ENPM662 - Introduction to Robot Modeling

Project 2
Modeling of Transformer Robot

AUTHOR:

SHYAMSUNDAR PRABHAKAR INDRA

UID - 119360842

DIRECTORY ID - SHYAMPI

ABSTRACT

This project explains the design, control, and simulation of a novel 'Transformer Robot' which can potentially make a huge difference of how storage systems work, which can especially be a huge game-changer in today's world with the amount of products being produced everyday. The robot consists of a mobile base and a manipulator arm, which were designed using Solidworks[1] and were exported as a URDF. A ROS[2] package was developed, with transmissions and controllers, and integrated with the URDF. A camera was attached to the end effector of the manipulator arm to visualize the environment. The forward and inverse kinematics of the robot's arm are calculated and verified. The robot has been simulated and controlled in Gazebo. A simulation of the robot doing a planned task in a custom environment has also been done. The challenges faced and future work planned have been described.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Chapter 1

INTRODUCTION

Mobile Manipulators aren't a rarity in today's technologically advanced world. However almost all the mobile manipulators one can come across are pretty tall in z-height, if not for their large overall size. One classical example is that of the Husky UGV with its tall Manipulator arm sticking out.

Contrary to such tall mobile manipulators, a novel 'Transformer Robot' is being introduced. Getting its name from the famous Transformers cartoon, it is indeed yet another Mobile Manipulator, but quite different. The Transformer Robot has a collapsible arm, i.e., the manipulator arm is custom designed such that in it's 'collapsed' state it's almost in level with the mobile base itself.

Such a robot with a reduced z-height, opens up to a plethora of opportunities. One of the most prominent is the way storage industries work. Conventionally such storage houses have pre-defined long pathways for the forklifts to move in a very inefficient manner as shown in figure 1.1. The Transformer robot can potentially revolutionize the way such storage houses work. When provided with shelves with a gap underneath, the Transform Robot with it's low-profile could easily navigate through these gaps, hence reducing the distance covered in the to and fro motion to a great level. Though this feels trivial, when looked at this over a period of years and years, this shall save thousands of hours and miles of extra work. This motion efficiency, can boost the energy efficiency of these places as well.

This project report describes the work done in designing, controlling, and simulating the above proposed Transformer Robot. The report is divided into various sections. The introduction provides an overview of the Transformer Robot and its novel applications. The detailed specifications of the Transformer Robot are described in the section titled **"The Transformer Robot"** The **"Kinematics and Kinematic Validation"** section covers the forward and inverse kinematics of the robot's manipulator arm and the validation of the calculations. The **"Workspace Study"** section discusses the total reach of the arm and the limitations of the space it cannot reach. The **"Assumptions"** section outlines the assumptions made during the development and design of the Transformer Robot. The **"Package, Control Method and Visualization"** provides a link to the entire working package itself,

Figure 1.1: Conventional Forklift in a Storage House[3]

the way the joints in the robots are controlled, and a visualization of the robot performing a specific task. The **"Problems Faced and Lessons Learned"** section covers the challenges encountered during the project and the lessons learned from them. The **"Future Work"** section lists potential improvements to the Transformer Robot. The **"Conclusion"** section summarizes the project, and the **"References"** section lists the sources used.

Chapter 2

THE TRANSFORMER ROBOT

This section briefs the proposed design, specifications, and the planned task of the robot being modeled for this project.

2.1 Specifications of the Transformer Robot

Mobile Base	
Length of the Mobile Base	0.914 m
Width of the Mobile Base	0.508 m
Height of the Mobile Base	0.3 m
Mass (Base + Wheels + Axles)	82.6 Kg
Wheel Base	0.508 m
Motion Model	Unicycle
Degrees of Freedom	3 in total - x, y and yaw.
Manipulator Arm	
Height of Arm with all joints extended	2.388 m
End Effector Used	Simple custom 1 DOF Grasping tool
Degrees of Freedom	7 in total - Five revolute and one prismatic, and one DoF for the End Effector
Length of End Effector	0.09 m

2.2 Type of Robot

The type of robot used here is a mobile manipulator. Mobile manipulators are used in several applications all over Earth and even beyond Earth. For example, Husky UGV is a mobile manipulator which is used for all-terrain Robotics research, and a Mars rover is also a type of mobile manipulator. The manipulator arm is a serial manipulator.

2.3 Degrees of Freedom of The System

The system has a total of 9 DoF. The mobile base has 3 DoF while the manipulator arm has 5 DoF. The mobile base can move along x and y directions and rotate around the z axis. The arm has 3 revolute joints, with a prismatic joint and revolute joint for the end-effector, and the end-effector itself has 1 DoF.

2.4 Application of The Transformer Robot

As the introduction described, the robot is a novel attempt at reforming how storages work. Conventionally, most of the storage houses use forklifts operated by humans, which have to navigate through longer pathways. The central idea behind the Transformer robot is a mobile manipulator being able to reduce the vertical height of itself so that it can easily navigate through remote spaces.

The proposed idea of reformed storage systems is that the shelves have ample space underneath them, enough for the transformer robot to pass through in their collapsed state. This will allow the transformer robot to navigate to the point of interest in the shortest way possible, hence saving thousands and thousands of man-hours and power in the long run.

2.5 Motivation for Choosing Transformer Robot

The main motivation was the novelty behind the transformer robot, which enables it to be used as a multi-purpose robot which, as explained before, can be used for efficient pick and place operations and at the same time, owing to its small size in its collapsed state, and can be used for search and rescue operations having acces to remote areas in case of anomalies at the same time.

The project presented an opportunity to create a holistic model right from basic 3D modelling to actually making a simulation of the same. The idea of modelling what was learnt in the theory into an actual working model from scratch was also a great motivation for the choice.

Chapter 3

CAD MODELS

This section contains pictures of the Transformer Robot assembly and that of the manipulator arm assembly in its two different modes.

3.1 The Transformer Robot CAD Model

The model of the Transformer Robot was created from scratch using Solidworks. The Robot's two modes are shown in figure 3.1 and figure 3.2.

