Project Part 2– ME4D002

Motion planning of 4 DOF serial manipulator for spray painting applications

Project Report

Submitted By

Name : Saharsh J

Roll No : 18ME01007

Under the guidance of

Dr. V. Pandu Ranga

Associate Professor

School of Mechanical Sciences

IIT Bhubaneswar

School of Mechanical Sciences

Indian Institute of Technology, Bhubaneswar

May, 2022

Indian Institute of Technology Bhubaneswar

School of Mechanical Sciences

Argul, Jatni, Bhubaneswar – 752050

Certificate

This is to certify that the end-semester report on

Motion planning of 4 DOF serial manipulator for spray painting applications

submitted by

Saharsh J

is a record of bonafide work for end-semester evaluation carried out by him, towards the completion of the Project Part-2 course, in the partial fulfilment of the requirement for the award of BTech degree in Mechanical Engineering stream, School of Mechanical Sciences, Indian Institute of Technology Bhubaneswar. This work is carried out during the academic year 2021-22, under my guidance.

Date: 12/05/2022

(Dr. V. Panduranga)

Project Supervisor

(Examiner 1)	**(Examiner 2)**	**(Examiner 3)**	**(Examiner 4)**

Acknowledgements:

Apart from the efforts of myself, the success of the project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in helping me with this project. First and foremost, I would like to thank my supervisor of this project, Dr. V. Pandu Ranga, IIT Bhubaneswar for the valuable guidance and advice during the complete semester. Besides, I would also like to thank PhD scholar Mr. Dev Singh Chouhan who helped me whenever I required help. Though the project was in online mode until now, it gave me many good experiences of research on this topic and I gained a good amount of knowledge in this topic. I would also like to thank Indian Institute of Technology Bhubaneswar for providing me with this wonderful opportunity to do my B.Tech project. Finally, an honourable mention goes to my family for their understanding and support on me in completing this project.

Abstract:

Nowadays, in industries, robots are widely used for various tasks. One such task is spray painting, a technique where the spray-painting gun sprays the paint through the air onto the surface. The spray-painting process is widely used in automobile industries. The main requirement of the spray-painting robot is that when humans are exposed to the chemicals, it has a significant effect on their health. Moreover, it is a highly skillful process where skilled labours are required which is not easy to get.

In this project, as the 4 DOF manipulator is already available in our lab, a base of the spray-painting apparatus and along with the model of the manipulator present are to be designed and required solid modelling and analysis of the serial manipulator will be done using SolidWorks. Calculations will be done in order to obtain the position and orientation of the end effector, i.e., in this case, the spray-painting gun. The position and orientation of the end effector will be done using forward kinematics, and the angles of each joint for a required position of the end effector will be calculated using inverse kinematics. Denavit–Hartenberg (D-H) methods are preferred in this case of the forward and inverse kinematics approach. In addition to this, motion planning, i.e., trajectory planning of the serial manipulator, will be done in order to obtain different spray-painting patterns. *Motion planning* is a computational problem that is used to find the path that should be taken in order to obtain the required movement of the end effector. MATLAB is used for the motion planning of the manipulator. MATLAB will also be used for calculation purposes since the calculations of the matrices are expected to be complicated. These are the main objectives of this project. ROS will be used for the interaction between the hardware and the software part in order to give information for trajectory planning to the hardware.

Keywords used:

Spray painting robot, Forward kinematics, Inverse kinematics, Jacobian, Trajectory planning

Abbreviations used:

(i) DOF – Degrees of Freedom

(ii) CAD – Computer-Aided Design

(iii) DH – Denavit Hartenberg

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1:

INTRODUCTION

1.1 Aim

The main objective of this project is to design and build a 4 DOF serial manipulator, its base and a mounter where the spray-painting apparatus can be mounted and to do the motion planning which includes forward and inverse kinematics, Jacobian analysis, trajectory planning using MATLAB and ROS. The hardware for the same is to be built and all the motion planning has to be implemented on the same using ROS.

1.2 Introduction

Today, different kinds of robots are being used in many fields. Especially in industries, robots are essential. One application in the industries is the spray-painting task which is no more suitable for human workers because it has a significant effect on health due to the presence of various chemicals. Also, spray painting is a challenging task and need significant skill. The spray-painting process is widely used in automobile industries. The main aim of this project is to build a 4 DOF serial manipulator and to do the motion planning for the manipulator.

The spray-painting process needs to be automated since it can affect the health of people working on it in different ways. Toxic fumes and mist are emitted into the environment. Due to this, the possibility of obtaining cancer increases for a person working at this place. Noise pollution created by the spray painting guns can cause a lot of hearing problems, and finally, the spray paints used are generally flammable. Hence, a fire might arise during the combination of paint and air.

The 4 DOF manipulator was chosen to be used for various reasons. Firstly, if we use a manipulator with fewer degrees of freedom, then we will not have the flexibility/ dexterity to manoeuvre in the workspace as the manipulator will be limited to move only to specific spaces even within a workspace. Secondly, if we are to use a higher degree of freedom manipulator, though the dexterity of the manipulator will be high, the computations involved in solving the kinematics and Jacobian will be very high and powerful computers might be required. Hence, we need to strike a balance between the two. Hence, the optimized solution was to use four degrees of freedom manipulator as it provides a decent amount of dexterity as well as it is not that computationally complex.

CHAPTER 2:

LITERATURE REVIEW

2.1 Literature Review

Initially, the literature review is done on various research articles which describes various structures and simulations of the spray-painting robot. Firstly, (Shabeeb, n.d)[1]. had proposed a geometry-based spray paint trajectory generation system for ruled surfaces which are extremely useful in modern manufacturing systems. It was used to simulate the spray painting process using an articulated arm robot. A ruled surface is one that is a polygon mesh built between two defined boundaries. Firstly, the kinematics model for the articulated armed robot was designed by determining the solutions of forward and inverse kinematics. So, initially, a coordinate system was assumed, and then the transformation matrix was determined. Then, the inverse kinematic model was derived. Secondly, (Prakash Gujela et al., 2007)[2] presented the analysis of a spray painting robot. Mathematical analysis were done to find the position and orientation of the end effector. Also, the spray painting patch and angles of each joint were calculated in this analysis. Position and orientation of end effector were analyzed by forward kinematics. The angles of each joint were found out by inverse kinematics. Denavit- Hartenberg (D-H) methods were used in forward and inverse kinematics. Spray painting patch was generated by finding the equation of the surface of regular shape work-piece. Measurement and observation of robots were made on SolidWorks software. Also, in this analysis, the calculation is quite complex and contains many variables, even in an element of a matrix. Such kind of long equations was simplified by using MATLAB software.

Consequently, (Madhuraghava et al., n.d.)[3] presented the structural analysis of a six degree of freedom robot spray coating manipulator using NX-CAD and ANSYS software. A robot can carry more weight than dedicated equipment or humans, thus allowing them to carry paint-delivery equipment, such as colour changing valves, on their arm and close to the applicator. In this analysis, the robot manipulator was analyzed. Analysis was made to get the position and orientation of the manipulator. Also, the spray coating manipulator strength and environment conditions were analyzed with various loading conditions and the position and orientation of the end-effector were analyzed by forward kinematics. Measurement and observation of robot manipulators were made in the above-mentioned software. Firstly, the CAD model of the manipulator was designed, and various analysis were done to improve its structure. The analysis included DOF in y and z direction, Stress in XY direction, Stress in XZ direction, stress in X and Z direction, von mises stress in the Y direction, etc., were analyzed. Then, a paper related to trajectory planning strategy by (Trigatti et al., 2018)[4] demonstrated a method to find the spray-painting arm's trajectory for the industrial robot. The main goal of the work was to provide an effective and easy implementation with the method, whose field of application is widened by choice of avoiding the use of numerical optimization routines. Starting from a parametrization of the end-effector path in the operative space red that ensures an accurate paint coverage, the velocity profile red of the spray gun was first defined by the algorithm to ensure the limitation of the tangential end-effector velocity red to enhance paint thickness uniformity. In a later stage, a sequence of filtering operations was performed in order to bound the joint accelerations and to improve smoothness.

Spray painting gun is one of the most essential parts of the spray-painting robot. Hence, it is necessary to do some background work on the spray gun. There are two types of spray guns that can be used to do spray painting. They are single nozzle spray gun and double nozzle spray gun. (Y. A. Wang et al., 2020)[5] developed a double nozzle air spray gun in view of the low efficiency of single-nozzle air spray guns when spraying large targets. A new double-nozzle air spray gun structure was designed in this paper based on the Coanda effect of double jets. Firstly, a 3-D physical model of the double-nozzle air spray gun was developed in SolidWorks, in which unstructured grids were generated. Secondly, the spray-painting process was numerically modelled with the help of the computational fluid dynamics (CFD) software ANSYS-Fluent 16.0. The interference flow field was studied numerically with the help of ANSYS. It was seen that a negative pressure was developed in the region between the two jets because of the entrainment of jets, which resulted in two jets attracting each other and finally, both the jets combined into one jet. The geometry of the air spray gun was also crucial as its influence on the interference spray flow field were also investigated by changing the distance between the centres(L) of paint holes and the angle between the axes of the paint holes(θ). The analysis showed that with small values of L and θ, the interference effect between the two jets is very strong, while when the concentration of the paint in the central region is more, it is easier for the overspray to occur. If L and θ are very large, then the interference effect becomes weak and more paint goes onto the side regions than the central region, which makes the coating film uneven.

The important part of the project is to do the motion planning and trajectory planning for the four degrees of freedom spray painting serial manipulator. Hence, a literature review on these parts is extremely necessary to know about the current methods and procedures which are being followed to obtain the trajectory for serial manipulators. (Patel et al., 2020)[6] developed a deep learning-based robot control using neural networks. It is known that nonlinearity, external disturbances are some of the significant challenges in robotics. To overcome these issues, robust adaptive control is required. Here, deep learning was used to build the inverse dynamics of a manipulator. They developed a robust adaptive motion control by effectively combining existing adaptive sliding mode controller (ASMC) with Recurrent Neural networks such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). A supervised learning approach was used to train LSTM and GRU models. These are the ones that replaced the inverse dynamic model.

(Lees & Chirikjian', 1996)[7] presented a combinatorial method to compute the inverse kinematics of a binary manipulator, which helps in reducing the search space to a reasonable/manageable size and also creates smooth motions that follow a specified trajectory. Binary manipulators are the ones that are powered by actuators that have only two stable states. When compared to continuous actuators, binary manipulators perform worse but is inexpensive. But the number of states in binary manipulator increases exponentially as the number of actuators increases.

(Liu et al., 2021)[8] presented an automatic spray motion planning for a shotcrete manipulator. These types of manipulators are widely used in the construction of railways, coal mines, tunnels, etc., and there are many problems such as high operation risk, injuries, etc. In order to reduce these, the author has proposed a spraying motion planning method for an 8 DOF manipulator arm. On the basis of kinematics modelling and analysis of the shotcrete manipulator, method design and motion planning of shotcrete were carried out according to tunnel conditions, and verification experiments were carried out in the actual tunnel.

(Macfarlane & Croft, 2003)[9] developed an online method for obtaining smooth and jerk bounded trajectories. The procedure used here involves the concatenation of fifth-order polynomials to provide a smooth trajectory between two-way points. The trajectory approximates a linear segment with parabolic blends trajectory. A sine wave template was used to calculate the end conditions for ramps from zero acceleration to nonzero acceleration. Joining these control points with quintic polynomials results in a controlled quintic trajectory that does not oscillate and is near time optimal for the jerk and acceleration limits specified. The method requires only the computation of the quintic control points, up to a maximum of eight points per trajectory waypoint. This provides hard bounds for online motion algorithm computation time. A method for blending these straight-line trajectories over a series of waypoints was also discussed.

(Zhang et al., 2020)[10] presented a continuous trajectory planning methodology for serial manipulators based on non-convex global optimization. Firstly, this model is constructed to balance motion rapidity and safety. Secondly, a model transformation method for a non-convex optimization model is done. By this, the accurate global solution was obtained with an iterative solver starting from arbitrary initializations, which significantly improved the computational accuracy and efficiency. Other than this, an efficient initialization method for the iterative solver based on multivariable-multiple regression was also presented, which further fastened up the solution process.

(J. Wang & Lei, 2018)[11] had proposed and tested an online trajectory planning method based on kinematical optimization. The particle swarm optimization (PSO)-based trajectory planner of the polynomial interpolation method optimized the minimum time and acceleration for velocity constraint.

(Arunadevi et al., 2021)[12]'s primary objective of robot manipulators was to achieve the desired orientation and point of the end effector in order to accomplish the pre-established task. To achieve a successful, optimized solution to model and operate the arm, the pioneer two robots made use of the inverse kinematics analysis. An economical, lightweight, and compact five degrees of freedom Pioneer2 manipulator is considered in this paper. We know that as the degrees of freedom of the robot increases, the complexity of the inverse kinematic also increases and we also obtain many different solutions for the same endpoint and this makes the problem more critical. In order to overcome this issue, the authors had performed intelligent computation of inverse kinematics with the help of three different machine learning ideas that includes an artificial neural network, K-nearest neighbour algorithm, and also linear regression. It was observed that among the three different algorithms used, K-nearest neighbour algorithm has the least root mean square error and R-square values the least. Hence, using this algorithm, the inverse kinematics was obtained at a fast pace and also with a great amount of accuracy. In addition to this, a cubic spline interpolation was used to obtain a smooth trajectory in this case.

In designing and using robots in the field of task execution, planning robot pose trajectory plays a very key role in creating effective robots. (Zha, n.d.)[13] proposed a unified approach for optimizing pose trajectory planning for manipulator robots by enhancing the optimization of pose ruled surfaces using a genetic algorithm (GA). The robot pose ruled surface concept is used to interpolate the end-effector positions and orientations. In order to plan robot pose trajectories, kinematics, dynamics, and control performances are used to generate and optimize robot pose ruled surfaces. Based on functional analysis and dynamics planning, the optimization model is instantiated by determining trajectory position and orientation based on high-order parametric space curves. In order to optimize the kinematic and dynamic performance of the pose-ruled surface, an enhanced optimization method is used to choose the parameters (coefficients) of the space curves. Optimal paths are achievable and robust in terms of control when incorporating the robot's dynamics constraints.

2.2 Learnings and approach from literature review

From various papers related to motion planning, it was observed that motion planning not only includes the decision of path to be taken by the manipulator, but the entire process starting from Designing a manipulator, D-H parameterisation, Forward Kinematics, Inverse Kinematics, then calculating the Jacobian for differential motion analysis, singularity analysis, the decision of sensors and actuators, path planning and then finally trajectory planning. These are the steps involved in the motion planning, and each step is very crucial in order to understand the manipulator. One more thing that should be considered while solving the inverse kinematics is that multiple solutions might arise for a given final position. This is due to the existence of trigonometric functions in the equations. Here we need to be careful in choosing which angle value must be considered by analysing the environment. See Fig 1 for what the motion planning includes and the order in which things need to be done.

