2))*c(5) + \\ & (-s(1)*c(3) + s(2)*s(3)*c(1))*s(5))*c(6) - ((-s(1)*s(3) - s(2)*c(1)* \\ & c(3))*s(4) + c(1)*c(2)*c(4))*s(6))*s(7) + (-((-s(1)*s(3) - s(2)*c(1)* \\ & c(3))*c(4) - s(4)*c(1)*c(2))*s(5) + (-s(1)*c(3) + s(2)*s(3)*c(1))*c(5))*c(7) \end{aligned} $$

$$ \begin{aligned} R_{13} = {} & (((-s(1)*s(3) - s(2)*c(1)*c(3))*c(4) - s(4)*c(1)*c(2))*c(5) + \\ & (-s(1)*c(3) + s(2)*s(3)*c(1))*s(5))*s(6) + ((-s(1)*s(3) - s(2)*c(1)* \\ & c(3))*s(4) + c(1)*c(2)*c(4))*c(6) \end{aligned} $$

$$ \begin{aligned} R_{14} = {} & L1*c(1) + L2*c(1)*c(2) + L3*(-s(1)*s(3) - s(2)*c(1)*c(3)) - L4* \\ & (-(-s(1)*s(3) - s(2)*c(1)*c(3))*s(4) - c(1)*c(2)*c(4)) + L5*(((-s(1)*s(3) - \end{aligned} $$

$s(2)*c(1)*c(3))*c(4)-s(4)*c(1)*c(2))*c(5)+(-s(1)*c(3)+s(2)*s(3)*c(1))*s(5))$

$R_{21} = ((((-s(1)*s(2)*c(3)+s(3)*c(1))*c(4)-s(1)*s(4)*c(2))*c(5)+(s(1)*s(2)*s(3)+c(1)*c(3))*s(5))*c(6)-((-s(1)*s(2)*c(3)+s(3)*c(1))*s(4)+s(1)*c(2)*c(4))*s(6))*c(7)+(-((-s(1)*s(2)*c(3)+s(3)*c(1))*c(4)-s(1)*s(4)*c(2))*s(5)+(s(1)*s(2)*s(3)+c(1)*c(3))*c(5))*s(7)$

$R_{22} = -((((-s(1)*s(2)*c(3)+s(3)*c(1))*c(4)-s(1)*s(4)*c(2))*c(5)+(s(1)*s(2)*s(3)+c(1)*c(3))*s(5))*c(6)-((-s(1)*s(2)*c(3)+s(3)*c(1))*s(4)+s(1)*c(2)*c(4))*s(6))*s(7)+(-((-s(1)*s(2)*c(3)+s(3)*c(1))*c(4)-s(1)*s(4)*c(2))*s(5)+(s(1)*s(2)*s(3)+c(1)*c(3))*c(5))*c(7)$

$R_{23} = (((-s(1)*s(2)*c(3)+s(3)*c(1))*c(4)-s(1)*s(4)*c(2))*c(5)+(s(1)*s(2)*s(3)+c(1)*c(3))*s(5))*s(6)+((-s(1)*s(2)*c(3)+s(3)*c(1))*s(4)+s(1)*c(2)*c(4))*c(6)$

$R_{24} = L1*s(1)+L2*s(1)*c(2)+L3*(-s(1)*s(2)*c(3)+s(3)*c(1))-L4*(-(-s(1)*s(2)*c(3)+s(3)*c(1))*s(4)-s(1)*c(2)*c(4))+L5*(((-s(1)*s(2)*c(3)+s(3)*c(1))*c(4)-s(1)*s(4)*c(2))*c(5)+(s(1)*s(2)*s(3)+c(1)*c(3))*s(5))$

$R_{31} = (((s(2)*s(4)-c(2)*c(3)*c(4))*c(5)+s(3)*s(5)*c(2))*c(6)-(-s(2)*c(4)-s(4)*c(2)*c(3))*s(6))*c(7)+(-(s(2)*s(4)-c(2)*c(3)*c(4))*s(5)+s(3)*c(2)*c(5))*s(7)$

$R_{32} = -(((s(2)*s(4)-c(2)*c(3)*c(4))*c(5)+s(3)*s(5)*c(2))*c(6)-(-s(2)*c(4)-s(4)*c(2)*c(3))*s(6))*s(7)+(-(s(2)*s(4)-c(2)*c(3)*c(4))*s(5)+s(3)*c(2)*c(5))*c(7)$