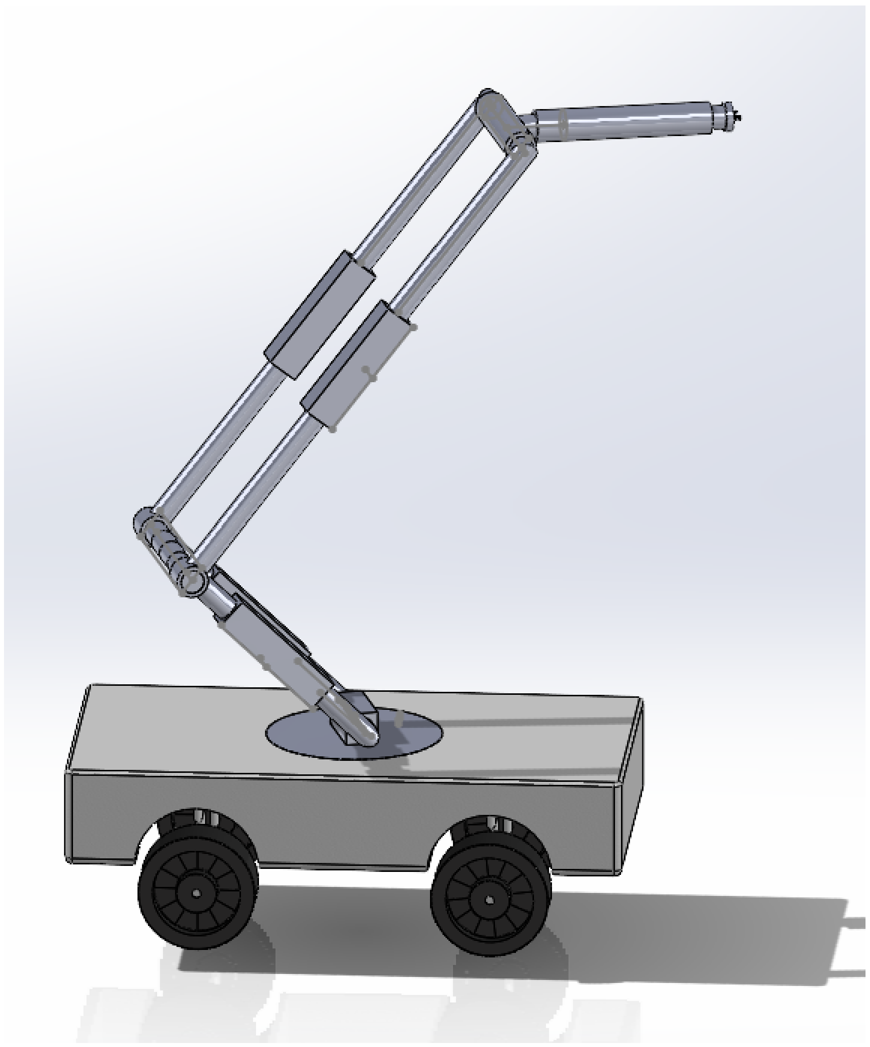


Figure 3.1: Transformer Robot in its extended mode

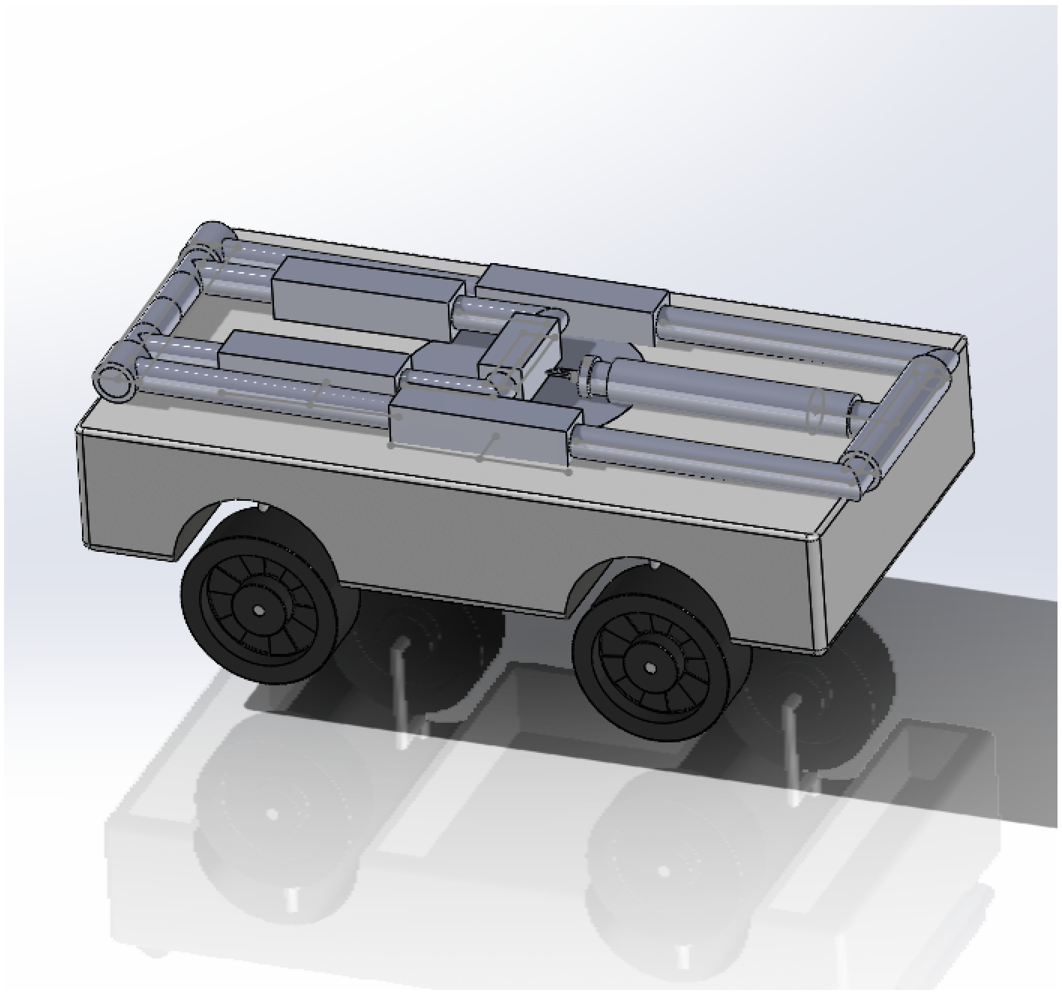


Figure 3.2: Transformer Robot in its collapsed mode

3.2 The Arm Model

This section has two views of the arm model as well as a close up of the custom designed end effector as shown in figure 3.3 and figure 3.4. The arm model has a rotating base which is attached to the mobile base, allowing full freedom of revolution. This joint is followed by three revolute joints with their rotation axis parallel to each other. Hence the arm itself has 6 degrees of freedom. This is followed by a prismatic joint, revolute joint and then the end effector.

The end effector is a simple setup with two fingers, each having a revolute joint tied to each other, hence having 1 DoF in total.