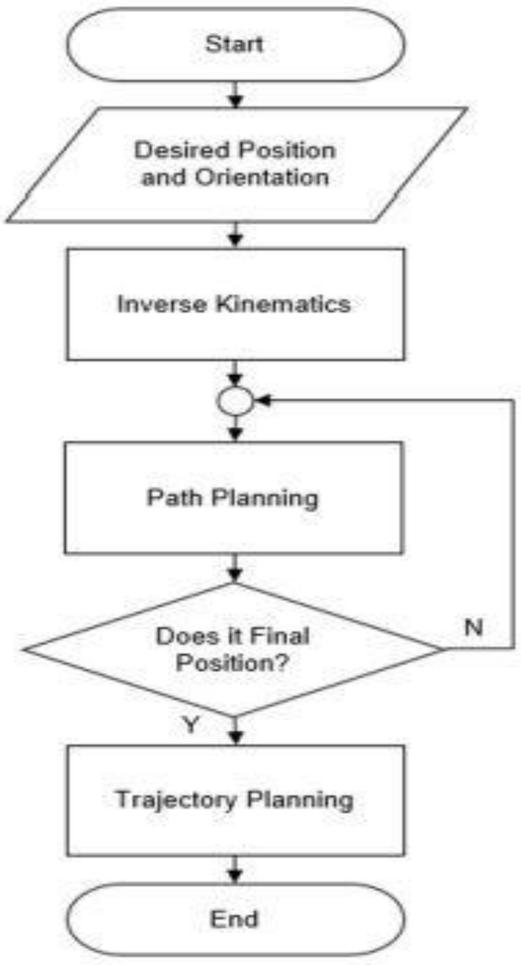


Fig 1. Flow chart of Motion planning for a manipulator

So, after going through various resources we can finalize that the steps needed inorder to complete this project includes starting from Designing a manipulator, D-H parameterisation, Forward Kinematics, Inverse Kinematics, then calculating the Jacobian for differential motion analysis, singularity analysis, the decision of sensors and actuators, path planning and then finally trajectory planning.

CHAPTER 3:

DESIGN OF THE SPRAY-PAINTING MANIPULATOR

3.1 Initial Design

Initially, a four DOF manipulator was made (see Fig 2). Few design considerations were made while building the shown manipulator:

1. About the minimisation of the material used. This is important since this directly affects the cost of the manipulator.
2. The centre of mass has to be taken care of, and the weight is distributed such that in a fully extended state, the centre of mass does not go far from the base of the manipulator.
3. The designed structure can be easily manufactured using the 3D printing technique since it does not have any complex parts, which might make manufacturing difficult.

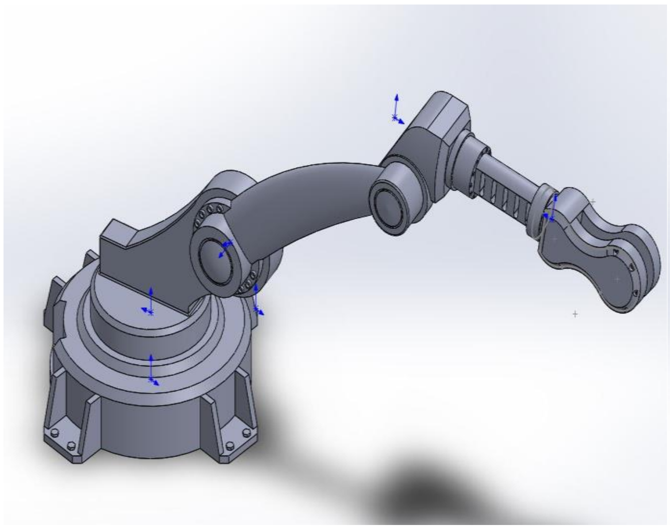


Fig 2. Concept design of the 4 DOF manipulator

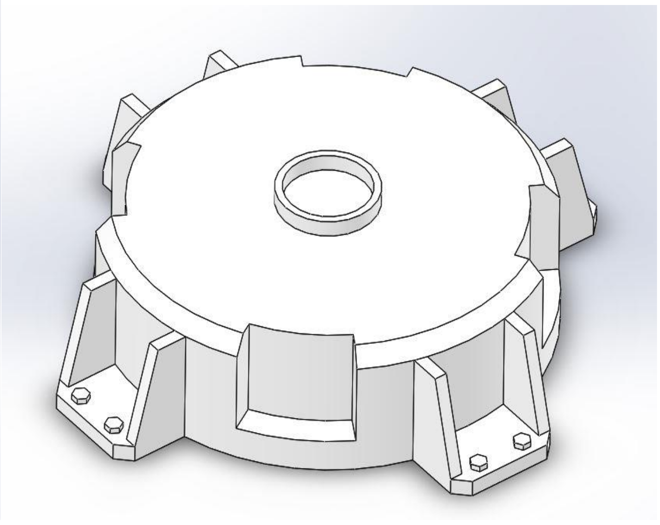


Fig 3. Base of the 4 DOF manipulator

A special look into the base of the manipulator (see Fig 3): It is the heaviest part of the whole setup. As previously said, this is done to make sure that the centre of mass lies nearby to the centre of the base.

The figure shows the 4 DOF manipulator (see Fig 4) designed based on the manipulator present in the AIM Lab. The model's dimensions are to scale and has been designed to import the model to MATLAB using Simscape multibody add-in, which will be helpful in further analysis. It contains a sturdy base that supports four links in which the 1st link is connected.

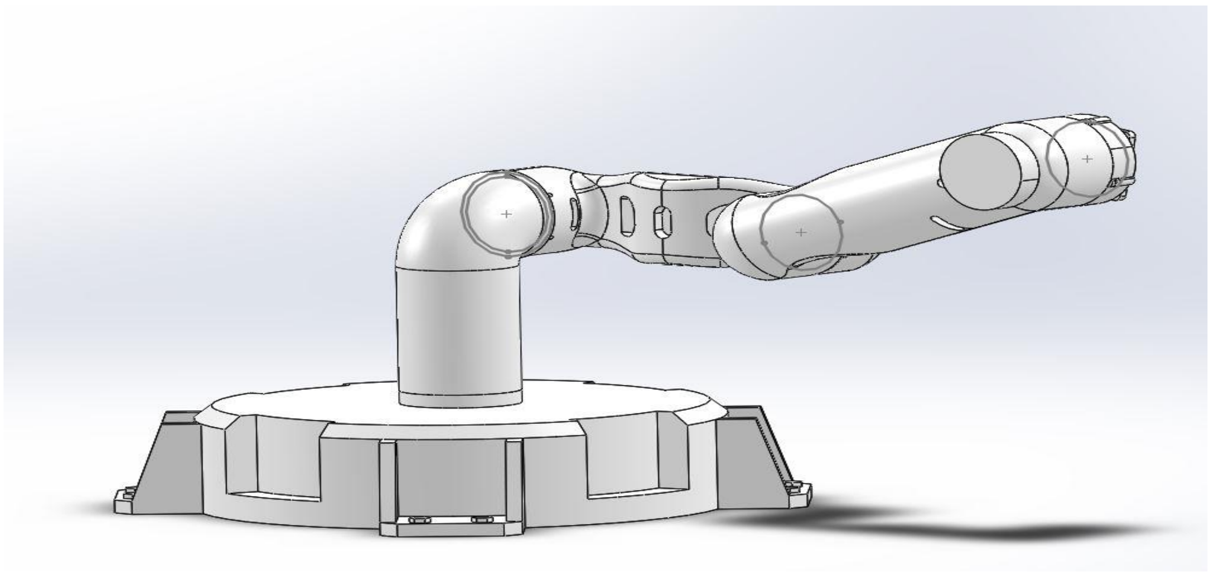
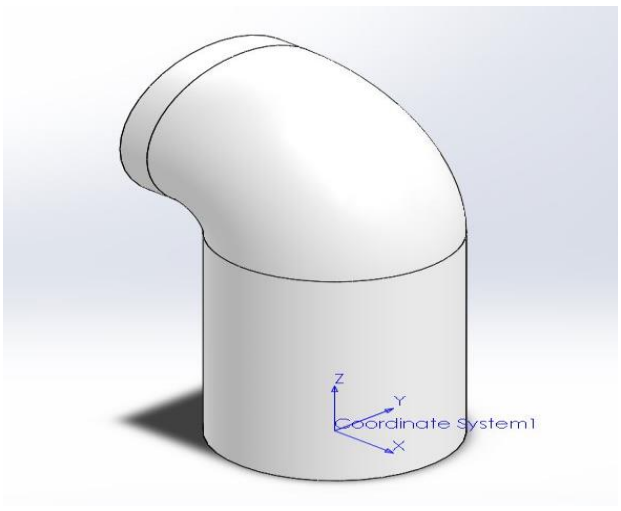


Fig 4. 4 DOF Manipulator present in the Lab with specially designed base

The figures of different links of the manipulator are shown in the following images. (see Fig 5,6 and 7).



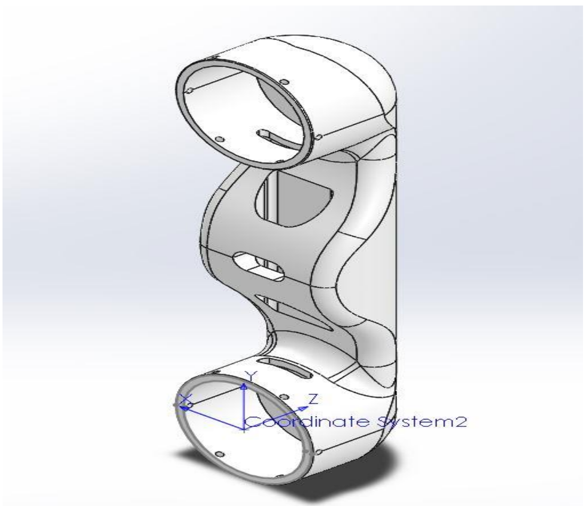


Fig 5. Link 1 of the Manipulator

Fig 6. Link 2 of the Manipulator

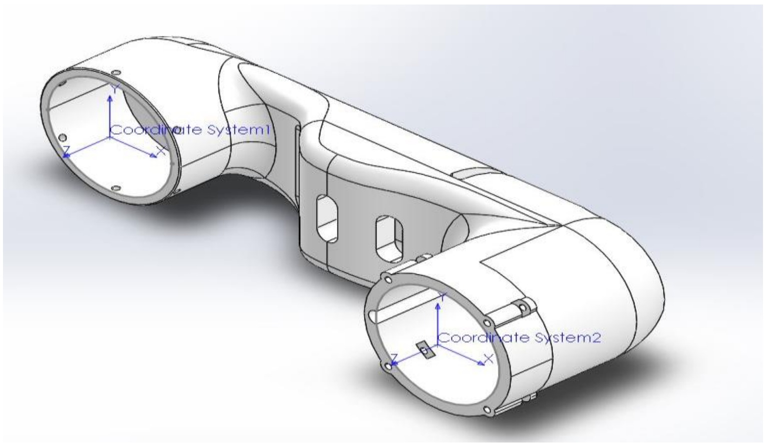


Fig 7. Link 3 of the Manipulator

3.2 Finalized Design of the Manipulator

The initial model that created was similar to that of the model of the manipulator available in the AIM lab. To do simulations project specific, a different type of end effector was designed at a later stage which is specific to the spray painting device. The model of the manipulator with the spray-painting end effector is shown in fig 49

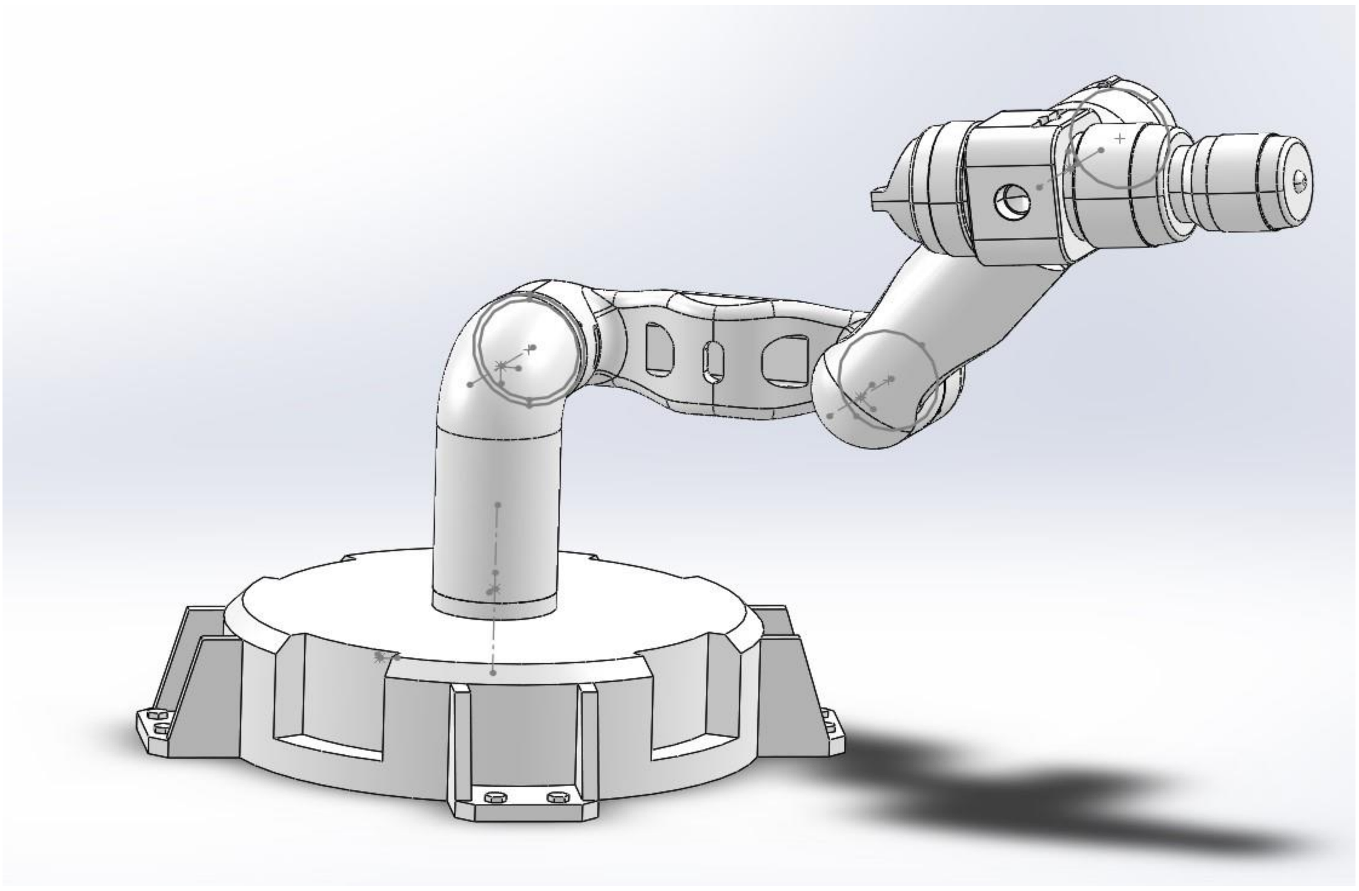


Fig 8. Finalized model of the Spray-Painting manipulator

CHAPTER 4:

ANALYSIS OF THE MANIPULATOR

4.1 Kinematic Analysis

D-H notations define the four joint link parameters and are very important to obtain the transformation matrix which is in turn required to do forward kinematics. With respect to frame (i-1) and frame (i), the four DH-parameters - two link parameters (a, alpha) and two joint parameters (d, theta) (see Fig 8)- are defined as:

(a) Link Length (ai) - distance measured along x-axis from the point of intersection of x-axis with z_I-1-axis to the origin of frame {i}.

(b) Link twist (αi) angle between zi-1 and zi -axes measured about xi-axis

(c) Joint distance (di) - distance measured along zi-1-axis from the origin of and in the right-hand sense. frame (i-1) to the intersection of x- axis with zi-1 -axis.

(d) Joint angle (θ) angle between xi-1 and xi-axes measured about the zi-1-axis in the right-hand sense.