$R_{33} = ((s(2)*s(4)-c(2)*c(3)*c(4))*c(5)+s(3)*s(5)*c(2))*s(6)+(-s(2)*c(4)-s(4)*c(2)*c(3))*c(6)$

$R_{34} = -L2*s(2)-L3*c(2)*c(3)-L4*(s(2)*c(4)+s(4)*c(2)*c(3))+L5*((s(2)*s(4)-c(2)*c(3)*c(4))*c(5)+s(3)*s(5)*c(2))$

$R_{41} = R_{42} = R_{43} = 0$

$R_{44} = 1$

where $s(theta) = sin(theta)$ and $c(theta) = cos(theta)$. The screenshot of the final transformation matrix in python as shown in Figure 6.

8 Inverse Kinematics

The Jacobian matrix is linear transformation used to map joint rates $\dot{\theta}$ to the Cartesian velocities $\dot{X}$. The Jacobian matrix has two components which are the translation jacobian and the rotation jacobian part. The translation part can be

```
In [77]: print(T07)

Matrix([[((((-sin(theta1)*sin(theta3) - sin(theta2)*cos(theta1)*cos(theta3))*cos(theta4) - sin(theta4)*cos(theta1)*c
os(theta2))*cos(theta5) + (-sin(theta1)*cos(theta3) + sin(theta2)*sin(theta3)*cos(theta1))*sin(theta5))*cos(theta6)
- ((-sin(theta1)*sin(theta3) - sin(theta2)*cos(theta1)*cos(theta3))*sin(theta4) + cos(theta1)*cos(theta2)*cos(theta
4))*sin(theta6))*cos(theta7) + (-((-sin(theta1)*sin(theta3) - sin(theta2)*cos(theta1)*cos(theta3))*cos(theta4) - sin
(theta4)*cos(theta1)*cos(theta2))*sin(theta5) + (-sin(theta1)*cos(theta3) + sin(theta2)*sin(theta3)*cos(theta1))*cos
(theta5))*sin(theta7), -((((-sin(theta1)*sin(theta3) - sin(theta2)*cos(theta1)*cos(theta3))*cos(theta4) - sin(theta
4)*cos(theta1)*cos(theta2))*cos(theta5) + (-sin(theta1)*cos(theta3) + sin(theta2)*sin(theta3)*cos(theta1))*sin(theta
5))*cos(theta6) - ((-sin(theta1)*sin(theta3) - sin(theta2)*cos(theta1)*cos(theta3))*sin(theta4) + cos(theta1)*cos(th
eta2)*cos(theta4))*sin(theta6))*sin(theta7) + (-((-sin(theta1)*sin(theta3) - sin(theta2)*cos(theta1)*cos(theta3))*co
s(theta4) - sin(theta4)*cos(theta1)*cos(theta2))*sin(theta5) + (-sin(theta1)*cos(theta3) + sin(theta2)*sin(theta3)*c
os(theta1))*cos(theta5))*cos(theta7), (((-sin(theta1)*sin(theta3) - sin(theta2)*cos(theta1)*cos(theta3))*cos(theta4)
- sin(theta4)*cos(theta1)*cos(theta2))*cos(theta5) + (-sin(theta1)*cos(theta3) + sin(theta2)*sin(theta3)*cos(theta
1))*sin(theta5))*sin(theta6) + ((-sin(theta1)*sin(theta3) - sin(theta2)*cos(theta1)*cos(theta3))*sin(theta4) + cos(t
heta1)*cos(theta2)*cos(theta4))*cos(theta6), L1*cos(theta1) + L2*cos(theta1)*cos(theta2) + L3*(-sin(theta1)*sin(thet
a3) - sin(theta2)*cos(theta1)*cos(theta3)) - L4*(-(-sin(theta1)*sin(theta3) - sin(theta2)*cos(theta1)*cos(theta3))*s
in(theta4) - cos(theta1)*cos(theta2)*cos(theta4)) + L5*(((-sin(theta1)*sin(theta3) - sin(theta2)*cos(theta1)*cos(the
ta3))*cos(theta4) - sin(theta4)*cos(theta1)*cos(theta2))*cos(theta5) + (-sin(theta1)*cos(theta3) + sin(theta2)*sin(t
heta3)*cos(theta1))*sin(theta5))], [(((((-sin(theta1)*sin(theta2)*cos(theta3) + sin(theta3)*cos(theta1))*cos(theta4)
- sin(theta1)*sin(theta4)*cos(theta2))*cos(theta5) + (sin(theta1)*sin(theta2)*sin(theta3) + cos(theta1)*cos(theta3))