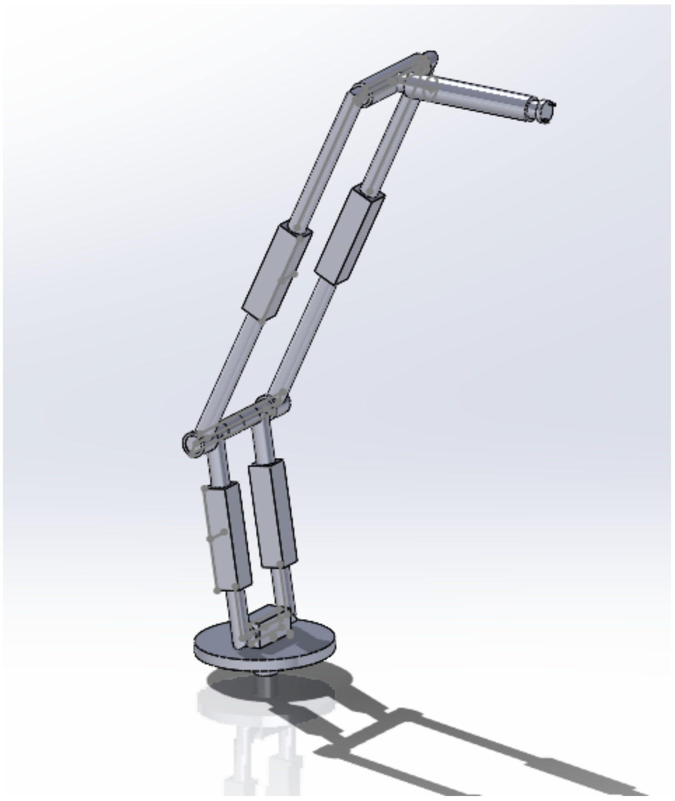


Figure 3.3: Manipulator Arm extended mode

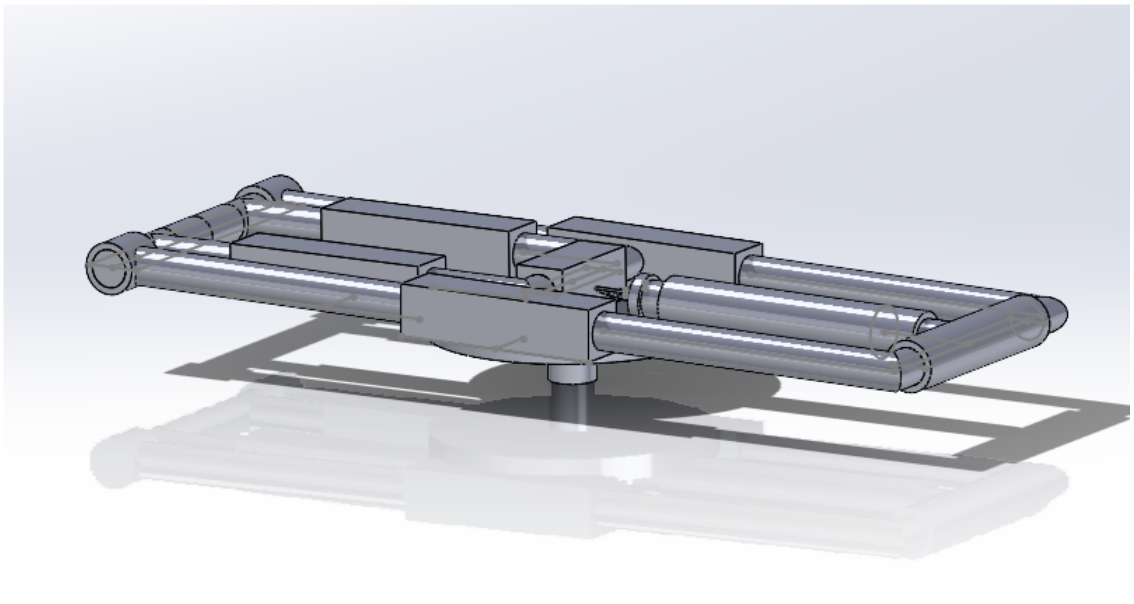


Figure 3.4: Manipulator Arm collapsed mode

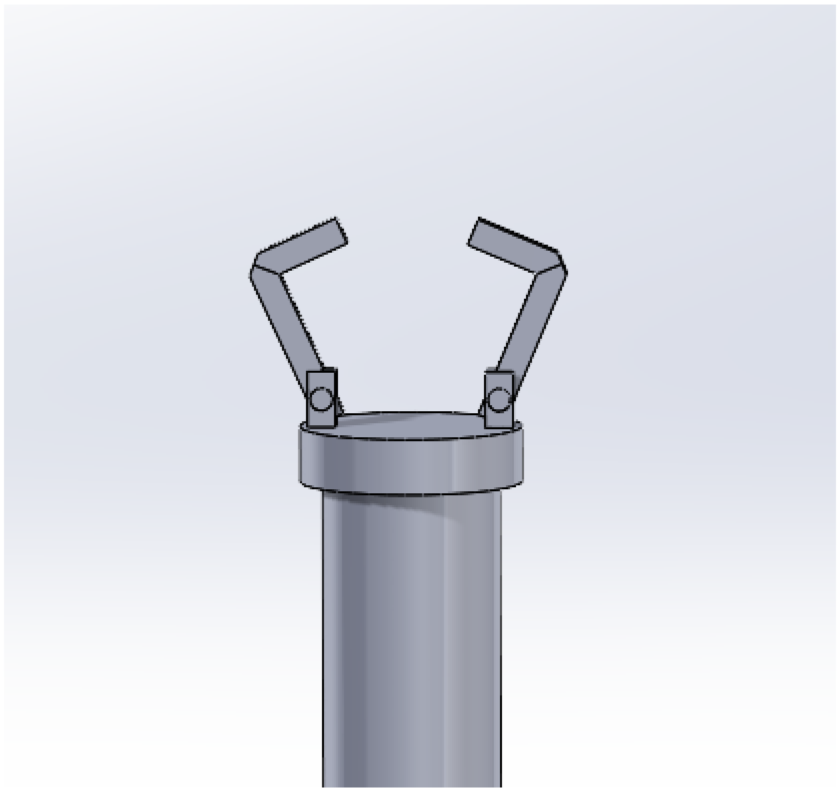


Figure 3.5: Close-Up of End Effector

Chapter 4

KINEMATICS

4.1 Forward Kinematics

Forward Kinematics is the computation of the end effector position given the joint angles at each joint. The Forward Kinematics is calculated only for the manipulator arm, assuming that it's mounted on top of a fixed base (Mobile base's wheels are fixed). The following steps are involved in calculation of the Forward Kinematics for the manipulator arm:

- Assigning the frames at each joint
- Finding the D-H table
- Calculating the Final Transformation Matrix

The manipulator of the Transformer robot has 6 degrees of freedom excluding the end effector, and the first joint is not the ground frame. Therefore 8 frames are required to describe the manipulator arm, 6 for the joints, 1 for ground frame and 1 for the end effector frame. However when formulating the DH[4] table, it was observed that it was necessary to introduce two dummy frames in order to successfully satisfy DH constraints. The assigned frames are as shown in Figure 4.1

The D-H table is calculated by using the assigned frames and the calculated D-H table is shown in Table 1. The D-H table is calculated by following the rules:

- θ_n is the rotation at the revolute joints
- q is the linear displacement at the prismatic joint
- a_n is the distance between the Z_{n-1} and Z_n (along X_n axis)
- d_n is the distance between the O_{n-1} and intersection of Z_{n-1} and X_n along Z_{n-1} axis
- α_n is the angle between Z_{n-1} and Z_n measured around X_n