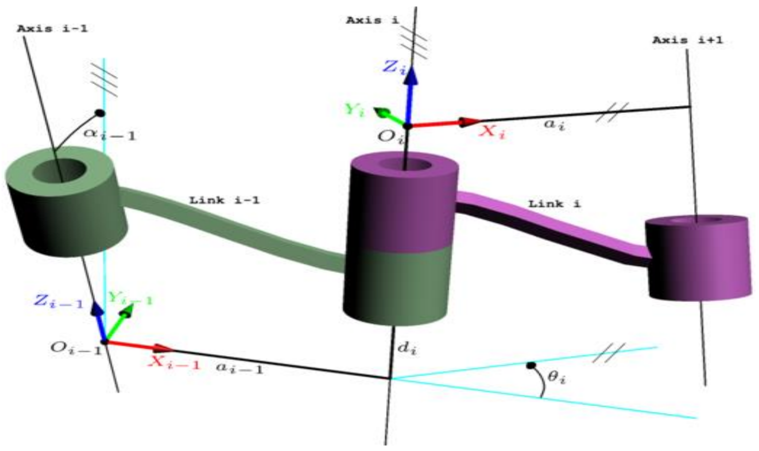


Fig 9. Description of Joint-Link parameters

4.1.1 Forward Kinematics

The manipulator consists of three revolute joints. Calculating the forward kinematics will be complex when the degrees of freedom goes beyond a limit. Hence, since the fourth degree of freedom is simply for orienting the spray-painting gun, we can consider that the manipulator is a 3 DOF system and then calculate the forward kinematics normally. However, in order to know the effect of four DOF, A MATLAB code to find the final position and orientation was done. The transformation matrix obtained, and for different input values of theta, we will be able to obtain the final position of the end effector, in this case, the spray-painting gun. Hence, we can see that the forward kinematics helps us in knowing the final position when we give different values of input to angles of the revolute joint. This is the frames assumption that was considered to obtain the transformation matrix. The z-axis is always oriented in the perpendicular direction to the plane in which the link is rotated. And x-axis is always perpendicular to the plane of the z-axis, and the remaining direction gives the y axis. Using this frame assignment, we can obtain the D-H parameters as shown in table 1 for links, and I have obtained the transformation matrix from this.

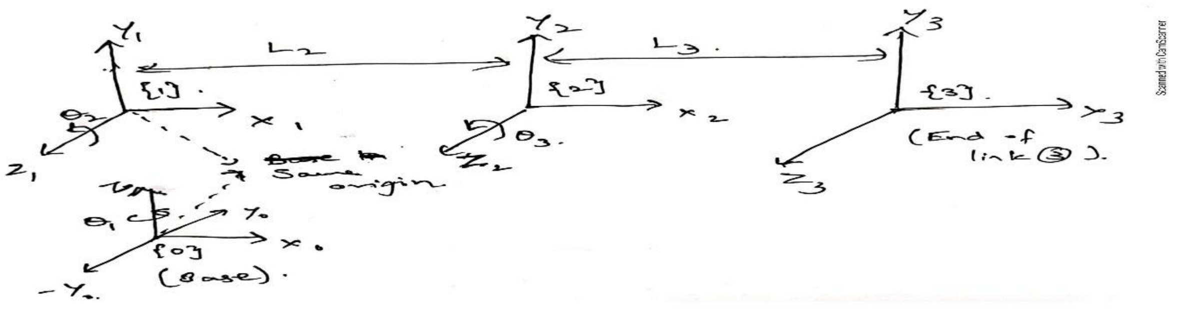


Fig 10. Assigning of frames for the arm

Link i	a_i	α_i	d_i	θ_i	q_i
1	0	90	0	θ_1	θ_1
2	a_2	0	0	θ_2	θ_2
3	a_3	0	0	θ_3	θ_3

Table 2. D-H parameters considered for the above transformation solution case

By multiplying the individual transformation matrices, we obtain the overall transformation matrix as follows:

$$ {}^0_3T = \begin{bmatrix} \cos\theta_1\cos(\theta_2+\theta_3) & \cos\theta_1\sin(\theta_2+\theta_3) & \sin\theta_1 & \cos\theta_1(a_3\cos(\theta_2+\theta_3)+a_2\cos\theta_2) \\ \sin\theta_1\cos(\theta_2+\theta_3) & -\sin\theta_1\sin(\theta_2+\theta_3) & -\cos\theta_1 & \sin\theta_1(a_3\cos(\theta_2+\theta_3)+a_2\cos\theta_2) \\ \sin(\theta_2+\theta_3) & \cos(\theta_2+\theta_3) & 1 & a_3\sin(\theta_2+\theta_3)+a_2\sin\theta_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1} $$

Extending the same concept for the four DOF manipulators, the transformation matrix has been obtained in MATLAB. The obtained transformation matrix from the code for four DOF manipulators was further used in the calculations whenever required. From the data of the present manipulator in the lab, i.e., length of link 1 = 132 mm, length of link 2 = 213mm, length of link 3 = 192mm and length of link 4 = 0mm. The position and orientation vector obtained at $\theta_1 = \frac{3\pi}{4}$, $\theta_2 = \frac{2\pi}{3}, \theta_3 = \frac{3\pi}{2}, \theta_4 = 0$, for instance was as follows in equation 4.2 and 4.3:

$$Position\ of\ the\ end\ effector = \begin{bmatrix} -135.6067 \\ 135.6067 \\ 280.4634 \end{bmatrix} \tag{4.2}$$

$$Orientation\ of\ the\ end\ effector = \begin{bmatrix} -0.6124 & 0.3536 & 0.7071 \\ 0.6124 & -0.3536 & 0.7071 \\ 0.5 & 0.86 & 0 \end{bmatrix} \tag{4.3}$$

4.1.2 Inverse Kinematics

There are two forms of inverse kinematics solutions. The first one is the closed-form solution which is nothing but the solutions obtained from directly solving the equations in the matrices. This is the case where the final position of the end effector is known and what we need to do is to find the angles of the links such that the end effector is able to reach the given point in the workspace. For the case shown above, the closed-form solutions for all the values of theta are as follows. This is which is done manually was also cross verified by using MATLAB, which I will be showing at the end. In the next step, I would like to concentrate on numerical solutions in order to obtain inverse kinematics.

For a given general final position and orientation matrix of three DOF manipulator,

$$Final\ Pos\ matrix\ = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

Solving the above final position matrix by equating it to the matrix obtained by solving the forward kinematics, we end up obtaining the following for different joint angles using which we can achieve the desired position of the end effector.

$$\theta_1 = \tan^{-1}(\frac{r_{24}}{r_{14}}) \tag{4.5}$$

$$\theta_2 = \tan^{-1}((-a_3 r_{31} + r_{34})/\pm\sqrt{(-a_3 r_{11} + r_{14})^2 + (-a_3 r_{21} + r_{24})^2} \tag{4.6}$$

$$\theta_3 = \tan^{-1}\left(\frac{r_{31}}{r_{32}}\right) - \theta_2 \tag{4.7}$$

The solutions of four DOF inverse kinematics are obtained using inversekinematics() function which is present in the MATLAB as a predefined function in Robotics System Toolbox. This function is used in the further trajectory planning analysis.

Secondly, an alternative method to calculate the inverse kinematics of the manipulator was also looked into. This is because, we would require the calculation of the inverse kinematics at a much faster pace since as the degrees of freedom increases, the time taken for the computation of the inverse kinematics using MATLAB will significantly increase. This was achieved by using a regression model. We know that the forward kinematics of the manipulator can always be easily determined using MATLAB. Now, using the forward kinematics equations, a dataset of 10,000 randomly generated positions and the joint angles were taken as shown in Fig 11.

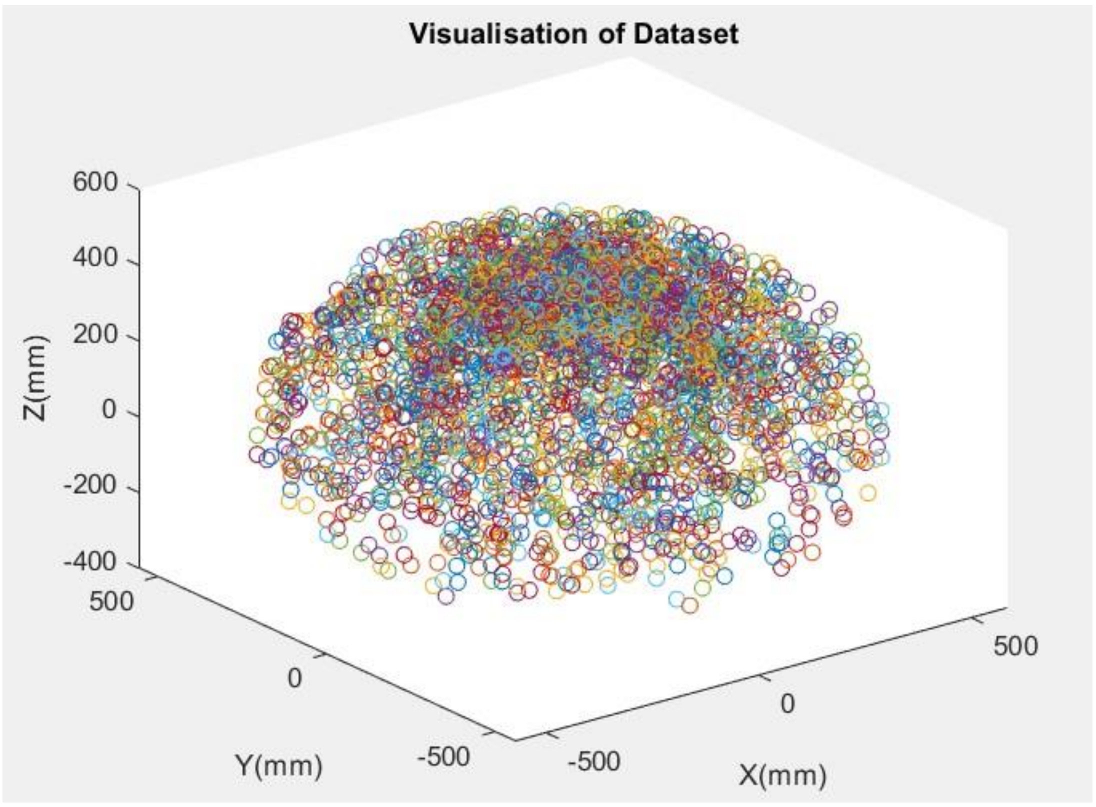


Fig 11. Dataset Visualisation of end effector positions

This dataset was used for the regression in order to obtain the required model. The mean absolute error obtained in this case was around 0.95167 which is pretty accurate.

4.2 Differential Motion and Jacobian Analysis

Jacobian matrix is similar to that of the transformation matrix. Transformation matrix is used to find the end effector's position when the position/angles of all the other links are provided. Whereas, Jacobian matrix is used to find the end effector's velocity when the angular and linear velocities of all the individual links are provided.

$$V(t) = J(q).\dot{q} \tag{4.8}$$

Here,

J(q) is called the Jacobian Matrix and the order of the matrix is 6*n

V(t) is called the velocity vector of the end effector and is of the order 6*1

$\dot{q}$ is called the velocity vector of n joints and is of the order n*1

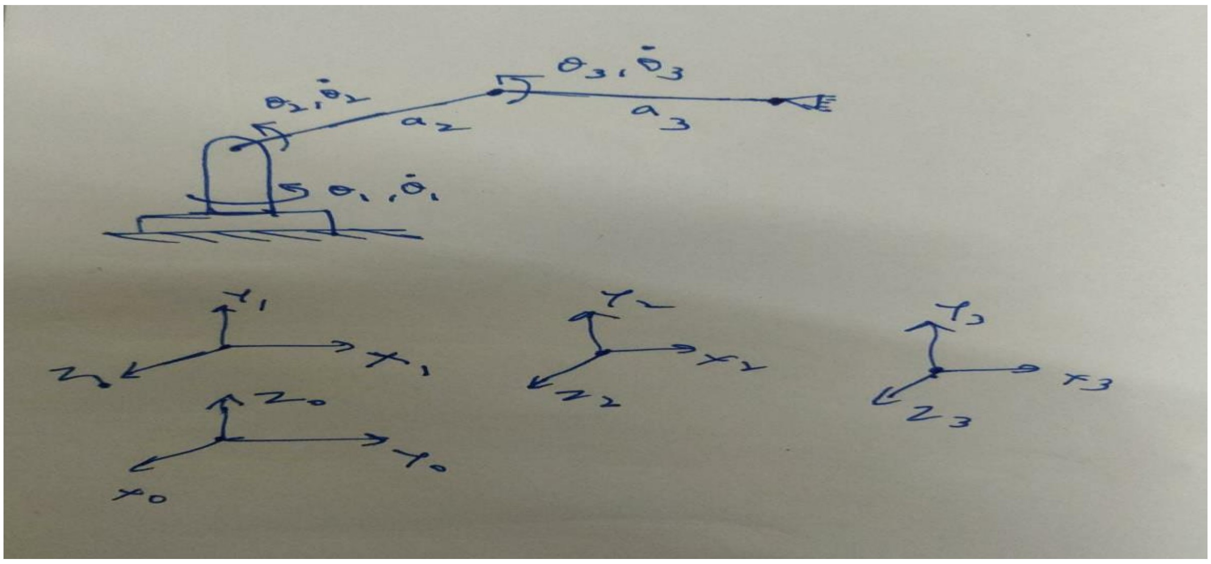
n is the number of links present in the manipulator

Here, the i'th column of the Jacobian matrix is defined as

$$J_i(q) = \begin{bmatrix} J_{vi} \\ J_{\omega i} \end{bmatrix} = \begin{cases} \begin{bmatrix} P_{i-1} \\ 0 \end{bmatrix} & for\ prismatic\ joint \\ \begin{bmatrix} P_{i-1} * P_n^{i-1} \\ P_{i-1} \end{bmatrix} & for\ revolute\ joint \end{cases} \tag{4.9}$$

Here, $$P_n^{i-1} = T_n^0 * O_n - T_{i-1}^0 * O_n \ . \tag{4.10}$$

By using the above equations, the Jacobian for the simplified case of the manipulator (three DOF) can be evaluated by hand. This can be considered as a simplified case since the fourth degree of freedom is used for the orientation of the spray-painting gun and hence the end effector's position will be same as that of the simplified case of the manipulator. However, the Jacobian for the four DOF system was calculated using MATLAB and the code has been attached at the end in appendix.



$\theta_3, \dot{\theta}_3$
$\theta_2, \dot{\theta}_2$
a_2
a_3
E
$\theta_1, \dot{\theta}_1$
Y_1
X_1
Z_1
Y_2
X_2
Z_2
Y_3
X_3
Z_3
Z_0
Y_0
X_0

Fig 12. Assigning of coordinate system for the arm

Using the above equations, we can find the Jacobian matrix for the simplified case and the results obtained are as follows:

$$J_1 = \begin{bmatrix} -s_1(a_3c_{23} + a_2c_2) \\ c_1(a_3c_{23} + a_2c_2) \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.11}$$

$$J_2 = \begin{bmatrix} -c_1(a_3s_{23} + a_2s_2) \\ s_1(a_3s_{23} + a_2s_2) \\ a_3c_{23} + a_2c_2 \\ s_1 \\ -c_1 \\ 1 \end{bmatrix} \tag{4.12}$$

$$J_3 = \begin{bmatrix} -c_1a_3s_{23} \\ s_1a_3s_{23} \\ a_3c_{23} \\ s_1 \\ -c_1 \\ 1 \end{bmatrix} \tag{4.13}$$

$$J = [J_1 \quad J_2 \quad J_3] \tag{4.14}$$

Similar calculations was done for the exact same case of four DOF manipulator in MATLAB. From the data of the present manipulator in the lab, i.e., length of link 1 = 132 mm, length of link 2 = 213mm, length of link 3 = 192mm and length of link 4 = 0mm. The combined Jacobian matrix vector obtained at $\theta_1 = \frac{3\pi}{4}$, $\theta_2 = \frac{2\pi}{3}, \theta_3 = \frac{3\pi}{2}, \theta_4 = 0$ for instance was as follows in equation 4.15 :

$$J = \begin{bmatrix} -135.6067 & 198.3176 & 67.8823 & 0 \\ -135.6067 & -198.3176 & -67.8823 & 0 \\ 0 & 59.7769 & 166.2769 & 0 \\ 0 & 0.7071 & 0.7071 & 0.7071 \\ 0 & 0.7071 & 0.7071 & 0.7071 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.15}$$

CHAPTER 5:

TRAJECTORY PLANNING AND ITS SIMULATIONS

The next step after differential motion analysis is Trajectory planning. The spray-painting device, which will be present at the end of link 4, can effectively be considered an end effector. The main aim of this project is to plan the motion, which includes path and trajectory planning for the device as per the surfaces.