*sin(theta5))*cos(theta6) - ((-sin(theta1)*sin(theta2)*cos(theta3) + sin(theta3)*cos(theta1))*sin(theta4) + sin(thet
a1)*cos(theta2)*cos(theta4))*sin(theta6))*cos(theta7) + (-((-sin(theta1)*sin(theta2)*cos(theta3) + sin(theta3)*cos(t
heta1))*cos(theta4) - sin(theta1)*sin(theta4)*cos(theta2))*sin(theta5) + (sin(theta1)*sin(theta2)*sin(theta3) + cos
(theta1)*cos(theta3))*cos(theta5))*sin(theta7), -(((((-sin(theta1)*sin(theta2)*cos(theta3) + sin(theta3)*cos(theta1))
*cos(theta4) - sin(theta1)*sin(theta4)*cos(theta2))*cos(theta5) + (sin(theta1)*sin(theta2)*sin(theta3) + cos(theta1)
*cos(theta3))*sin(theta5))*cos(theta6) - ((-sin(theta1)*sin(theta2)*cos(theta3) + sin(theta3)*cos(theta1))*sin(theta
4) + sin(theta1)*cos(theta2)*cos(theta4))*sin(theta6))*sin(theta7) + (-((-sin(theta1)*sin(theta2)*cos(theta3) + sin
(theta3)*cos(theta1))*cos(theta4) - sin(theta1)*sin(theta4)*cos(theta2))*sin(theta5) + (sin(theta1)*sin(theta2)*sin
(theta3) + cos(theta1)*cos(theta3))*cos(theta5))*cos(theta7), (((-sin(theta1)*sin(theta2)*cos(theta3) + sin(theta3)*
cos(theta1))*cos(theta4) - sin(theta1)*sin(theta4)*cos(theta2))*cos(theta5) + (sin(theta1)*sin(theta2)*sin(theta3) +
cos(theta1)*cos(theta3))*sin(theta5))*sin(theta6) + ((-sin(theta1)*sin(theta2)*cos(theta3) + sin(theta3)*cos(theta
1))*sin(theta4) + sin(theta1)*cos(theta2)*cos(theta4))*cos(theta6), L1*sin(theta1) + L2*sin(theta1)*cos(theta2) + L3
*(-sin(theta1)*sin(theta2)*cos(theta3) + sin(theta3)*cos(theta1)) - L4*(-(-sin(theta1)*sin(theta2)*cos(theta3) + sin
(theta3)*cos(theta1))*sin(theta4) - sin(theta1)*cos(theta2)*cos(theta4)) + L5*(((-sin(theta1)*sin(theta2)*cos(theta
3) + sin(theta3)*cos(theta1))*cos(theta4) - sin(theta1)*sin(theta4)*cos(theta2))*cos(theta5) + (sin(theta1)*sin(thet
a2)*sin(theta3) + cos(theta1)*cos(theta3))*sin(theta5))], [((((sin(theta2)*sin(theta4) - cos(theta2)*cos(theta3)*cos
(theta4))*cos(theta5) + sin(theta3)*sin(theta5)*cos(theta2))*cos(theta6) - (-sin(theta2)*cos(theta4) - sin(theta4)*c
os(theta2)*cos(theta3))*sin(theta6))*cos(theta7) + (-(sin(theta2)*sin(theta4) - cos(theta2)*cos(theta3)*cos(theta4))
*sin(theta5) + sin(theta3)*cos(theta2)*cos(theta5))*sin(theta7), -(((sin(theta2)*sin(theta4) - cos(theta2)*cos(theta
3)*cos(theta4))*cos(theta5) + sin(theta3)*sin(theta5)*cos(theta2))*cos(theta6) - (-sin(theta2)*cos(theta4) - sin(the
ta4)*cos(theta2)*cos(theta3))*sin(theta6))*sin(theta7) + (-(sin(theta2)*sin(theta4) - cos(theta2)*cos(theta3)*cos(th
eta4))*sin(theta5) + sin(theta3)*cos(theta2)*cos(theta5))*cos(theta7), ((sin(theta2)*sin(theta4) - cos(theta2)*cos(t
heta3)*cos(theta4))*cos(theta5) + sin(theta3)*sin(theta5)*cos(theta2))*sin(theta6) + (-sin(theta2)*cos(theta4) - sin
(theta4)*cos(theta2)*cos(theta3))*cos(theta6), -L2*sin(theta2) - L3*cos(theta2)*cos(theta3) - L4*(sin(theta2)*cos(th
eta4) + sin(theta4)*cos(theta2)*cos(theta3)) + L5*((sin(theta2)*sin(theta4) - cos(theta2)*cos(theta3)*cos(theta4))*c
os(theta5) + sin(theta3)*sin(theta5)*cos(theta2))], [0, 0, 0, 1]])
```