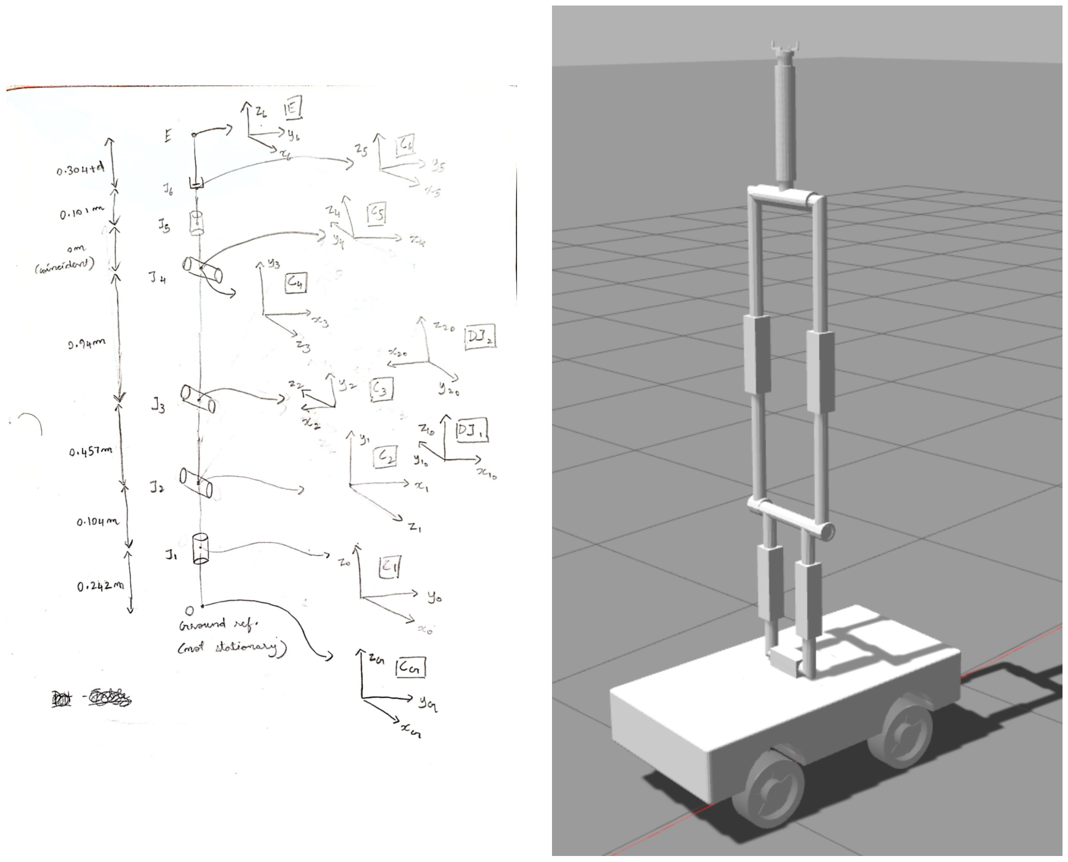


Figure 4.1: Coordinate Systems assignment

D-H TABLE				
	α	θ	d	a
G - 0	0	0	0.242	0
0 - 1	$\pi/2$	θ_1	0.104	0
1 - 1_0	$-\pi/2$	θ_2	0	0
1_0 - 2	$-\pi/2$	0	0.457	0
2 - 2_0	$\pi/2$	θ_3	0	0
2_0 - 3	$\pi/2$	0	0.94	0
3 - 4	$-\pi/2$	θ_4	0	0
4 - 5	0	θ_5	0.101	0
5 - 6	0	0	0.304 + q	0

Then using the D-H table and the assigned frames the final transformation matrix was calculated which gives the pose and orientation of the end-effector. To find the final transformation matrix we have to follow the following steps:

1. Individual Transformation matrices are to be found (i.e.) T_G^0, T_0^1, $T_1^{1_0}$, $T_{1_0}^2$, $T_2^{2_0}$, $T_{2_0}^3$, T_3^4, T_4^5 and T_5^6
2. The individual transformation can be found by plugging in the joint angle values and the rest of the values from the D-H table given above into the below matrix:

$$\begin{bmatrix} cos\theta_i & -sin\theta_i cos\alpha_i & sin\theta_i sin\alpha_i & a_i cos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sin\alpha_i & a_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. The final transformation can be found by multiplying all the individual transformation matrices in the format given below

$$T_G^6 = T_G^0 * T_0^1 * T_1^{1_0} * T_{1_0}^2 * T_2^{2_0} * T_{2_0}^3 * T_3^4 * T_4^5 * T_5^6$$

The Final transformation matrix after all the computations is shown is too long to be shown as an image. Please follow this link to the Google Colab notebook to see the printed Final Transformation matrix:

FK and IK Results

4.2 Inverse Kinematics

In Inverse Kinematics, we do the reverse of Forward Kinematics by calculating the joint angles by using a known end-effector position. One can either calculate one of these or both, for Inverse Kinematics :

- Inverse Position Kinematics
- Inverse Velocity Kinematics

For the case of the Transformer robot, the inverse velocity kinematics were computed by calculating the Jacobian. To find the joint angles by using the end-effector position one has to follow these steps:

1. Finding all the individual Transformation matrices and the final transformation by multiplying them
2. Find Z_i which are the column matrices containing the elements of the 3^{rd} column in the T_i matrix
3. After finding the Z_i matrices, Then find the X_{pi} matrices which contain the elements from the last column of each individual matrices
4. Differentiate the matrices X_{p1}, X_{p2}, X_{p3} and X_{p4} with respect to the joint angles θ_1, θ_2, q_3 and θ_4 respectively
5. Then form the individual Jacobian matrices in the format given below

$$ {}^0J_n = \begin{bmatrix} \frac{\partial X_p^0}{\partial q_1} & \cdots & \frac{\partial X_p^0}{\partial q_i} & \cdots & \frac{\partial X_p^0}{\partial q_n} \\ {}^0Z_1 & \dots & {}^0Z_i & \dots & {}^0Z_n \end{bmatrix} $$

The resultant Jacobian is too big to be fit into a screenshot since it's a 6 DoF manipulator. Find the printed parametric Jacobian in the attached Google Colab notebook below:

FK and IK Results

Chapter 5

KINEMATIC VALIDATIONS

5.1 Forward Kinematics Validation

For validating the forward kinematics of the manipulator arm, two sets of joint parameters were inputted to the manipulator arm in Gazebo, and the respective end effector positions in Gazebo were noted down. The same joint parameters were inputted to the final transformation matrix computed in section 4.1, and the respective end effector positions were obtained from the last column of the resultant transformation matrix.

Joint parameters set 1 = [0, 0, 0, 0, 0, 0]
Joint parameters set 2 = [0, 0, $\pi/2$, $\pi/2$, 0, 0]

The position vector from the above two steps for both the cases of joint parameter sets are shown below:

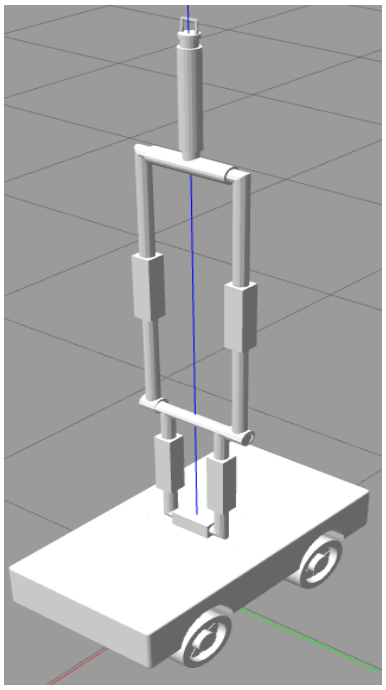
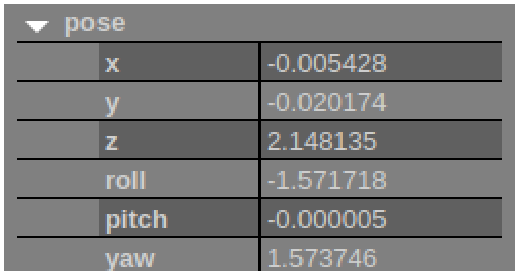
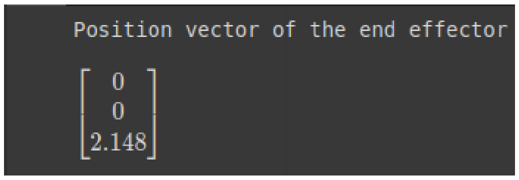