Firstly, the path needs to be specified. The path is nothing but a set of points traversed by the end effector between two waypoints. The path only provides the geometrical data of the movement of the end effector. Once it is done, a trajectory has to be created. The trajectory is the one that includes the time information along with the position. If we are to know about the trajectory, then we should be having the information of velocity and acceleration of all the links and end effector along with the path.

There are two ways by which the trajectories can be formulated. Both have their advantages and disadvantages. We have to consider the requirements of the spray-painting manipulator and the difficulties in the employment of the code. Hence, an analysis was done using MATLAB to determine which trajectory formation technique would be adequate for our application.

5.1 Choosing between Task space and Joint space techniques

5.1.1 Task space technique

Let us consider two waypoints between which the trajectory has to be formulated. Now, we generate a path from one waypoint to another by using curves. The curves can be anything like a straight line, circular path, parabolic, etc. Once the path is decided, we take a set of points between the two waypoints. The more the number of points we consider, the more accurate the movement of links will be present. Now, the inverse kinematics at each intermediate point is calculated, and the values of joint angles and the time at which they have to be executed can be stored.

5.1.2 Joint space technique

Similarly, we can consider two waypoints the manipulator has to execute the motion. Now, instead of formulating a path, the inverse kinematics are calculated only in the given two waypoints. Once the inverse kinematics are calculated at those points, we know the joint angles and the position of every link at the given waypoints. With this knowledge, the positions of the links are linearly interpolated from the initial point to the final point. This will create an entirely different trajectory than that of the task space technique.

In order to observe the differences between the two, a MATLAB code was designed such that the differences between the trajectory generation can be clearly seen between the same waypoints. The MATLAB code has been attached at the end of the document in the appendix. The result obtained is as follows. Four different waypoints were given as inputs, and the code was designed as per the explanation above. It can be seen that the task space trajectory was designed linearly, and the inverse kinematics for 80 points was calculated for the whole path. In the joint space trajectory space, we can see that the formed trajectory is of irregular curves. These curves are generated due to linear interpolation of joint angles, which was calculated using inverse kinematics at the waypoints specified.

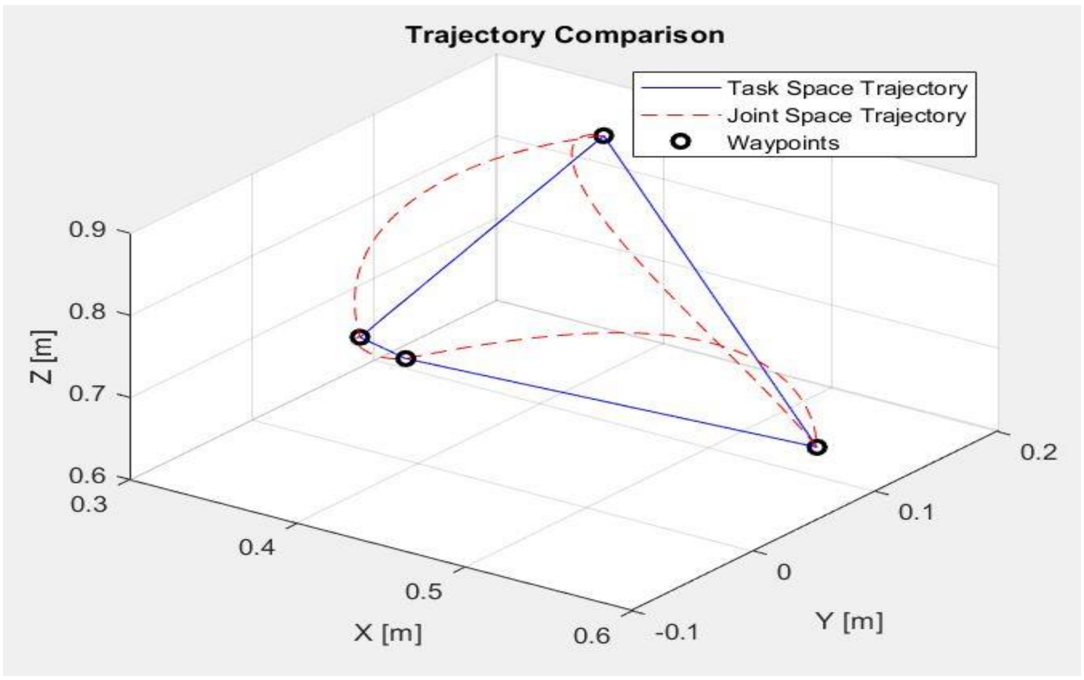
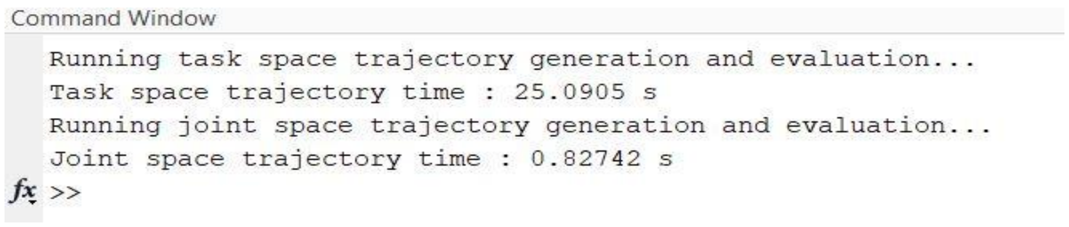


Fig 13. Trajectory generated between waypoints using task space and joint space trajectories

Hence, we can see that in task space, we are doing less work in trajectory generation and more work to make it follow the required trajectory by solving inverse kinematics at each step. This is the major disadvantage of task space trajectory generation. However, the significant advantage is that the path decided is in our control as we generate the trajectory without depending on the manipulator. In the case of Joint space trajectory, since the actuations in the manipulator's joints are linearly controlled with respect to time, the trajectory generated are irregular curves. The major disadvantage, in this case, is that we have no control over the trajectory between the waypoints given. However, the major advantage, in this case, is that the process is significantly less time-consuming. In order to have an idea of how much time it takes to generate, the MATLAB timer function (tic and toc function) was used to calculate the trajectory generation times for the two cases, and the output was obtained as follows:



```
Command Window
  Running task space trajectory generation and evaluation...
  Task space trajectory time : 25.0905 s
  Running joint space trajectory generation and evaluation...
  Joint space trajectory time : 0.82742 s
fx >>
```

Fig 14. Result obtained for time taken for trajectory generation for two cases

From this, it can be seen that the time taken to generate the trajectory varies significantly. Hence it becomes crucial to choose which kind of trajectory is the best for our usage.

The plot of actuators at four joints of the manipulator was also obtained and are as follows:

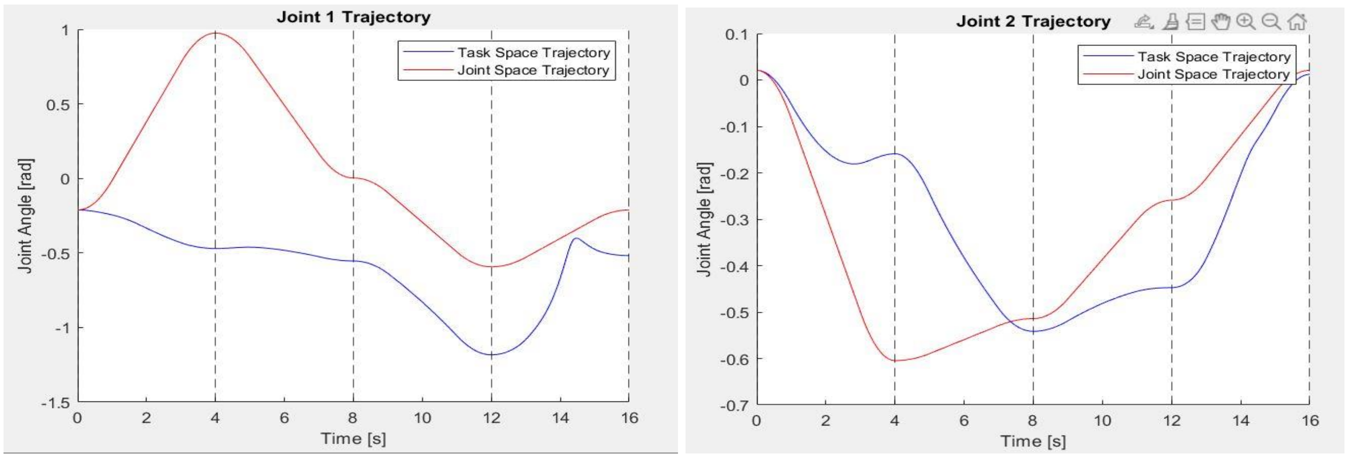


Fig 15. Joint angle vs time plot for joint 1 and joint 2 for task space and joint space trajectory

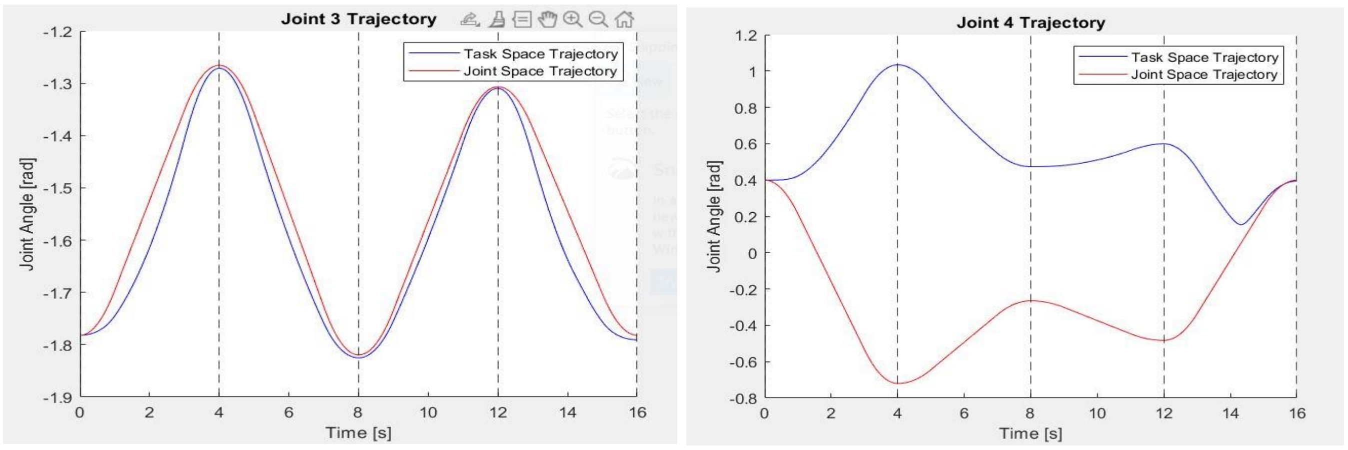


Fig 16. Joint angle vs time plot for joint 3 and joint 4 for task space and joint space trajectory

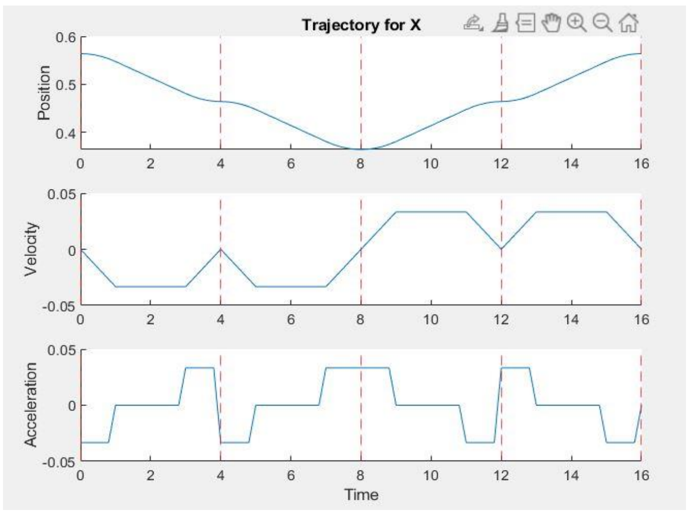
From these plots, it can be seen that the joint space trajectory which in red is generally smoother than the task space trajectory which is shown in blue. This can be seen from the slopes of the curves. This leads to more actuator motion in task space trajectory.

5.2 Different types of Trajectories

Regardless of choosing task space or joint space trajectories, there are mainly three ways of deciding the interpolating trajectories. They are trapezoidal, cubic and quintic trajectories. Each of them was thoroughly analyzed both in task space and joint space techniques.