Figure 6: Python output of final transformation matrix.

obtained by differentiating the position vector with respect to joint velocities. The snapshot of the calculation in python is shown in Figure 7. Hence the Jacobian comes out to be;

$$J_{trans} = \begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} & J_{15} & J_{16} & J_{17} \\ J_{21} & J_{22} & J_{23} & J_{24} & J_{25} & J_{26} & J_{27} \\ J_{31} & J_{32} & J_{33} & J_{34} & J_{35} & J_{36} & J_{37} \end{bmatrix}$$

where

$$\mathrm{J}_{11} = -L1 * s(1) - L2 * s(1) * c(2) + L3 * (s(1) * s(2) * c(3) - s(3) * c(1)) - L4 * ((-s(1) * s(2) * c(3) + s(3) * c(1)) * s(4) + s(1) * c(2) * c(4)) + L5 * (((s(1) * s(2) * c(3) - s(3) * c(1)) * c(4) + s(1) * s(4) * c(2)) * c(5) + (-s(1) * s(2) * s(3) - c(1) * c(3)) * s(5))$$

$$\mathrm{J}_{12} = J_{13} = J_{22} = J_{23} = J_{31} = J_{32} = J_{33} = 0$$

$$\mathrm{J}_{14} = J_{14} = L5 * (-((-s(1) * s(3) - s(2) * c(1) * c(3)) * c(4) - s(4) * c(1) * c(2)) * s(5) + (-s(1) * c(3) + s(2) * s(3) * c(1)) * c(5))$$

```
pv = T07[0:3,3]
```

```
J1 = diff(pv,theta1)
J2 = diff(pv,theta2)
J3 = diff(pv,theta3)
J4 = diff(pv,theta4)
J5 = diff(pv,theta5)
J6 = diff(pv,theta6)
J7 = diff(pv,theta7)
```

```
J_trans = J1.col_insert(1,J2).col_insert(1,J3).col_insert(1,J4).
↪col_insert(1,J5).col_insert(1,J6).col_insert(1,J7)
print(J_trans)
```

Figure 7: Python code for Translation Jacobian.

$$J_{15} = -L4 * ((s(1) * s(3) + s(2) * c(1) * c(3)) * c(4) + s(4) * c(1) * c(2)) + L5 * (-(-s(1) * s(3) - s(2) * c(1) * c(3)) * s(4) - c(1) * c(2) * c(4)) * c(5)$$

$$J_{16} = L3 * (-s(1) * c(3) + s(2) * s(3) * c(1)) - L4 * (s(1) * c(3) - s(2) * s(3) * c(1)) * s(4) + L5 * ((s(1) * s(3) + s(2) * c(1) * c(3)) * s(5) + (-s(1) * c(3) + s(2) * s(3) * c(1)) * c(4) * c(5))$$

$$J_{17} = -L2 * s(2) * c(1) - L3 * c(1) * c(2) * c(3) - L4 * (s(2) * c(1) * c(4) + s(4) * c(1) * c(2) * c(3)) + L5 * ((s(2) * s(4) * c(1) - c(1) * c(2) * c(3) * c(4)) * c(5) + s(3) * s(5) * c(1) * c(2))$$

$$J_{21} = L1 * c(1) + L2 * c(1) * c(2) + L3 * (-s(1) * s(3) - s(2) * c(1) * c(3)) - L4 * ((s(1) * s(3) + s(2) * c(1) * c(3)) * s(4) - c(1) * c(2) * c(4)) + L5 * (((-s(1) * s(3) - s(2) * c(1) * c(3)) * c(4) - s(4) * c(1) * c(2)) * c(5) + (-s(1) * c(3) + s(2) * s(3) * c(1)) * s(5))$$