Figure 5.1: Representation of case 1 in Gazebo



pose	
x	-0.005428
y	-0.020174
z	2.148135
roll	-1.571718
pitch	-0.000005
yaw	1.573746



Position vector of the end effector

$$\begin{bmatrix} 0 \\ 0 \\ 2.148 \end{bmatrix}$$

Figure 5.2: FK Validation for Case 1

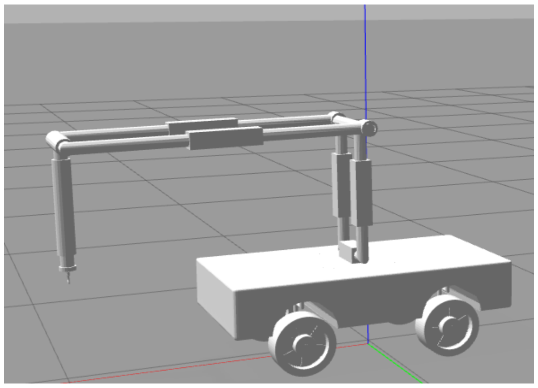
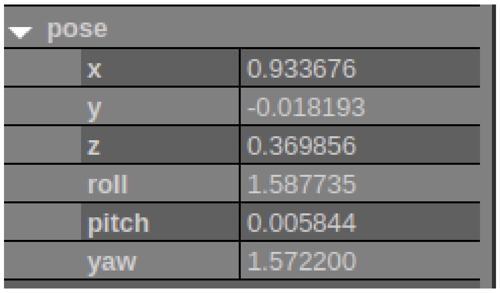
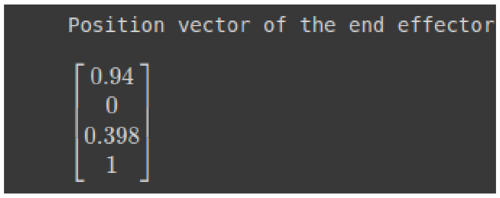


Figure 5.3: Representation of case 2 in Gazebo



pose	
x	0.933676
y	-0.018193
z	0.369856
roll	1.587735
pitch	0.005844
yaw	1.572200



Position vector of the end effector

$$\begin{bmatrix} 0.94 \\ 0 \\ 0.398 \\ 1 \end{bmatrix}$$

Figure 5.4: FK validation for Case 2

5.2 Inverse Kinematics Validation

The validation of the inverse kinematics part was done by plotting a straight line of length 0.2 metres along the z-axis direction in 100 seconds (so that the process is quasistatic). A python script was created to do the same by following the below steps:

1. After finding the final parametric Jacobian matrix in 4.2, the initial angles of the manipulator before the validation are given as input, and the hence the

initial Jacobian is populated.

2. In this case, the Jacobian is a square 6x6 matrix, hence there is no need for pseudoinverse. A small value of delta = 1e-6 is added to all elements of the Jacobian before taking inverse, in order to avoid any singular matrix errors.

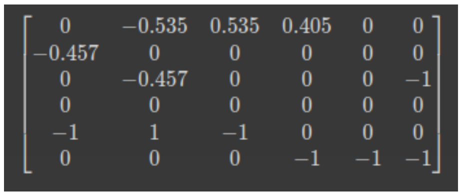
3. Compute $\dot{X}$ according to the trajectory intended to trace. In this case $\dot{X}$ only had a constant z velocity component.

4. Then find the $\dot{Q}$ matrix by multiplying the inverse jacobian matrix and the $\dot{X}$ matrix, where ($\dot{X}$ matrix is the matrix that contains the linear and angular velocity components of the end effector)

5. The $\dot{Q}$ matrix contains the values of the joint angles and those values are obtained and numerical integration is performed to get the corresponding joint angle values for every time step.

6. These computed joint angles are then used as an input to the final Transformation matrix from 4.1, and the end effector position is obtained at every time step

7. The so obtained end effector positions are plotted as a 3D plot and visualized. As seen below, the resultant trajectory traced is a straight line as intended.



$$\begin{bmatrix} 0 & -0.535 & 0.535 & 0.405 & 0 & 0 \\ -0.457 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.457 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -1 \end{bmatrix}$$