5.2.1 Trapezoidal Trajectory

Trapezoidal trajectories are the ones which have linearly ramped velocity profile and the velocity profile kind of resembles the trapezoidal shape. Position varies quadratically during constant acceleration and deceleration and the trajectory plot obtained in the case of both task space and joint space technique for trapezoidal trajectory are shown below:



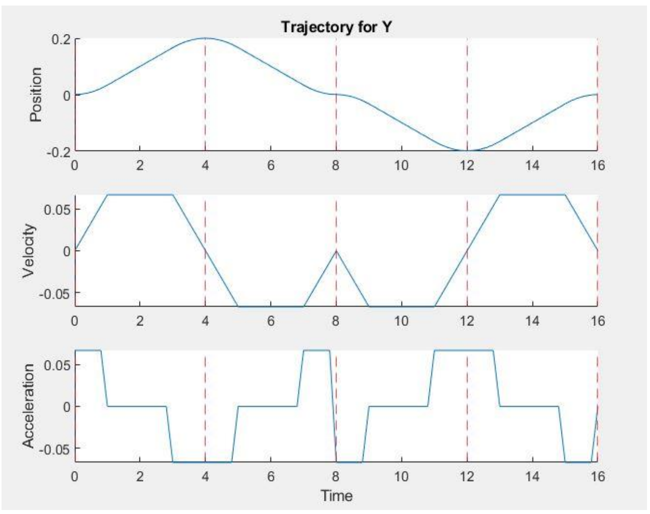
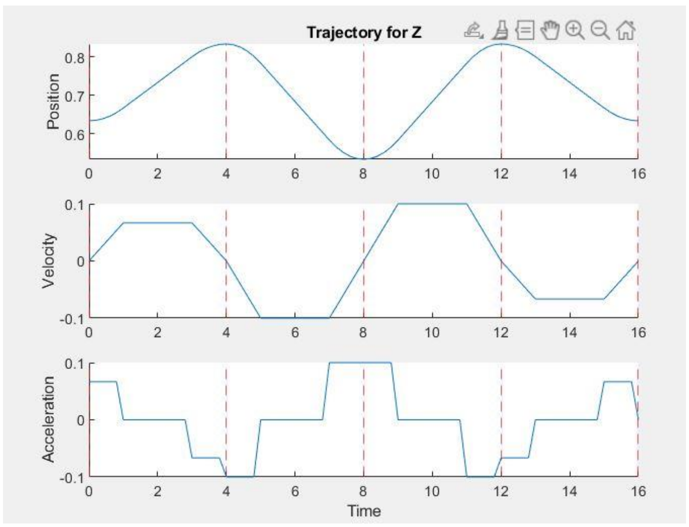


Fig 17. Trapezoidal trajectory obtained in X and Y direction in task space technique



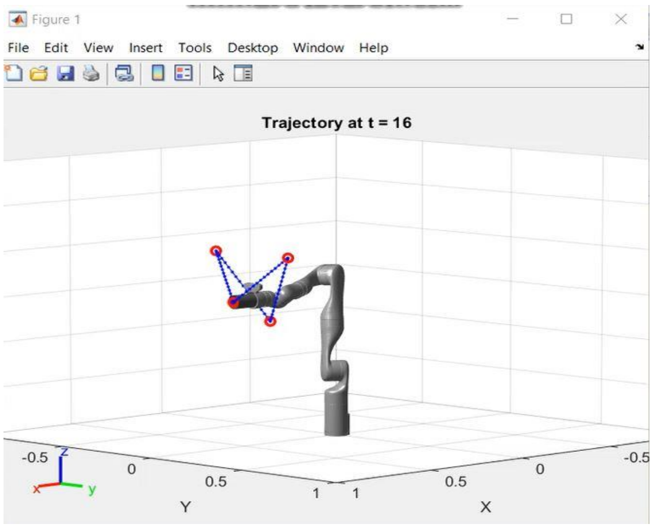


Fig 18. Trapezoidal trajectory obtained in Z direction in task space technique and the trajectory generated due to trapezoidal trajectory condition

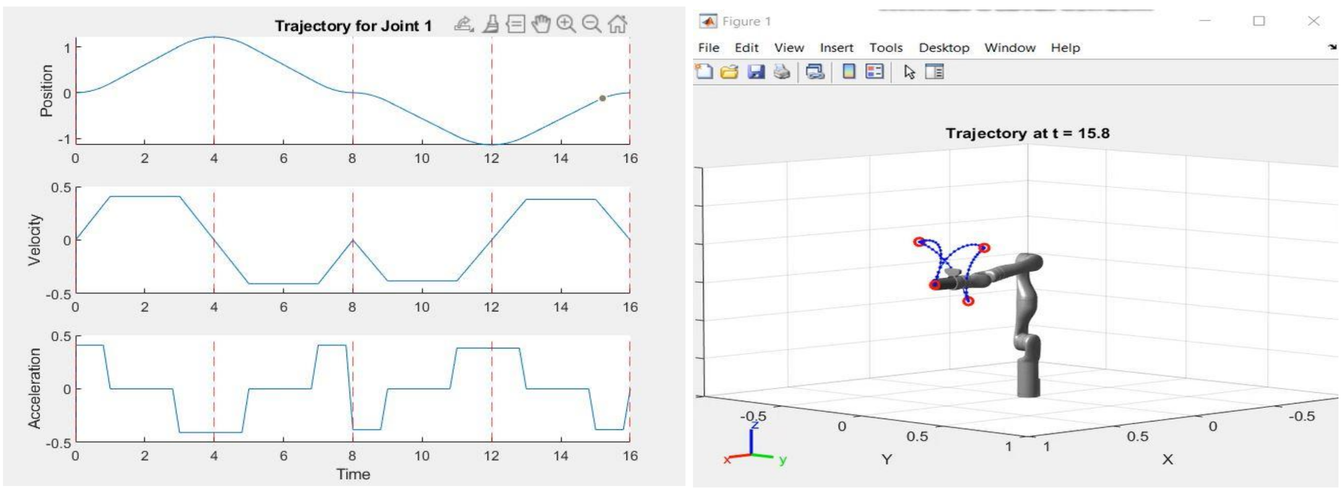


Fig 19. Trapezoidal trajectory obtained in Joint 1 and the trajectory generated in the joint space technique

The main advantage of trapezoidal trajectory is that they are simple to implement since they do not involve complex equations to solve for the trajectory.

5.2.2 Cubic polynomial Trajectory

In the case of cubic polynomial trajectory, the position is defined with the help of a cubic equation which contains four constants between two waypoints. These constants are found out with the help of four starting and end point conditions on position and velocity. Once this is found, the velocity and the accelerations at different points on the trajectory are obtained.

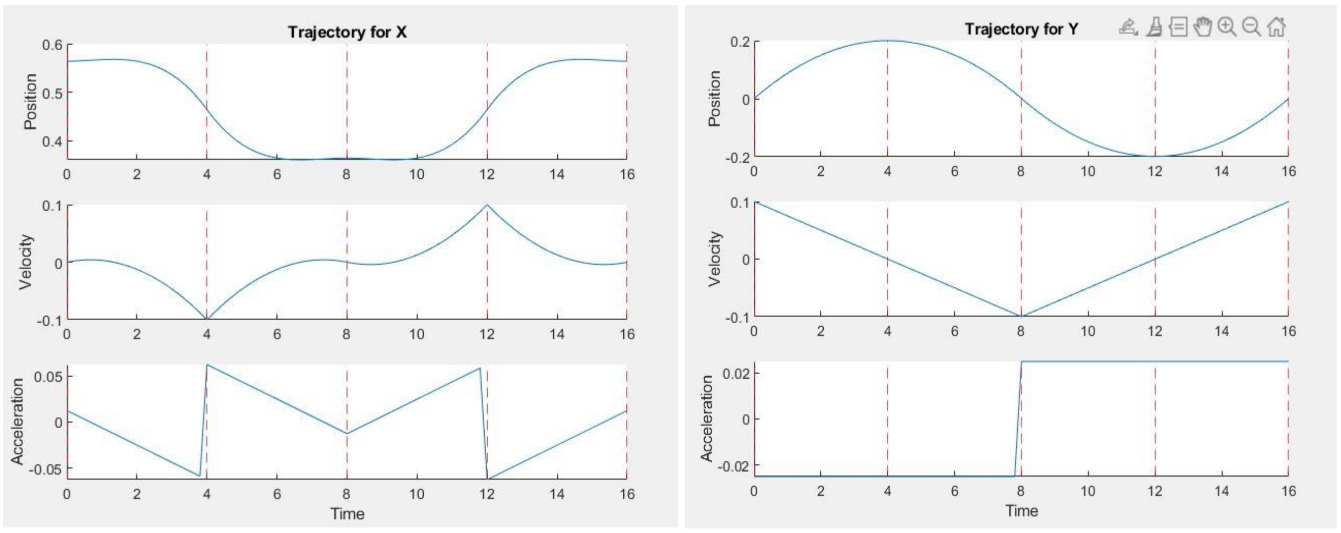


Fig 20. Cubic trajectory obtained in X and Y direction in task space technique

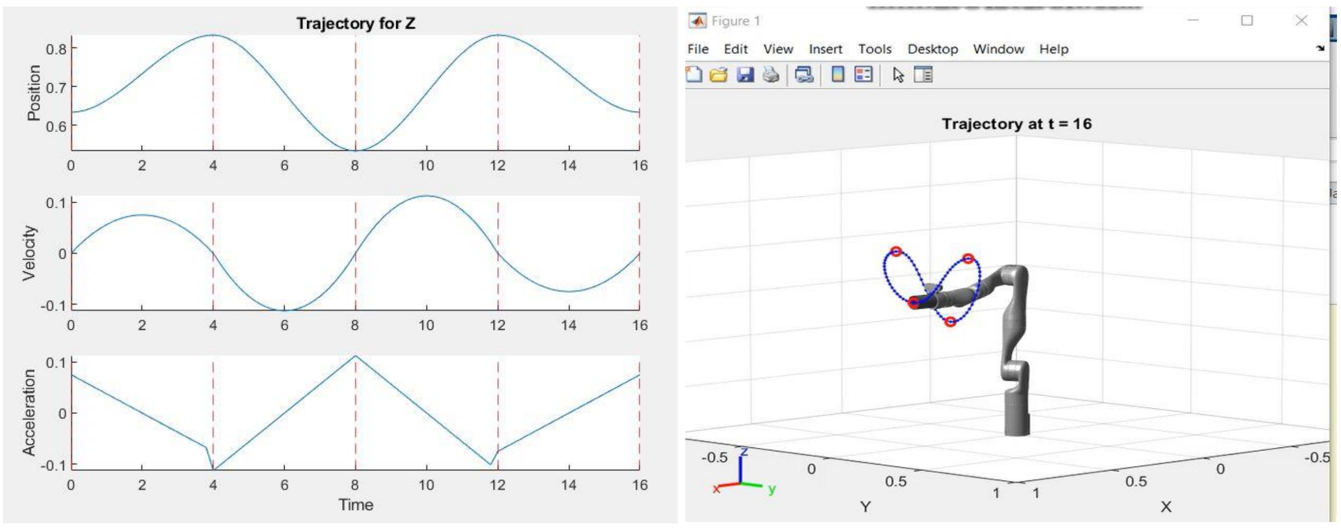


Fig 21. Cubic trajectory obtained in Z direction in task space technique and the trajectory generated due to cubic trajectory condition

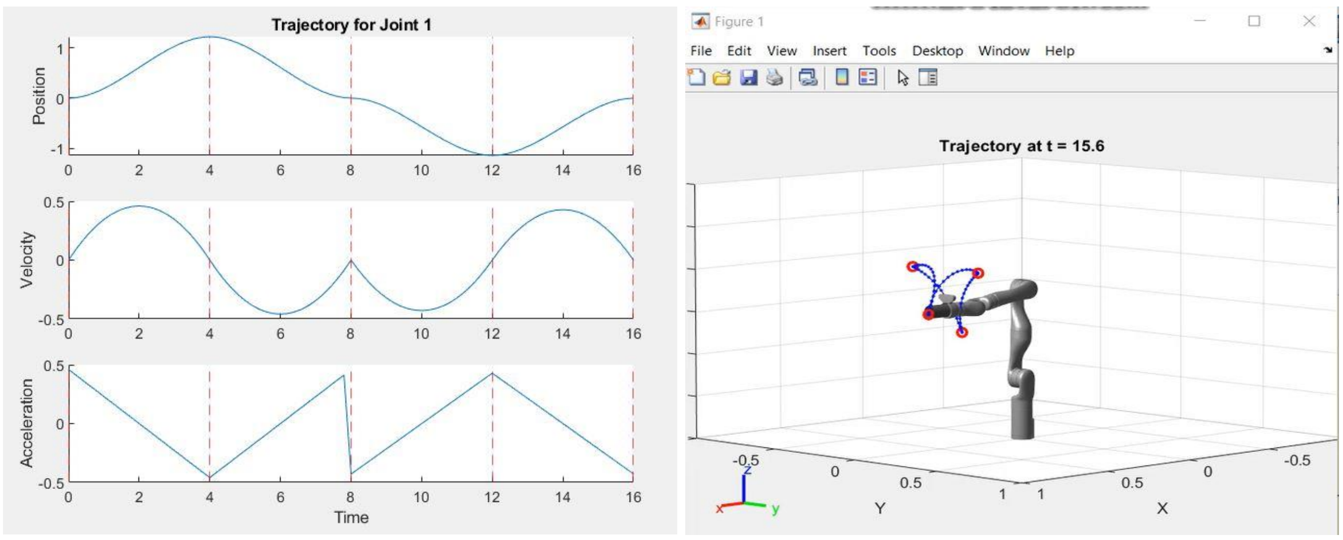


Fig 22. Cubic trajectory obtained in Joint 1 and the trajectory generated in the joint space technique

The main advantage of the polynomial trajectories is that we have the control on velocity and acceleration (in quintic) and velocity (in cubic). With this in control, we can perform motion as we require. For instance, in case of trapezoidal trajectory, the motion of the links stop every time when it reaches a waypoint. But in the case of polynomials, since we have the control on velocity and acceleration, we can keep the manipulator in motion even when the manipulator reaches the waypoints by setting non zero intermediate velocities.

5.2.3 Quintic polynomial Trajectory

In the case of quintic polynomial trajectory, the position is defined with the help of a fifth order equation which contains six constants between two waypoints. These constants are found out with the help of six starting and end point conditions on position, velocity and acceleration. Once this is found, the velocity and the accelerations at different points on the trajectory are obtained.

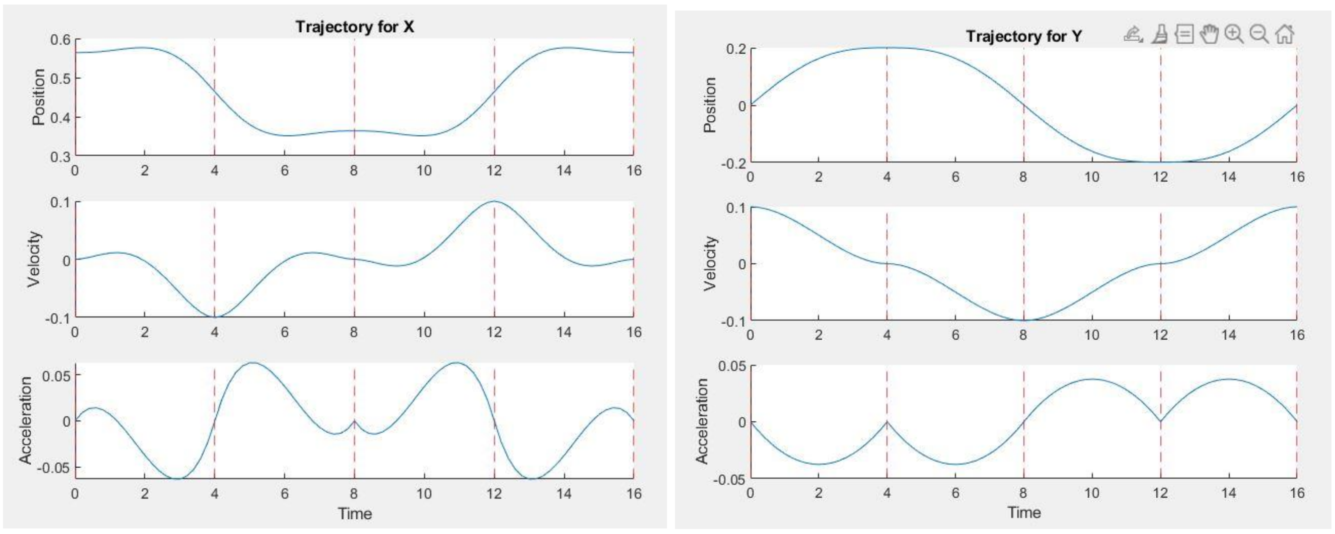


Fig 23. Quintic trajectory obtained in X and Y direction in task space technique

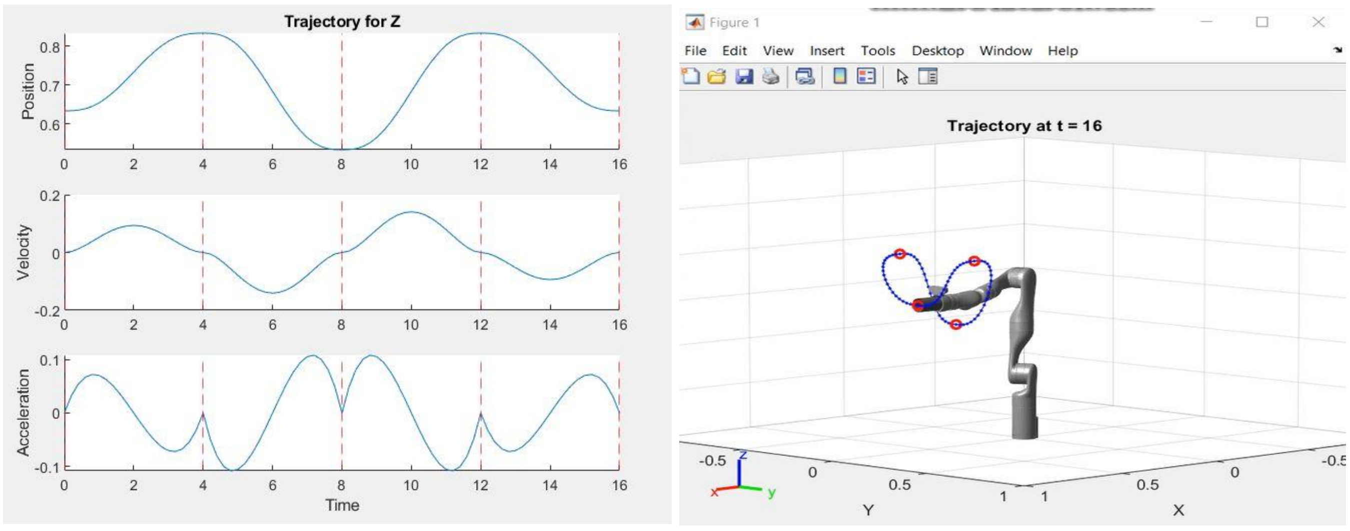


Fig 24. Quintic trajectory obtained in Z direction in task space technique and the trajectory generated due to quintic trajectory condition