$$J_{24} = L5 * (-((-s(1) * s(2) * c(3) + s(3) * c(1)) * c(4) - s(1) * s(4) * c(2)) * s(5) + (s(1) * s(2) * s(3) + c(1) * c(3)) * c(5))$$

$$J_{25} = -L4 * ((s(1) * s(2) * c(3) - s(3) * c(1)) * c(4) + s(1) * s(4) * c(2)) + L5 * (-(-s(1) * s(2) * c(3) + s(3) * c(1)) * s(4) - s(1) * c(2) * c(4)) * c(5)$$

$$J_{26} = L3 * (s(1) * s(2) * s(3) + c(1) * c(3)) - L4 * (-s(1) * s(2) * s(3) - c(1) * c(3)) * s(4) + L5 * ((s(1) * s(2) * s(3) + c(1) * c(3)) * c(4) * c(5) + (s(1) * s(2) * c(3) - s(3) * c(1)) * s(5))$$

$$J_{27} = -L2 * s(1) * s(2) - L3 * s(1) * c(2) * c(3) - L4 * (s(1) * s(2) * c(4) + s(1) * s(4) * c(2) * c(3)) + L5 * ((s(1) * s(2) * s(4) - s(1) * c(2) * c(3) * c(4)) * c(5) + s(1) * s(3) * s(5) * c(2))$$

$$J_{34} = L5 * (-(s(2) * s(4) - c(2) * c(3) * c(4)) * s(5) + s(3) * c(2) * c(5))$$

$$J_{35} = -L4 * (-s(2) * s(4) + c(2) * c(3) * c(4)) + L5 * (s(2) * c(4) + s(4) * c(2) * c(3)) * c(5)$$

$$J_{36} = L3 * s(3) * c(2) + L4 * s(3) * s(4) * c(2) + L5 * (s(3) * c(2) * c(4) * c(5) + s(5) * c(2) * c(3))$$

$$J_{36} = -L2 * c(2) + L3 * s(2) * c(3) - L4 * (-s(2) * s(4) * c(3) + c(2) * c(4)) + L5 * ((s(2) * c(3) * c(4) + s(4) * c(2)) * c(5) - s(2) * s(3) * s(5))$$

And the rotation Jacobian is obtained extracting the rotation matrices from the homogeneous transforms and multiplying them with $\hat{Z}$. The python code block is shown in 8 and the output rotation Jacobian matrix J_{rot} which is formed by inserting the rotation vectors in the Jacobian of dimensions $(3 * 7)$ is shown in Figure 9.

```
R01 = T01[0:3,0:3]*Matrix([[0],[0],[1]])
R02 = T02[0:3,0:3]*Matrix([[0],[0],[1]])
R03 = T03[0:3,0:3]*Matrix([[0],[0],[1]])
R04 = T04[0:3,0:3]*Matrix([[0],[0],[1]])
R05 = T05[0:3,0:3]*Matrix([[0],[0],[1]])
R06 = T06[0:3,0:3]*Matrix([[0],[0],[1]])
R07 = T07[0:3,0:3]*Matrix([[0],[0],[1]])
```

```
J_rot = R01.col_insert(1,R02).col_insert(1,R03).col_insert(1,R04).
↪col_insert(1,R05).col_insert(1,R06).col_insert(1,R07)
print(J_rot)
```