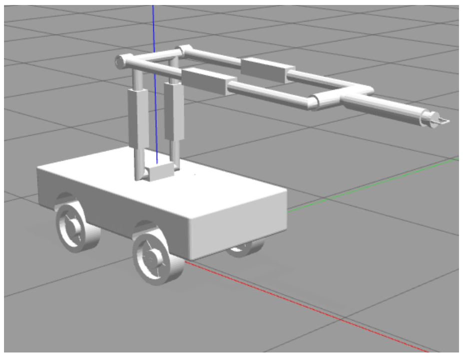


Figure 5.5: Initialized Jacobian Matrix and corresponding robot pose

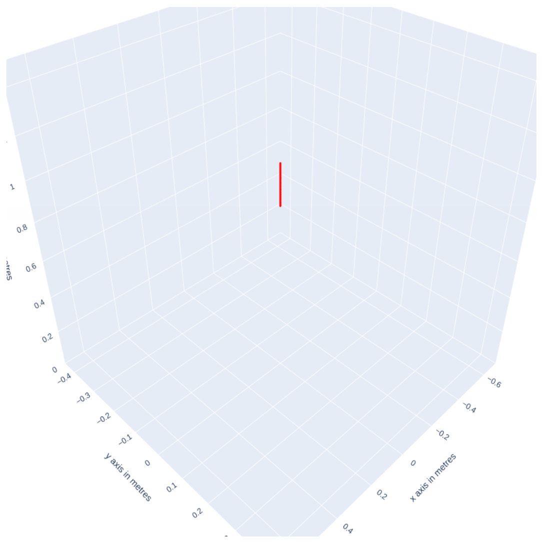


Figure 5.6: Trajectory traced using Jacobian in python script

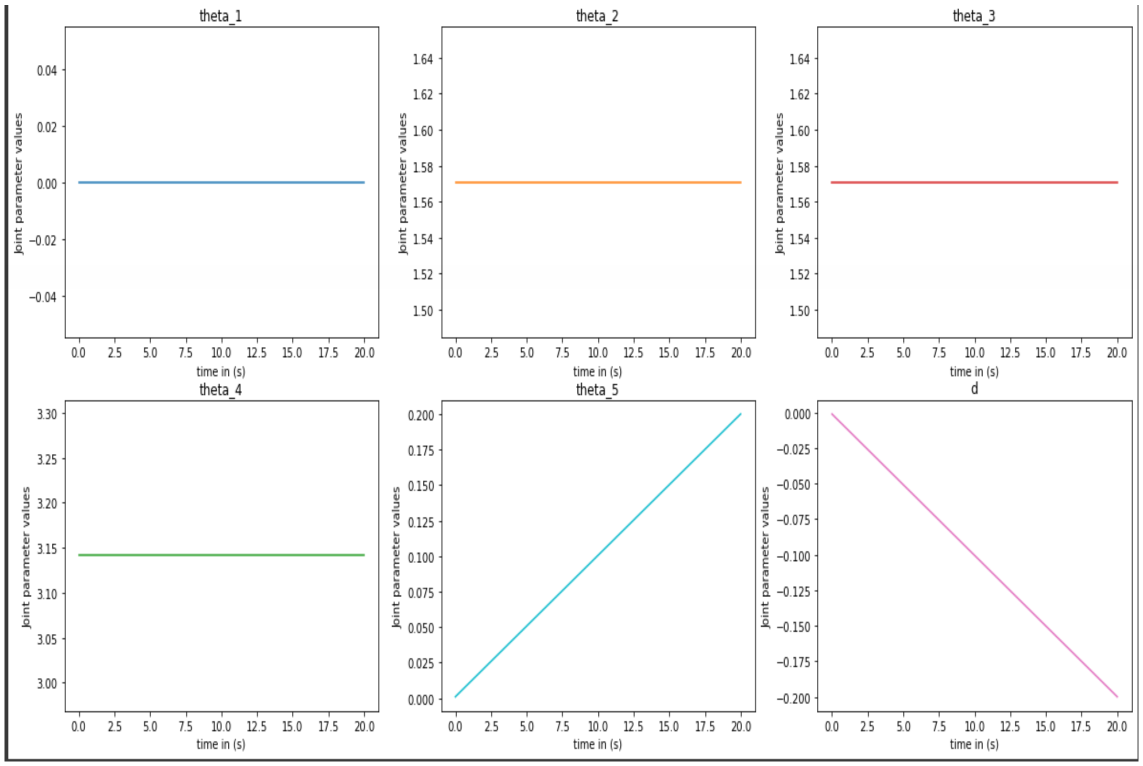


Figure 5.7: Variation of joint parameters with time, to achieve the trajectory

Chapter 6

WORKSPACE STUDY

The workspace study was done for only the manipulator arm by fixing the mobile base (making the wheels static). The workspace of the manipulator arm is a hemisphere of radius 2.25 metres, centred at the origin.

The first revolute joints allows a full 360 degree rotation along the global z axis, and the second revolute joints let's the whole manipulator make a 180 degree sweep, hence achieving a hemisphere. Due to the redundant number of joints, the end effector can actually reach below the plane containing the first joints coordinate axes.

Analytically the reach of the arm was found out by using the parametric end effector position from the final transformation matrix. This was followed by fixing every joint parameter except for θ_1 and θ_2, which were made to sweep [0,360] and [-90,90] degrees respectively, and the point clouds as a result of the sweep were plotted in a 3D plot. The below figure shows the reachable workspace of the manipulator arm.

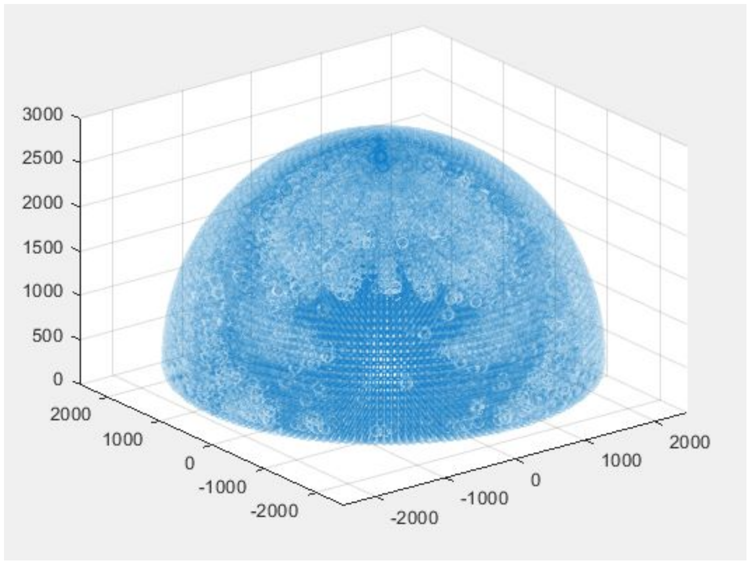


Figure 6.1: Workspace of the Robotic Arm in 'mm'

Chapter 7

ASSUMPTIONS

The following are the assumptions made to make a successful model of the transformer robot, and thereby verify its Forward and Inverse Kinematics:

- All the Kinematics computations for the manipulator arm in its extended state.
- All input parameters were within the range of motion allowed by every joint.
- Although the 3D model of the wheel is smooth, there is friction acting between the wheels and the Gazebo world.
- External forces like friction have not been considered.
- Throughout the period of validation, the mobile base is docked, i.e., zero motion in the mobile base.
- Self collisions between parts of the robot arm were turned off.
- The weight of the mobile base is enough to not let the robot topple at any orientation of the manipulator arm.

Chapter 8

PACKAGE, CONTROL METHOD AND VISUALIZATION

8.1 Package

The package containing all the necessities to spawn and run the transformer robot in Gazebo can be found in the link to my Github repository found below

Repository Link

The repository also contains the 3D models (including the part and sldasm files) of the Transformer Robot. The instructions to run the ROS package itself can be found in the readme file of the repository.

8.2 Control Method

The mobile base of the Transformer robot consists of 2 revolute joints for the steering, and 4 continuous joints for the wheels. The 2 revolute joints for the steering were controlled with Joint position controllers and the two continuous joints of the rear wheel were controlled using Velocity controllers.

For the manipulator arm, there are 5 revolute joints and 1 prismatic joint for the arm itself and 2 revolute joints for the fingers of the end effector. All these joints are controlled using Joint Position controllers.

All the Joint Position controllers needed careful tuning of the PID values[5], without which there was jittering or abrupt motion in the joints. Different joints needed different PID values especially according to the mass and inertia of the link they were controlling.

Controllers YAML file link

8.3 Simulation

The spawing and simulation of the transformer robot was done using Gazebo. The video of the same can be found in the following folder:

Simulation video link

Chapter 9

PROBLEMS FACED AND LESSONS LEARNED

Extremely unstable Ubuntu OS, which I had to reinstall twice during the duration of this project. I had to keep it open for 2 days straight so that there are no last minute risks. Buggy URDF import plugin and several Ubuntu and Gazebo woes. PID controller tuning, which was very difficult for parts of low mass like that of the end effector fingers. Idle drift in Gazebo which is a known issue, which does not have a readily available foolproof fix. Due to the presence of long narrow links, the effect of geometry when actuating these was very difficult to deal with, and required precise tuning of the PID. Had to account for sudden jerks caused by the controllers, by doing trajectory planning and slowly actuating, hence making it a quasistatic process.