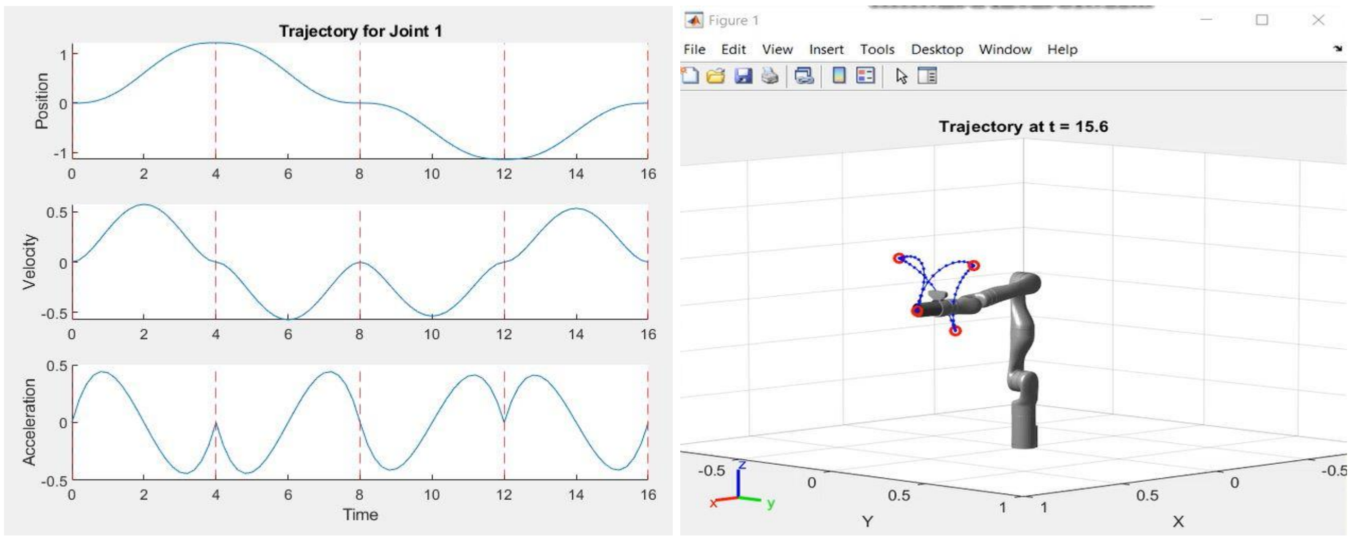


Fig 25. Quintic trajectory obtained in Joint 1 and the trajectory generated in the joint space technique

In previous two cases, we can see that the trajectory generated is smooth because of the non-zero velocities which were given for the analysis. It was also observed that, when we change the velocity conditions, the trajectory varies significantly in order to adjust to the conditions given.

CHAPTER 6:

SIMULATIONS IN ROS

6.1 Robot Operating System (ROS)

The primary software used for designing the manipulator is ROS (Robot Operating System). ROS is a robotics middleware (i.e., collection of software frameworks for robot software development). It provides a flexible framework for writing robot software. It provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message passing between processes, and package management. It is mainly used to connect and interface between various hardware components to work together unanimously by passing the information between them in a seamless manner irrespective of the language they are coded in different languages. This software can be specifically used in Ubuntu operating system.

The main use of this software comes in handy due to our use different hardware of different build, make and from different manufacturers. The main motors are from Kinova. All these motors are different and have different motor drivers and controllers. To make them work in unison together, there is a need to have a central control system provided by ROS(Robot Operating System). This software can be used as an interface between the code running in many different machines simultaneously.

6.2 URDF files

Unified Robotics Description Format or URDF is the way by which the CAD models can be given into the ROS software. It is an XML specification used in academia and industry to model multibody systems such as robotic manipulator arms for manufacturing assembly lines and animatronic robots for amusement parks.

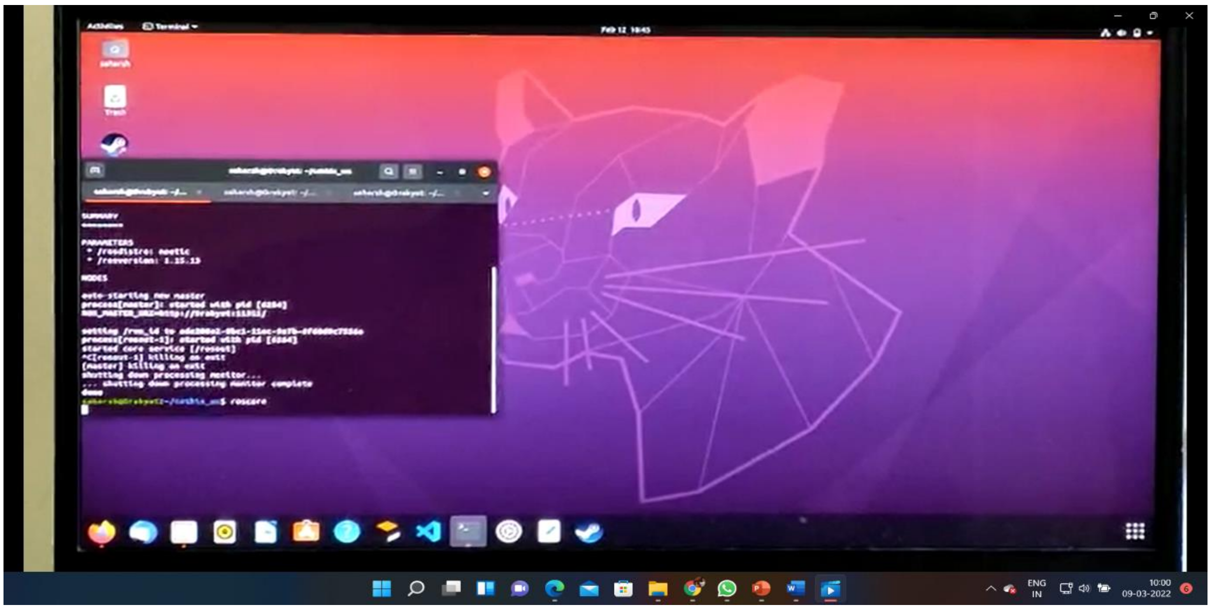
The URDF file consists of the details of joints and links. For links it consists of inertial and geometric data and for joints it contains the parent and the child link details, limits of the angle moved by the joint and the type of joint. The URDF file was created for the manipulator present in the lab. The URDF files contains details such as center of mass in all the directions of different links and also for the whole model, Moment of Inertia of each link, the material used to make the model and safety factors.

6.3 Simulations in Rviz

Rviz is a ROS package that is used to visualize the simulations done in ROS. Firstly, using MoveIt setup assistant, a MoveIt package was created using the URDF file made for the model and the joints, and kinematics solvers were initialised. While defining the MoveIt package, we need to give all the information related to self-collisions, virtual joints, planning groups, robot poses, end effectors and configuration files. Once the package is created, demo.launch and gazebo.launch are created that can be used to view the manipulator in Rviz and gazebo simulator respectively.

Now, a python script was created to calculate the inverse kinematics. Then, a trajectory generation script was also written, which contains a class definition that helps us in coordinating with the inverse kinematics script. We create another python file through which the waypoints are given. Now this python file calls the trajectory generation script to update the joint angles of the links with respect to the coordinates given. The trajectory generation script calls the inverse kinematics script to find the joint angles for the given coordinates. Hence the inverse kinematics is calculated for all the ways points through this method. Now, this script was run using ROS and the trajectory was visualised using ROS. Now, it is possible to simulate any kinds of shapes which are the essential part of the spray painting.

Let's go through the procedure to visualize the simulation. The below Fig 26 shows the start of the roscore in the terminal.



```
SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.13

NODES

auto-starting new master
process[master]: started with pid [6284]
ROS_MASTER_URI=http://[illegible]:11311/

setting /run_id to [illegible]
process[rosout-1]: started with pid [6304]
started core service [/rosout]
^C[rosout-1] killing on exit
[master] killing on exit
shutting down processing monitor...
... shutting down processing monitor complete
done
[illegible]$ roscore
```

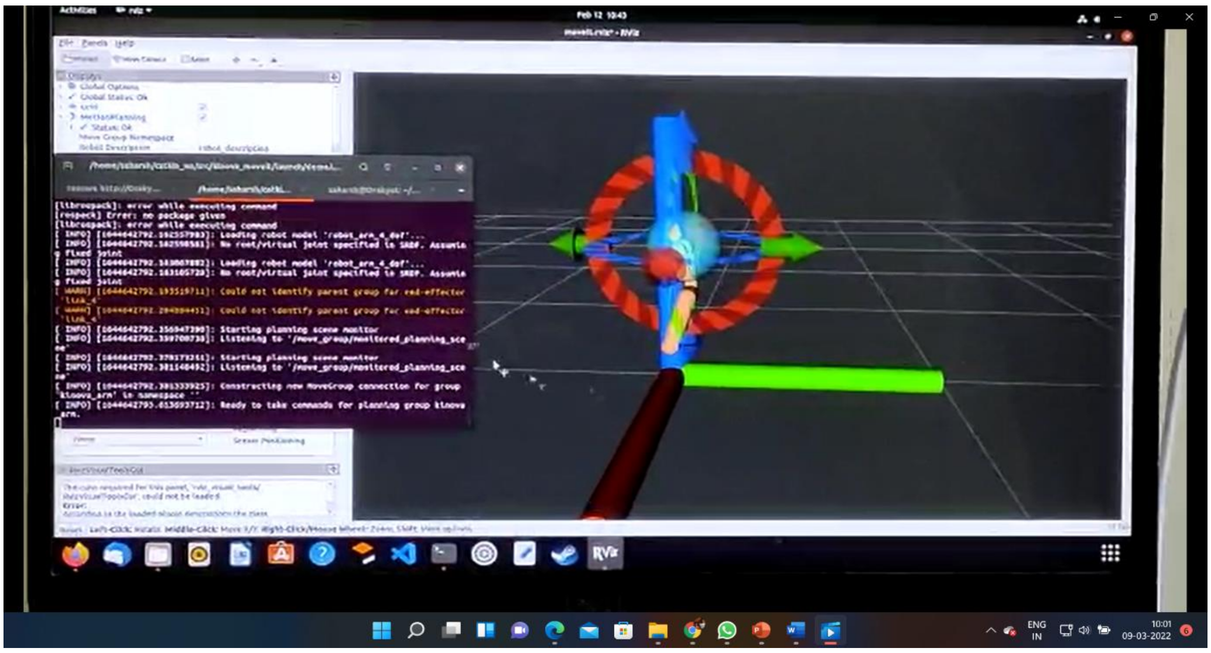
Fig 26. Initialisation of roscore

Now, inorder to view the manipulator, we launch demo.launch file that was created during the formation of moveit package using the moveit setup assistant. To do this, we use the command:

roslaunch package_name launchfile_name that is,

roslaunch kinova_moveit demo.launch

Once the demo.launch file is launched, we can see that the Rviz software has opened. Once we change the option to display robot model, the manipulator is displayed as shown in Fig 27.



```
[librospack]: error while executing command
[rospack] Error: no package given
[librospack]: error while executing command
[ INFO] [1644642792.[illegible]]: Loading robot model 'robot_arm_4_dof'...
[ INFO] [1644642792.[illegible]]: No root/virtual joint specified in SRDF. Assuming fixed joint
[ INFO] [1644642792.[illegible]]: Loading robot model 'robot_arm_4_dof'...
[ INFO] [1644642792.[illegible]]: No root/virtual joint specified in SRDF. Assuming fixed joint
[ WARN] [1644642792.[illegible]]: Could not identify parent group for end-effector 'link_4'
[ WARN] [1644642792.[illegible]]: Could not identify parent group for end-effector 'link_4'
[ INFO] [1644642792.[illegible]]: Starting planning scene monitor
[ INFO] [1644642792.[illegible]]: Listening to '/move_group/monitored_planning_scene'
[ INFO] [1644642792.[illegible]]: Starting planning scene monitor
[ INFO] [1644642792.[illegible]]: Listening to '/move_group/monitored_planning_scene'
[ INFO] [1644642792.[illegible]]: Constructing new MoveGroup connection for group 'kinova_arm' in namespace ''
[ INFO] [1644642793.[illegible]]: Ready to take commands for planning group kinova_arm.
```

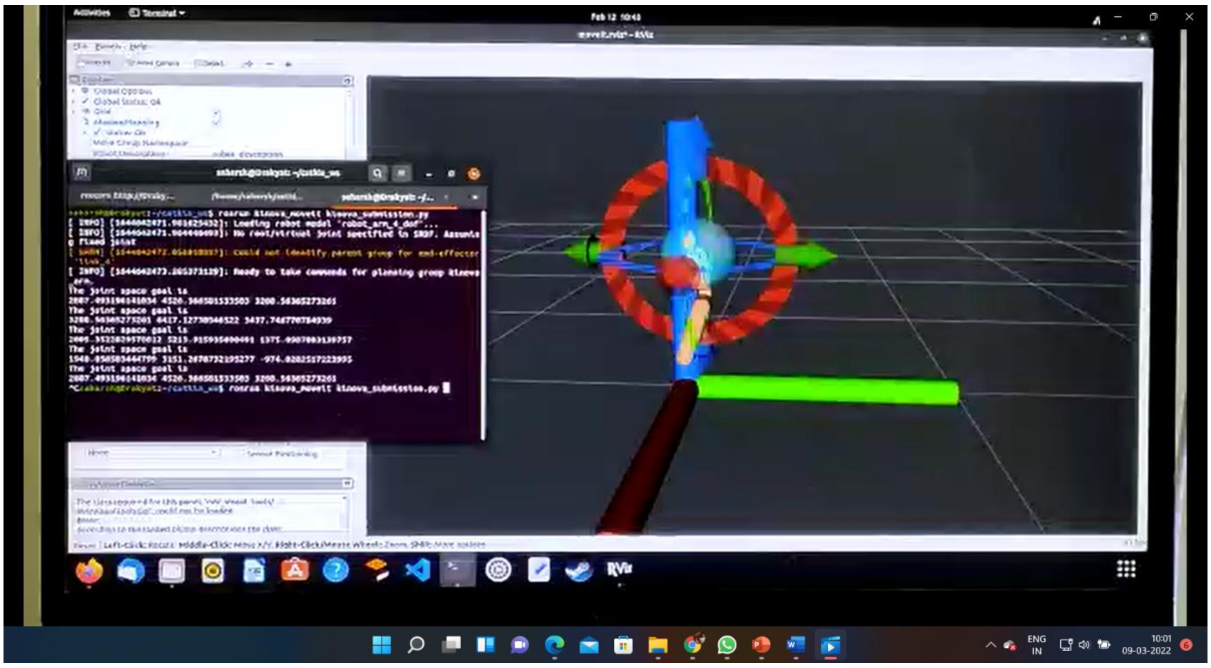
Fig 27. Opening and displaying the manipulator using Rviz