Figure 8: Python code for Rotation Jacobian.

```
Matrix([[0, (((-sin(theta1)*sin(theta3) - sin(theta2)*cos(theta1)*cos(theta3))*cos(theta4) - sin(theta4)*cos(theta1)
*cos(theta2))*cos(theta5) + (-sin(theta1)*cos(theta3) + sin(theta2)*sin(theta3)*cos(theta1))*sin(theta5))*sin(theta
6) + ((-sin(theta1)*sin(theta3) - sin(theta2)*cos(theta1)*cos(theta3))*sin(theta4) + cos(theta1)*cos(theta2)*cos(the
ta4))*cos(theta6), -((-sin(theta1)*sin(theta3) - sin(theta2)*cos(theta1)*cos(theta3))*cos(theta4) - sin(theta4)*cos
(theta1)*cos(theta2))*sin(theta5) + (-sin(theta1)*cos(theta3) + sin(theta2)*sin(theta3)*cos(theta1))*cos(theta5), (-
sin(theta1)*sin(theta3) - sin(theta2)*cos(theta1)*cos(theta3))*sin(theta4) + cos(theta1)*cos(theta2)*cos(theta4), -s
in(theta1)*cos(theta3) + sin(theta2)*sin(theta3)*cos(theta1), cos(theta1)*cos(theta2), -sin(theta1)], [0, (((-sin(th
eta1)*sin(theta2)*cos(theta3) + sin(theta3)*cos(theta1))*cos(theta4) - sin(theta1)*sin(theta4)*cos(theta2))*cos(thet
a5) + (sin(theta1)*sin(theta2)*sin(theta3) + cos(theta1)*cos(theta3))*sin(theta5))*sin(theta6) + ((-sin(theta1)*sin
(theta2)*cos(theta3) + sin(theta3)*cos(theta1))*sin(theta4) + sin(theta1)*cos(theta2)*cos(theta4))*cos(theta6), -((-
sin(theta1)*sin(theta2)*cos(theta3) + sin(theta3)*cos(theta1))*cos(theta4) - sin(theta1)*sin(theta4)*cos(theta2))*si
n(theta5) + (sin(theta1)*sin(theta2)*sin(theta3) + cos(theta1)*cos(theta3))*cos(theta5), (-sin(theta1)*sin(theta2)*c
os(theta3) + sin(theta3)*cos(theta1))*sin(theta4) + sin(theta1)*cos(theta2)*cos(theta4), sin(theta1)*sin(theta2)*sin
(theta3) + cos(theta1)*cos(theta3), sin(theta1)*cos(theta2), cos(theta1)], [1, ((sin(theta2)*sin(theta4) - cos(theta
2)*cos(theta3)*cos(theta4))*cos(theta5) + sin(theta3)*sin(theta5)*cos(theta2))*sin(theta6) + (-sin(theta2)*cos(theta
4) - sin(theta4)*cos(theta2)*cos(theta3))*cos(theta6), -(sin(theta2)*sin(theta4) - cos(theta2)*cos(theta3)*cos(theta
4))*sin(theta5) + sin(theta3)*cos(theta2)*cos(theta5), -sin(theta2)*cos(theta4) - sin(theta4)*cos(theta2)*cos(theta
3), sin(theta3)*cos(theta2), -sin(theta2), 0]])
```

Figure 9: Python snapshot showing the Rotation Jacobian matrix for Baxter Left Arm.

9 Forward kinematics Validation

I have chosen the home configuration as shown in Figure 5 where the joint angles are known to be $\theta_1 = \theta_3 = \theta_4 = \theta_5 = \theta_6 = \theta_7 = 0$ and $\theta_2 = 90$; which has already been considered while calculating its transformation matrix ${}^{1}_{2}T$(refer python code fk.py).Using the position vector from base frame to the end effector frame ${}^{0}_{7}P$ which is calculated by multiplying the final transformation matrix ${}^{0}_{7}T$ with position of the end effector P_{tool}. The snap of python code for the same is shown in Figure 10. The position vector is calculated as:

```
In [106]: P_tool = Matrix([[0],[0],[symbols('L6')],[1]])
          pprint(P_tool)
```

$$\begin{bmatrix} 0 \\ 0 \\ L_6 \\ 1 \end{bmatrix}$$

Figure 10: Python snap showing the Position of Tool frame.

$$ {}^{0}_{7}P = {}^{0}_{7}T * P_{tool} $$

which comes out to be;