9.1 Problems Faced

This section gives an overview of the number of problems I came across when trying to implement the project.

Instability of the Ubuntu OS

There was some extreme instability with the installed Ubuntu OS in my laptop, which ended up crashing and stopping the detection of GPU and wifi adapter thrice during the duration of the project.

Lack of teammate

Due to the lack of a teammate due to unforeseen reasons, the project became infinite times harder. This was made worse by my choice of doing everything in the project from scratch.

Buggy URDF Export

The URDF export was extremely buggy. I had to go through some 6 iterations of URDF generation and visualizing it in Gazebo before I got it to work properly. For example, once it automatically changed the axes defined for the mobile base's wheels, due to which the mobile base stopped moving. This other time, upon exporting the URDF, the plugin decided to replace the mesh of every part of the robot with a single link's mesh file and the resultant model wasn't a sight to behold.

PID Values tuning

The PID tuning for very small parts like that of the end effector fingers was very difficult since the response was extremely sensitive. Whereas for the long parts, PID tuning was extremely difficult due their length and mass, which made them to bounce back and forth before settling at the required joint position.

Sudden Jerks in Joints

High D value in the PID controllers caused sudden jerks when actuating the manipulator. This situation was alleviated by slow trajectory planning, hence making actuation a quasistatic process.

Difficult D-H setup

The created arm for the manipulator arm ended up giving a DH table that was very difficult to formulate. The DH table ended up needing one shift of axes and creation of 2 dummy joints for properly describing the system.

9.2 Lessons Learnt

The project ended up being an extremely steep learning curve especially with time management. The experience of creating a custom model from scratch gave a great learning experience with Solidworks, and URDF import. I also learned how to properly interpret a URDF file, even to the extent of modifying intrinsic parameters of each link to make the model work successfully. Post that, the project also gave a great insight into the working of ROS, writing subscriber and publisher nodes, and programming ROS controllers for every joint of the robot and controlling it using ROS topics and messages.

Especially in the forward and inverse kinematics section, I learned a lot by creating a DH table for the custom manipulator arm model and verifying its Kinematics. It provided hands on experience of transferring the theoretical knowledge gained in ENPM662, to a working simulated model. This process, though tedious, taught me how to correctly assign frames, shift origins, and use dummy frames even for complex systems, as any errors in these steps caused discrepancies between theoretical and simulated values. Choosing controllers for the model's joints also proved to be a challenging lesson, as making incorrect choices resulted in unexpected behaviors when the model was spawned and controlled in Gazebo.

Chapter 10

FUTURE WORK

The project was successfully completed, but there is potential for further improvements in the future. Some ideas for improvement include performing more in depth analysis and addressing minor issues, which can make the transformer robot move a step closer to becoming a real robot in the storage industry. Some of the work I plan to do in the future includes:

- Performing stress analysis in ANSYS Mechanical which is crucial in this case because of Transformer Robot's slender arms
- Adding a LIDAR sensor to the end effector link, so that the end efffector gets a better imaging of what's in its surroundings.
- Increase the Degree of Freedom of the arm so that it can execute more range of motion for various tasks.
- Adding a rotating assembly with multiple tools as the end effector for doing different tasks.
- Adding more cameras especially one to the chassis of the mobile base which would help in successful navigation of the robot.
- Add more sensors to the robot to collect more information about the environment it is in.
- Implementing autonomous navigation for the robot from a start point to a target point.
- Adding a more efficient path planning algorithm rather than just using Inverse Kinematics.

Chapter 11

CONCLUSION

The project was successfully completed, including the study of a novel transformer robot in a virtual environment and its mathematical formulation. The design, simulation, and control of the robot were all successful, and the forward and inverse kinematics were calculated and verified via simulation. The learning outcomes listed in the report were covered, and the project met its goals.

However, some fallback goals, such as simulating the robot in a simpler world and designing the arm with lower number of sensors had to be adopted. The ambitious goals will be part of future work. The project provided a solid understanding of the modeling and control of a complicated mobile manipulator, which could potentially be useful in storage houses for efficient processing, and change the way storage houses work. I also gained skills that can be applied to similar real-time applications and entry-level experience with ROS and Gazebo. I am very thankful to Prof.Reza and the TAs who made this amazing learning experience possible.

REFERENCES

[1] *Solidworks*. https://www.solidworks.com/. [Online].

[2] *Robotic Operating System*. https://www.ros.org/. [Online].

[3] *forklift$_i$mage*. https://www.catlifttruck.com/products/warehouse-equipment. [Online].

[4] *DH Parameters*. https://www.researchgate.net/publication/262166607_Identification_of_Denavit-Hartenberg_Parameters_of_an_Industrial_Robot. [Online]. 2013.

[5] *PID tuning*. https://realpars.com/pid-tuning/. [Online].