Once, the manipulator is displayed, we run the python scripts that were created to plan the trajectory for the given way points. This procedure was explained in the previous pages. To do this, we give the following command:

rosrun package_name script name that is,

rosrun kinova_moveit kinova_submission.py

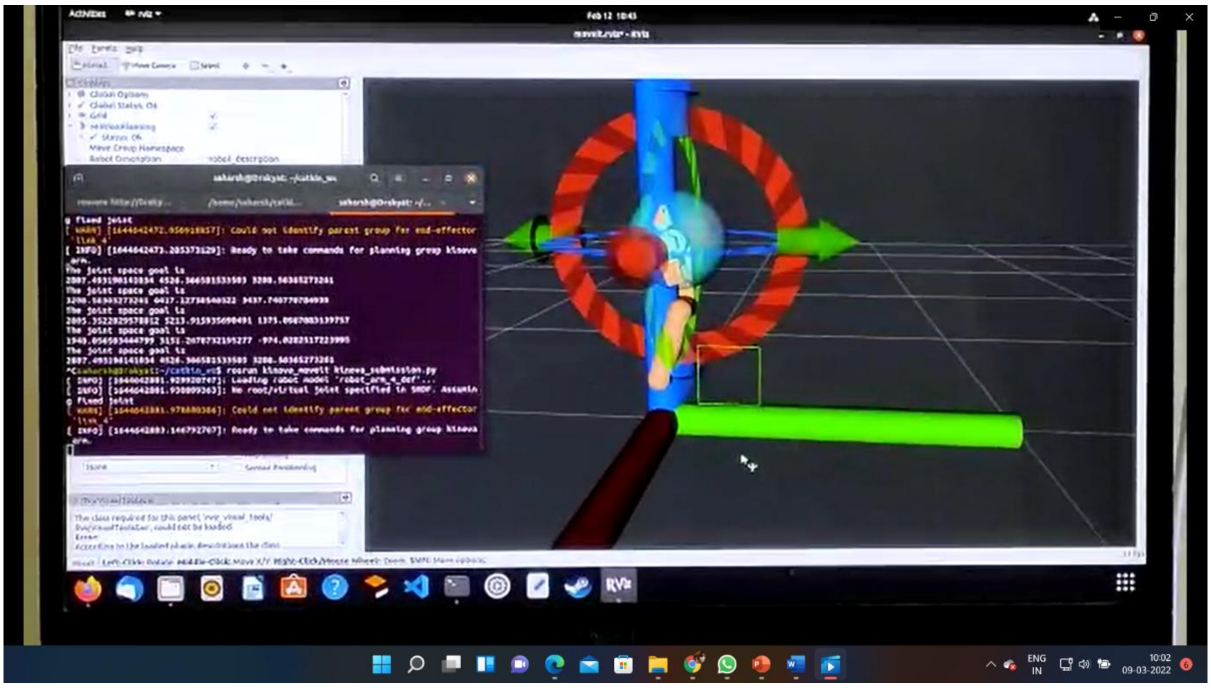
Here the kinova_submission.py is the file which coordinates with the trajectory generation and inverse kinematics script in order to find inverse kinematics for the given waypoints.



```
sakarsh@Drakyat:~/catkin_ws$ rosrun kinova_moveit kinova_submission.py
[ INFO] [1644042471.981825432]: Loading robot model 'robot_arm_4_dof'...
[ INFO] [1644042471.984440493]: No root/virtual joint specified in SRDF. Assumin
g fixed joint
[ WARN] [1644042472.056910857]: Could not identify parent group for end-effector
 'link_4'
[ INFO] [1644042473.205373129]: Ready to take commands for planning group kinova
_arm.
The joint space goal is
2807.493190141034 4526.366581533503 3208.56365273261
The joint space goal is
3208.56365273261 6417.12730540322 3437.748770784939
The joint space goal is
2805.3522829578812 5213.915935690491 1375.0507083139757
The joint space goal is
1940.056503444799 3151.2070732195277 -974.0282517223995
The joint space goal is
2807.493190141034 4526.366581533503 3208.56365273261
^Csakarsh@Drakyat:~/catkin_ws$ rosrun kinova_moveit kinova_submission.py
```

Fig 28. Calculation of inverse kinematics takes place once the rosrun is done

In order to visualize the path created by the coordinates given, we turn on the planned path option in the Rviz. There is an internal inverse kinematics solver present in the Rviz which creates the path between the waypoints. This is made to done using the joint space technique, that we had earlier concluded that it was the best way for us to make the trajectory. This path is displayed in green as shown in Fig 29.



```
g fixed joint
[ WARN] [1644042472.056910857]: Could not identify parent group for end-effector
 'link_4'
[ INFO] [1644042473.205373129]: Ready to take commands for planning group kinova
_arm.
The joint space goal is
2807.493190141034 4526.366581533503 3208.56365273261
The joint space goal is
3208.56365273261 6417.12730540322 3437.740770784939
The joint space goal is
2805.3522829578812 5213.915935690491 1375.0587083139757
The joint space goal is
1940.056503444799 3151.2070732195277 -974.0282517223995
The joint space goal is
2807.493190141034 4526.366581533503 3208.56365273261
^Csakarsh@Drakyat:~/catkin_ws$ rosrun kinova_moveit kinova_submission.py
[ INFO] [1644042881.929920767]: Loading robot model 'robot_arm_4_dof'...
[ INFO] [1644042881.930009363]: No root/virtual joint specified in SRDF. Assumin
g fixed joint
[ WARN] [1644042881.978880386]: Could not identify parent group for end-effector
 'link_4'
[ INFO] [1644042883.146792767]: Ready to take commands for planning group kinova
_arm.
```

Fig 29. Display of planned path in Rviz

This calculation done in real time between two waypoints. This motion of the manipulator can be viewed in Fig 30.

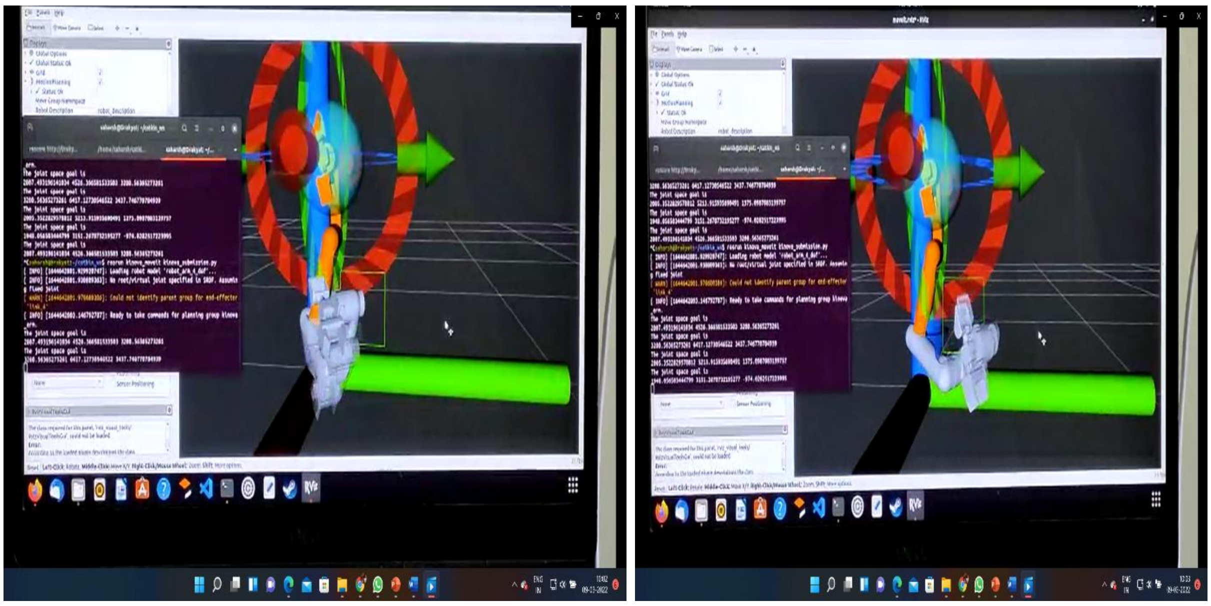
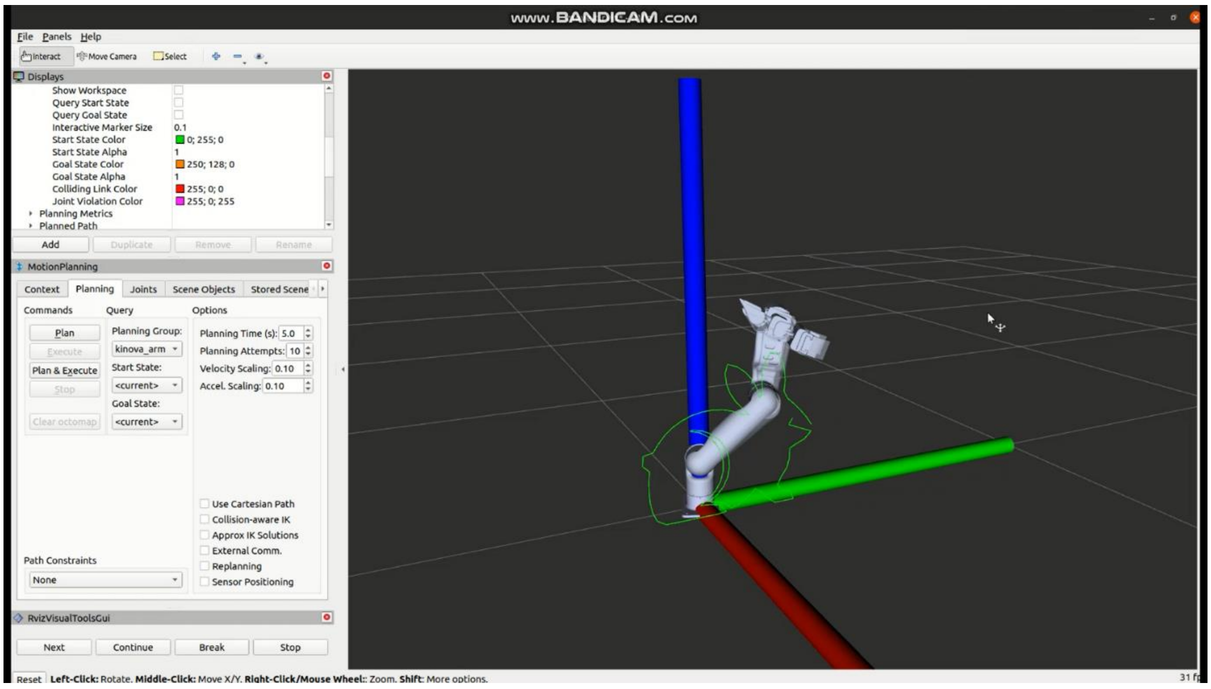


Fig 30. Motion of manipulator along the path given

Now, since the model created is a generalised one, this can be extended to any complex shape. This is very important from the spray-painting perspective since spray painting is all about implementing complex shapes in a given plane or in a three-dimensional space. Hence, to implement a complicated shape, the coordinates were obtained from external source[16] and was scaled as per the size of the manipulator. A model of a rabbit was given as large number of waypoints and path was created as shown in Fig 31.



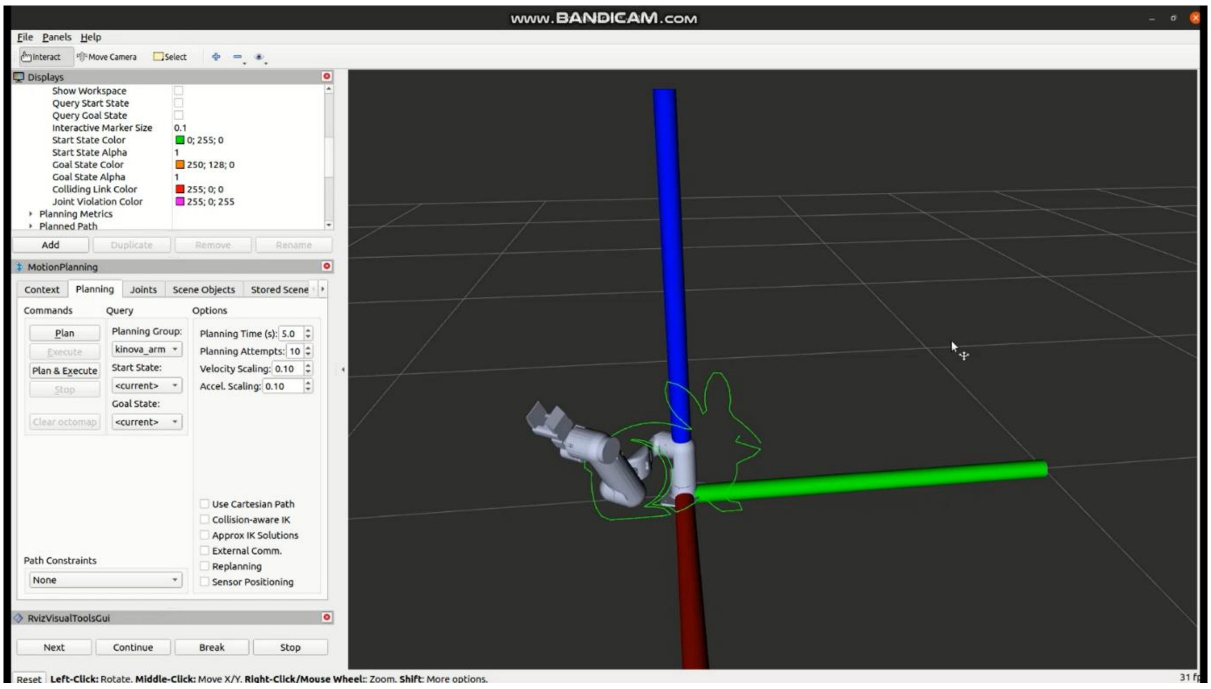
```
Show Workspace           ☐
Query Start State        ☐
Query Goal State         ☐
Interactive Marker Size  0.1
Start State Color        🟩 0; 255; 0
Start State Alpha        1
Goal State Color         🟧 250; 128; 0
Goal State Alpha         1
Colliding Link Color     🟥 255; 0; 0
Joint Violation Color    🟪 255; 0; 255
▸ Planning Metrics
▸ Planned Path

Planning Group: kinova_arm
Start State: <current>
Goal State: <current>
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.10
Accel. Scaling: 0.10
☐ Use Cartesian Path
☐ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning
Path Constraints: None

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
```

Fig 31. Path planned by Rviz for coordinates of rabbit model

Now, the intermediate position of the manipulator is shown in Fig 32.


www.BANDICAM.com
File Panels Help
Interact Move Camera Select
Displays
Planning Metrics
Planned Path
Add Duplicate Remove Rename
MotionPlanning
Context Planning Joints Scene Objects Stored Scene
Commands Query Options
Plan Execute Plan & Execute Stop Clear octomap
RvizVisualToolsGui
Next Continue Break Stop
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.