$$ {}^{0}_{7}P = Matrix([[L1*c(1)+L2*c(1)*c(2)+L3*(-s(1)*s(3)-s(2)*c(1)*c(3))-L4*(-(-s(1)*s(3)-s(2)*c(1)*c(3))*s(4)-c(1)*c(2)*c(4))+L5*(((-s(1)*s(3)-s(2)*c(1)*c(3))*c(4)-s(4)*c(1)*c(2))*c(5)+(-s(1)*c(3)+s(2)*s(3)*c(1))*s(5))+L6*((((-s(1)*s(3)-s(2)*c(1)*c(3))*c(4)-s(4)*c(1)*c(2))*c(5)+(-s(1)*c(3)+s(2)*s(3)*c(1))*s(5))*s(6)+((-s(1)*s(3)-s(2)*c(1)*c(3))*s(4)+c(1)*c(2)*c(4))*c(6))], [L1*s(1)+L2*s(1)*c(2)+L3*(-s(1)*s(2)*c(3)+s(3)*c(1))-L4*(-(-s(1)*s(2)*c(3)+s(3)*c(1))*s(4)-s(1)*c(2)*c(4))+L5*(((-s(1)*s(2)*c(3)+s(3)*c(1))*c(4)-s(1)*s(4)*c(2))*c(5)+(s(1)*s(2)*s(3)+c(1)*c(3))*s(5))+L6*((((-s(1)*s(2)*c(3)+s(3)*c(1))*c(4)-s(1)*s(4)*c(2))*c(5)+(s(1)*s(2)*s(3)+c(1)*c(3))*s(5))*s(6)+((-s(1)*s(2)*c(3)+s(3)*c(1))*s(4)+s(1)*c(2)*c(4))*c(6))], [-L2*s(2)-L3*c(2)*c(3)-L4*(s(2)*c(4)+s(4)*c(2)*c(3))+L5*((s(2)*s(4)-c(2)*c(3)*c(4))*c(5)+s(3)*s(5)*c(2))+L6*(((s(2)*s(4)-c(2)*c(3)*c(4))*c(5)+s(3)*s(5)*c(2))*s(6)+(-s(2)*c(4)-s(4)*c(2)*c(3))*c(6))], [1]]) $$

The validation test vector P_{test} comes to be as shown in figure 12. This result is valid as by inspection we can see that summation of link lengths $L1 + L_2 + L_4$ and L_6; the tool position are in the X_0 direction with respect to end effector frame. And the lengths $L_3 + L_5$ are the offsets in the negative Z_0 direction.

```
P_test=P07.subs('theta1',0).subs('theta2',0).subs('theta3',0).subs('theta4',0).subs('theta5',0).subs('theta6',0).subs
pprint(P_test)
```

```
⎡L₁ + L₂ + L₄ + L₆⎤
⎢                 ⎥
⎢        0        ⎥
⎢                 ⎥
⎢    -L₃ - L₅     ⎥
⎢                 ⎥
⎣        1        ⎦
```

Figure 11: Python output for Forward Kinematics validation using test vector P_{test}.