Appendix A

PYTHON CODE

Python Code for Kinematics validation of Robotic arm

```
# -*- coding: utf-8 -*-
"""ENPM662_project2_code_shyampi.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1XfMuMxmm27r3gcsec8usFtIJDnRUijgn

##ENPM662 - Project2 Kinematics code
"""

import numpy as np
from matplotlib import pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib import rc
from pprint import pprint

import sympy as sym
from sympy import cos as c
from sympy import sin as s
from sympy import Eq, MatMul, Function, diff
from sympy.matrices import Matrix
from IPython.display import display, Math

import plotly.graph_objs as go
def vis3D(EE_positions):
  X = np.array(EE_positions)
  print(X.shape)
  trace = go.Scatter3d(
    x = X[:,0], y = X[:,1], z = X[:,2],mode = 'markers', marker = dict(
        size = 1,
        color = 'red', # set color to an array/list of desired values
        colorscale = 'reds'
        )
    )

  fig = go.Figure(data = [trace])

  fig.update_layout(
      scene = dict(
              xaxis = dict(nticks=10, range=[-3,3], title = "x axis in metres"),
                      yaxis = dict(nticks=10, range=[-2,2], title = "y axis in me
                      zaxis = dict(nticks=10, range=[-2,2], title = "z axis in me
      width=1000,
      height=1000,
      margin=dict(r=20, l=10, b=10, t=10))
  fig.show()

"""#Problem 1 : Position Kinematics - Panda"""

theta, d, alpha, a = sym.symbols('theta d alpha a', real = 'True')

theta1, theta2, theta3, theta4, theta5, theta6, theta7 = sym.symbols('theta1 thet

d1, d3, d5, d7, a3 = sym.symbols('d1 d3 d5 d7 a3', real = 'True')

# thetadot1, thetadot2, thetadot3 = sym.symbols('thetadot1 thetadot2 thetadot3',

T = Matrix([[c(theta), -s(theta)*c(alpha), s(theta)*s(alpha), a*c(theta)],
            [s(theta), c(theta)*c(alpha), -c(theta)*s(alpha), a*s(theta)],
            [0, s(alpha), c(alpha), d],
            [0, 0, 0, 1]])

print("General Transformation Matrix in DH system: \n")
T
```

```
"""#Individual Transformation Matrices:"""

TG0 = T.subs([(a, 0), (alpha, 0), (d, 0.242), (theta, 0)])

print("Transformation matrix from frame 0 to G:\n")
TG0

T01 = T.subs([(a, 0), (alpha, sym.pi/2), (d, 0.104), (theta, theta1)])
print("Transformation matrix from frame 1 to 0:\n")
T01

T11_ = T.subs([(a, 0), (alpha, -sym.pi/2), (d, 0), (theta, theta2)])
print("Transformation matrix from frame 3 to 2:\n")
T11_

T1_2 = T.subs([(a, 0), (alpha, -sym.pi/2), (d, 0.457), (theta, 0)])
print("Transformation matrix from frame 4 to 3:\n")
T1_2

T22_ = T.subs([(a, 0), (alpha, sym.pi/2), (d, 0), (theta, theta3)])
print("Transformation matrix from frame 5 to 4:\n")
T22_

T2_3 = T.subs([(a, 0), (alpha, sym.pi/2), (d, 0.94), (theta, 0)])
print("Transformation matrix from frame 6 to 5:\n")
T2_3

T34 = T.subs([(a, 0), (alpha, -sym.pi/2), (d, 0), (theta, theta4)])
print("Transformation matrix from frame 7 to 6:\n")
T34

T45 = T.subs([(a, 0), (alpha, 0), (d, 0.101), (theta, theta5)])
print("Transformation matrix from frame 7 to 6:\n")
T45
```

```
T56 = T.subs([(a, 0), (alpha, 0), (d, 0.304+d), (theta, 0)])
print("Transformation matrix from frame 7 to 6:\n")
T56

"""#Final FK Transformation Matrix"""

TG6 = sym.MatMul(TG0,T01,T11_,T1_2,T22_,T2_3,T34,T45,T56, evaluate = True)
print("Final Transformation Matrix before substituting the length values : \n")
TG6

"""#Geometric Validation

Case 1 - All thetas = 0, d = 0
"""

res = TG6.subs([(theta1, 0), (theta2, 0), (theta3, 0), (theta4, 0), (theta5, 0),

print("Position vector of the end effector with respect to the ground coordinate
res[:-1,3]

"""Case 2 - Theta3 = 90 degrees, and theta4 = 270 degrees. All other parameters 0

res = TG6.subs([(theta1, 0), (theta2, 0), (theta3, sym.pi/2), (theta4, 3*sym.pi/2

print("Position vector of the end effector with respect to the first joint’s coor
res[:,3]

"""#Validating Inverse Kinematics"""

TG1 = sym.MatMul(TG0, T01, evaluate = True)

TG2 = sym.MatMul(TG1, T11_,T1_2, evaluate = True)

TG3 = sym.MatMul(TG2, T22_,T2_3, evaluate = True)

TG4 = sym.MatMul(TG3,T34, evaluate = True)
```

```
TG5 = sym.MatMul(TG4,T45, evaluate = True)

TG6 = sym.MatMul(TG5,T56, evaluate = True)

"""#Populating the parametric Jacobian matrix"""

J = Matrix( sym.symarray('a', (6,6)) )

J[-3:,0] = TG1[:-1,2]
J[-3:,1] = TG2[:-1,2]
J[-3:,2] = TG3[:-1,2]
J[-3:,3] = TG4[:-1,2]
J[-3:,4] = TG5[:-1,2]
J[-3:,5] = TG6[:-1,2]

J[0:3,0] = TG6[:-1,3].diff(theta1)
J[0:3,1] = TG6[:-1,3].diff(theta2)
J[0:3,2] = TG6[:-1,3].diff(theta3)
J[0:3,3] = TG6[:-1,3].diff(theta4)
J[0:3,4] = TG6[:-1,3].diff(theta5)
J[0:3,5] = TG6[:-1,3].diff(d)

# J = J.subs([(d1, 0.333),(d3, 0.3160),(d5,0.3840),(d7,0.2070),(a3,0.088)])

print("Printing parametric Jacobian:\n")
J

"""#Quantizing the duration of motion in n time steps, and finding velocity of e

steps = 200
t = np.linspace(0,20,steps)
t_step = 20/steps

z_init = 0.773
x_init = 1.347
```

```
z_dot = 0.01

vel_origin = np.zeros((6,steps))
vel_origin[0,:] = 0
vel_origin[1,:] = 0
vel_origin[2,:] = z_dot

thetas = np.array([0,sym.pi/2,sym.pi/2,sym.pi,0,0])
J_init = J.subs([(theta1, thetas[0]), (theta2, thetas[1]), (theta3, thetas[2]),
J_init

thetas = np.array([0,sym.pi/2,sym.pi/2,sym.pi,0,0])

j_angles = []
EE_positions = []

EE_base = TG6

for i in range(steps):
  J_step = J.subs([(theta1, thetas[0]), (theta2, thetas[1]), (theta3, thetas[2])
  J_step = np.array(J_step).astype(np.float64) + 0.00001
  joint_vel = np.linalg.inv(J_step) @ vel_origin[:,i]

  new_thetas = thetas + t_step*joint_vel

  # print(joint_vel.shape)

  j_angles.append(new_thetas)

  EE_base_temp = EE_base.subs([(theta1, new_thetas[0]), (theta2, new_thetas[1]),

  # print(EE_base_temp)

  EE_base_temp = np.array(EE_base_temp).astype(np.float64)
  EE_position = EE_base_temp[:-1,3]

  EE_positions.append(EE_position)

  thetas = new_thetas

  print("Completed = " + str(i+1) +"/" + str(steps))

j_params = np.array(j_angles).reshape(steps, 6)

fig, axs = plt.subplots(2, 3, figsize = (20,10))
axs[0, 0].plot(t, j_params[:,0]*0)
axs[0, 0].set_title('theta_1')
axs[0, 1].plot(t, j_params[:,1], 'tab:orange')
axs[0, 1].set_title('theta_2')
axs[0, 2].plot(t, j_params[:,2], 'tab:red')
axs[0, 2].set_title('theta_3')
axs[1, 0].plot(t, j_params[:,3], 'tab:green')
axs[1, 0].set_title('theta_4')
axs[1, 1].plot(t, j_params[:,4], 'tab:cyan')
axs[1, 1].set_title('theta_5')
axs[1, 2].plot(t, j_params[:,5], 'tab:pink')
axs[1, 2].set_title('d')

for ax in axs.flat:
    ax.set(xlabel='time in (s)', ylabel='Joint parameter values')

import plotly.graph_objs as go

X = np.array(EE_positions)

trace = go.Scatter3d(
   x = X[:,0]*0, y = X[:,1]*0, z = X[:,2],mode = 'markers', marker = dict(
      size = 1,
      color = 'red', # set color to an array/list of desired values
      colorscale = 'reds'
```

```
        )
    )

fig = go.Figure(data = [trace])

fig.update_layout(
    scene = dict(
            xaxis = dict(nticks=10, range=[-0.7,0.7], title = "x axis in metres")
                    yaxis = dict(nticks=10, range=[-0.45,0.45], title = "y axis
                    zaxis = dict(nticks=10, range=[0,1.5], title = "z axis in me
    width=1000,
    height=1000,
    margin=dict(r=20, l=10, b=10, t=10))
fig.show()
```