10 Workspace

The workspace analysis of left arm of Baxter using the transformation matrix along with the lengths of the links and offsets using the DH parameters and the real values of those lengths is shown in the Figure **??**. The python code for the same is included in the python file.

```
: P_test0=P07.subs('L1',69.00).subs('L2',364.35).subs('L3',69.00).subs('L4',374.29).subs('L5',10.00).subs('L6',368.30)

: P_test1=P_test0.subs('theta1',pi/4).subs('theta2',0).subs('theta3',0).subs('theta4',0).subs('theta5',0).subs('theta6'
  P_test2=P_test0.subs('theta1',0).subs('theta2',pi/4).subs('theta3',0).subs('theta4',0).subs('theta5',0).subs('theta6'
  P_test3=P_test0.subs('theta1',0).subs('theta2',0).subs('theta3',pi/4).subs('theta4',0).subs('theta5',0).subs('theta6'
  P_test4=P_test0.subs('theta1',0).subs('theta2',0).subs('theta3',0).subs('theta4',pi/4).subs('theta5',0).subs('theta6'
  P_test5=P_test0.subs('theta1',0).subs('theta2',0).subs('theta3',0).subs('theta4',0).subs('theta5',pi/4).subs('theta6'
  P_test6=P_test0.subs('theta1',0).subs('theta2',0).subs('theta3',0).subs('theta4',0).subs('theta5',0).subs('theta6',pi
  P_test7=P_test0.subs('theta1',0).subs('theta2',0).subs('theta3',0).subs('theta4',0).subs('theta5',0).subs('theta6',0)
  P_test8=P_test0.subs('theta1',-pi/4).subs('theta2',0).subs('theta3',0).subs('theta4',0).subs('theta5',0).subs('theta6
  P_test9=P_test0.subs('theta1',0).subs('theta2',-pi/4).subs('theta3',0).subs('theta4',0).subs('theta5',0).subs('theta6
  P_test10=P_test0.subs('theta1',0).subs('theta2',0).subs('theta3',-pi/4).subs('theta4',0).subs('theta5',0).subs('theta
  P_test11=P_test0.subs('theta1',0).subs('theta2',0).subs('theta3',0).subs('theta4',-pi/4).subs('theta5',0).subs('theta
  P_test12=P_test0.subs('theta1',0).subs('theta2',0).subs('theta3',0).subs('theta4',0).subs('theta5',-pi/4).subs('theta

: P_ws = P_test1.col_insert(1,P_test2).col_insert(1,P_test3).col_insert(1,P_test4).col_insert(1,P_test5).col_insert(1,P
  print(P_ws)

  Matrix([[587.97*sqrt(2), 1175.94000000000, 433.35 + 376.295*sqrt(2), 1175.94000000000, 69.0 + 592.97*sqrt(2), 587.97
  *sqrt(2), 1175.94000000000, 184.15*sqrt(2) + 807.64, 1175.94000000000, 433.35 + 366.295*sqrt(2), 1175.94000000000, 6
  9.0 + 513.97*sqrt(2)], [587.97*sqrt(2), -5.0*sqrt(2), 0, -39.5*sqrt(2), 0, -587.97*sqrt(2), 0, 0, 5.0*sqrt(2), 0, 3
  9.5*sqrt(2), 0], [-79.0000000000000, -69.0 - 5.0*sqrt(2), -69.0 + 366.295*sqrt(2), -39.5*sqrt(2), 513.97*sqrt(2), -7
  9.0000000000000, -79.0000000000000, -184.15*sqrt(2) - 79.0, -69.0 - 5.0*sqrt(2), -376.295*sqrt(2) - 69.0, -39.5*sqrt
  (2), -592.97*sqrt(2)], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

Figure 12: Python output for workspace analysis

11 Assumptions

In this project I am trying to work on the original Baxter Intera 3 model without making any changes. Assumptions are that all the joints are considered to be rigid, the friction and the other external disturbances are not taken into account, robot self-collision is not considered, the conditions on ISS are the same as earth.

12 Conclusions

The project focuses on the detailed kinematic analysis for the two 7-dof Baxter Humanoid Robot arms. The Craig (modified) Denavit-Hartenberg Parameters for each serial chain, specific length

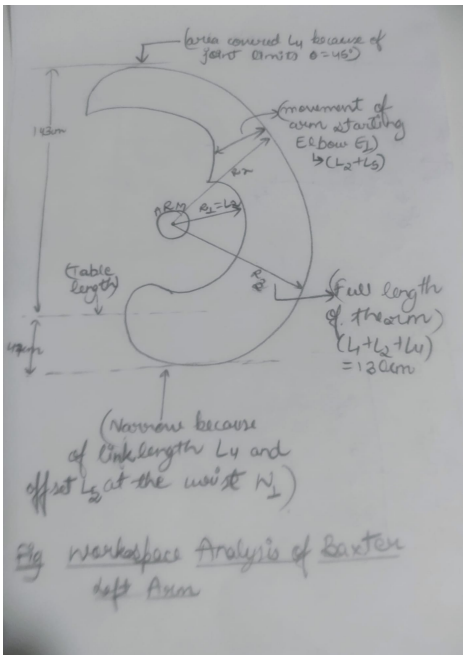

Figure 13: A rough workspace analysis of Baxter arm.

13 Future works

Space farming is very useful for the deep space missions. I wanted to learn the use of Baxter Research Robot for the same and if possible learn to use MoveIt files provided by the makers of the robot and use them on the actual Baxter bot.

References

[1] John Craig. *Introduction to Robotics Mechanics and Control; 4th Edition.* Pearson Pub., 1955.

[2] T.L. Harman and Carol Fairchild. Introduction to baxter. Publication,https://sceweb.sce.uhcl.edu/harman/CENG5931Baxter2015/Guides/ $BAXTER_Introduction_2082016.pdf$.

[3] By Hal Hodson. Baxter the robot brings his gentle touch to novel jobs. 2014. https://www.newscientist.com/article/mg22329793-700-baxter-the-robot-brings-his-gentle-touch-to-novel-jobs/.

[4] R.L. Williams II. Baxter humanoid robot kinematics. Publication,https://www.ohio.edu/mechanicalfaculty/williams/html/pdf/Baxter Kinematics.pdf,April 2017.