Show Workspace ☐
Query Start State ☐
Query Goal State ☐
Interactive Marker Size 0.1
Start State Color 🟩 0; 255; 0
Start State Alpha 1
Goal State Color 🟧 250; 128; 0
Goal State Alpha 1
Colliding Link Color 🟥 255; 0; 0
Joint Violation Color 🟪 255; 0; 255

Planning Group: kinova_arm
Start State: <current>
Goal State: <current>
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.10
Accel. Scaling: 0.10
☐ Use Cartesian Path
☐ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning
Path Constraints: None

Fig 32. Image showing in between position of manipulator and the planned path

CHAPTER 7:

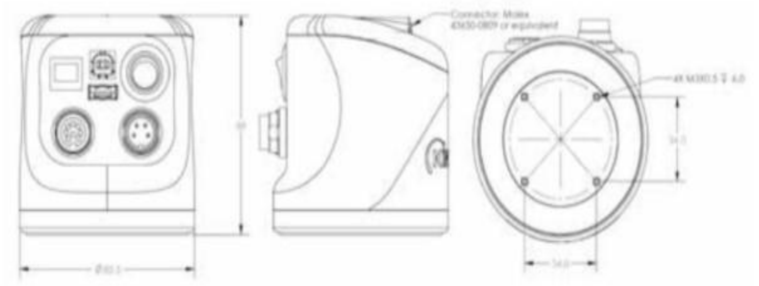
HARDWARE IMPLEMENTATION AND RESULTS

7.1 Hardware equipments used

The manipulator arms designed were 3D printed in the laboratory. The actuators used are Kinova KA-58 brushless dc motors. These are the apt actuators available in the market for making of robotics arm. These motors incorporate many functionalities in them such as precise angular control and sensor pre-embedded in them. The torque offered by these motors is around 3.6 Nm and its peak value is around 6.8 Nm.

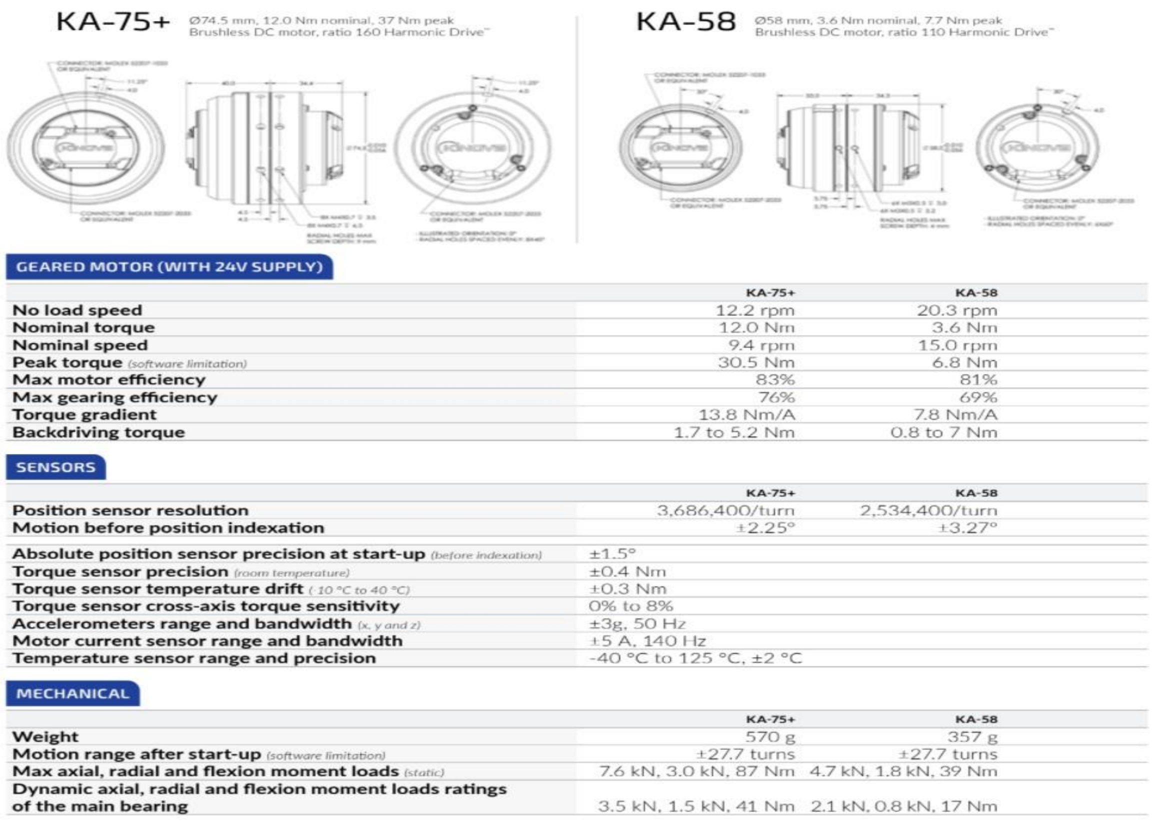
The motors of Kinova are only of 3 in number but our arm that has to be made is of 4-DOF which requires 4 motors including end-effector. Therefore, the kinematics data will be generated for 4 motors and transferred in between the controllers of the respective motors with the help of ROS. The below Fig 33 shows the details of the controller used and Fig 34 shows the details about the actuators used in the project.

CONTROLLER



Ports	Joystick	1 Mbps Canbus
	Power supply	18 to 29 VDC
	USB 2.0 (API)	12 Mbps
	Ethernet (API)	100 Mbps
Control system frequency	High level (API)	100 Hz
	Low level (API)	Up to 500 Hz
CPU		360 MHz
SDK	APIs	High and low level
	Compatibility	Windows, Linux Ubuntu & ROS
	Port	USB 2.0, Ethernet
	Programming languages	C++
Control		Force, cartesian & angular

Fig 33. Details about the Kinova controller



KA-75+ Ø74.5 mm, 12.0 Nm nominal, 37 Nm peak
Brushless DC motor, ratio 160 Harmonic Drive™

KA-58 Ø58 mm, 3.6 Nm nominal, 7.7 Nm peak
Brushless DC motor, ratio 110 Harmonic Drive™

GEARED MOTOR (WITH 24V SUPPLY)

	KA-75+	KA-58
No load speed	12.2 rpm	20.3 rpm
Nominal torque	12.0 Nm	3.6 Nm
Nominal speed	9.4 rpm	15.0 rpm
Peak torque *(software limitation)*	30.5 Nm	6.8 Nm
Max motor efficiency	83%	81%
Max gearing efficiency	76%	69%
Torque gradient	13.8 Nm/A	7.8 Nm/A
Backdriving torque	1.7 to 5.2 Nm	0.8 to 7 Nm

SENSORS

	KA-75+	KA-58
Position sensor resolution	3,686,400/turn	2,534,400/turn
Motion before position indexation	±2.25°	±3.27°

Absolute position sensor precision at start-up *(before indexation)*	±1.5°
Torque sensor precision *(room temperature)*	±0.4 Nm
Torque sensor temperature drift *(10 °C to 40 °C)*	±0.3 Nm
Torque sensor cross-axis torque sensitivity	0% to 8%
Accelerometers range and bandwidth *(x, y and z)*	±3g, 50 Hz
Motor current sensor range and bandwidth	±5 A, 140 Hz
Temperature sensor range and precision	-40 °C to 125 °C, ±2 °C

MECHANICAL

	KA-75+	KA-58
Weight	570 g	357 g
Motion range after start-up *(software limitation)*	±27.7 turns	±27.7 turns
Max axial, radial and flexion moment loads *(static)*	7.6 kN, 3.0 kN, 87 Nm	4.7 kN, 1.8 kN, 39 Nm
Dynamic axial, radial and flexion moment loads ratings of the main bearing	3.5 kN, 1.5 kN, 41 Nm	2.1 kN, 0.8 kN, 17 Nm

Fig 34. Details about the Kinova KA-58 actuator

The actuator connections and hardware after setting up completely is shown in Fig 35.

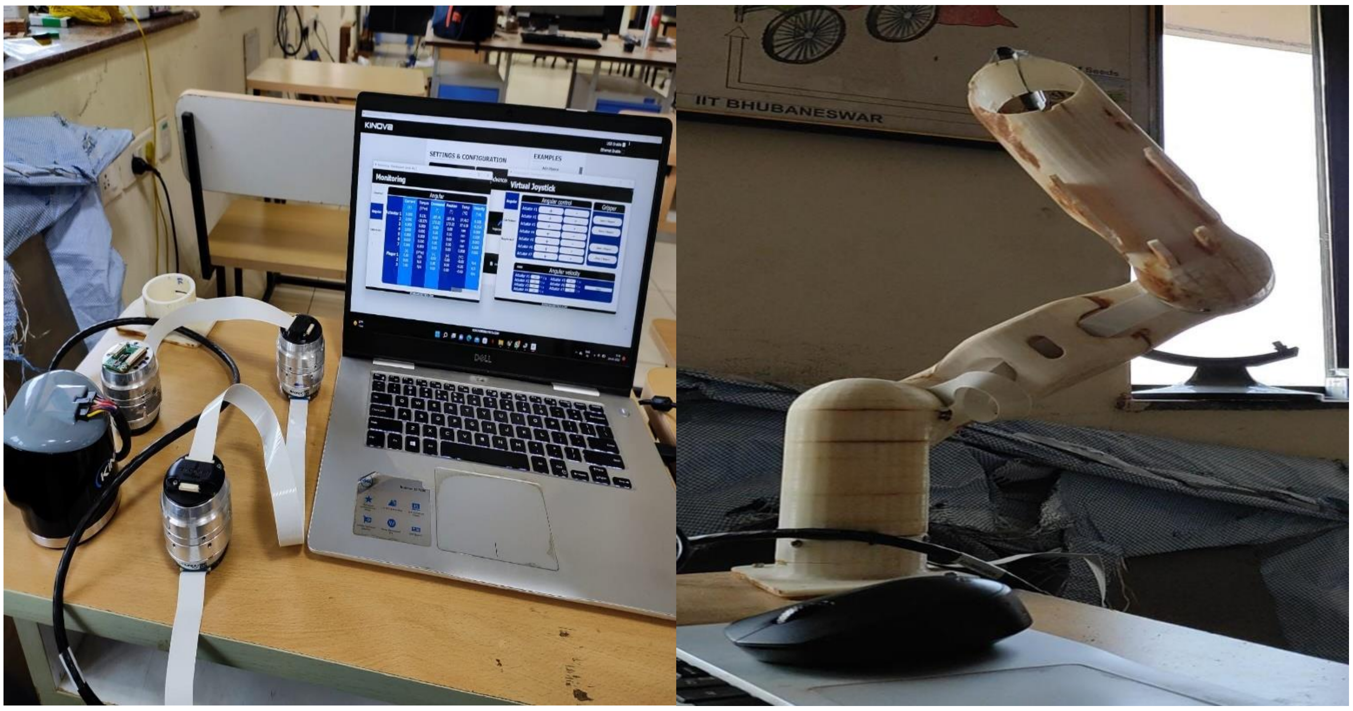
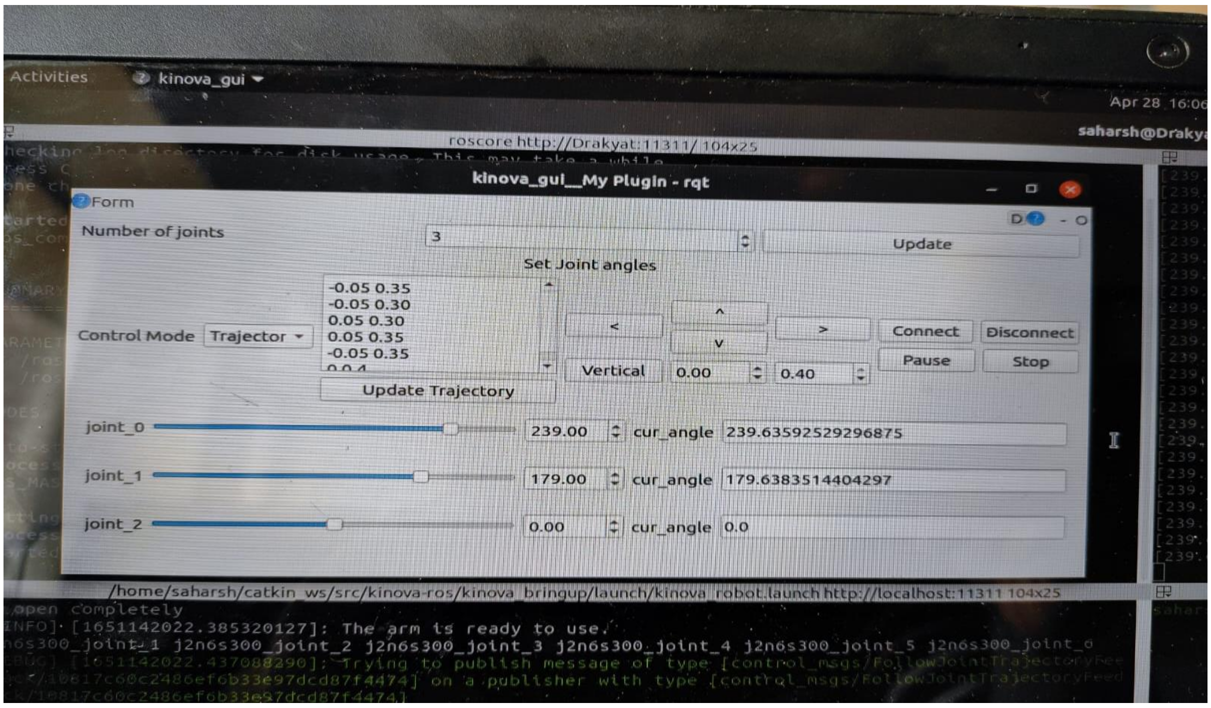


Fig 35. Actuators and Complete Hardware Setup

7.2 Interaction between hardware and software using ROS

The hardware setup is connected such that the Arduino cable is used to connect the Kinova controller to the laptop. Then, the power cable is also connected to the controller via which the power for the controller and the actuator is sent. Using the 20-pin cable provided, the actuator and the chip is connected and the other controller specific connector is connected between the controller and the chip. The further actuators are connected in series with the first actuator which is connected to the controller. This completes the actuator setup.

Now, in order to interact with the manipulator, GUI was created using rqt console with the help of ROS. The following Fig. 36 shows the GUI created.



```
kinova_gui__My Plugin - rqt
Form
Number of joints  3                          Update
Set Joint angles
-0.05 0.35
-0.05 0.30
0.05 0.30
0.05 0.35
-0.05 0.35
Control Mode  Trajector
Vertical  0.00  0.40
joint_0  239.00  cur_angle  239.63592529296875
joint_1  179.00  cur_angle  179.6383514404297
joint_2  0.00    cur_angle  0.0
```

```
roscore http://Drakyat:11311/ 104x25
/home/saharsh/catkin_ws/src/kinova-ros/kinova_bringup/launch/kinova_robot.launch http://localhost:11311 104x25
open completely
INFO] [1651142022.385320127]: The arm is ready to use.
n6s300_joint_1 j2n6s300_joint_2 j2n6s300_joint_3 j2n6s300_joint_4 j2n6s300_joint_5 j2n6s300_joint_6
EBUG] [1651142022.437088290]: Trying to publish message of type [control_msgs/FollowJointTrajectoryFee
ck/10817c60c2486ef6b33e97dcd87f4474] on a publisher with type [control_msgs/FollowJointTrajectoryFeed
ck/10817c60c2486ef6b33e97dcd87f4474]
```

Fig 36. GUI to control the manipulator

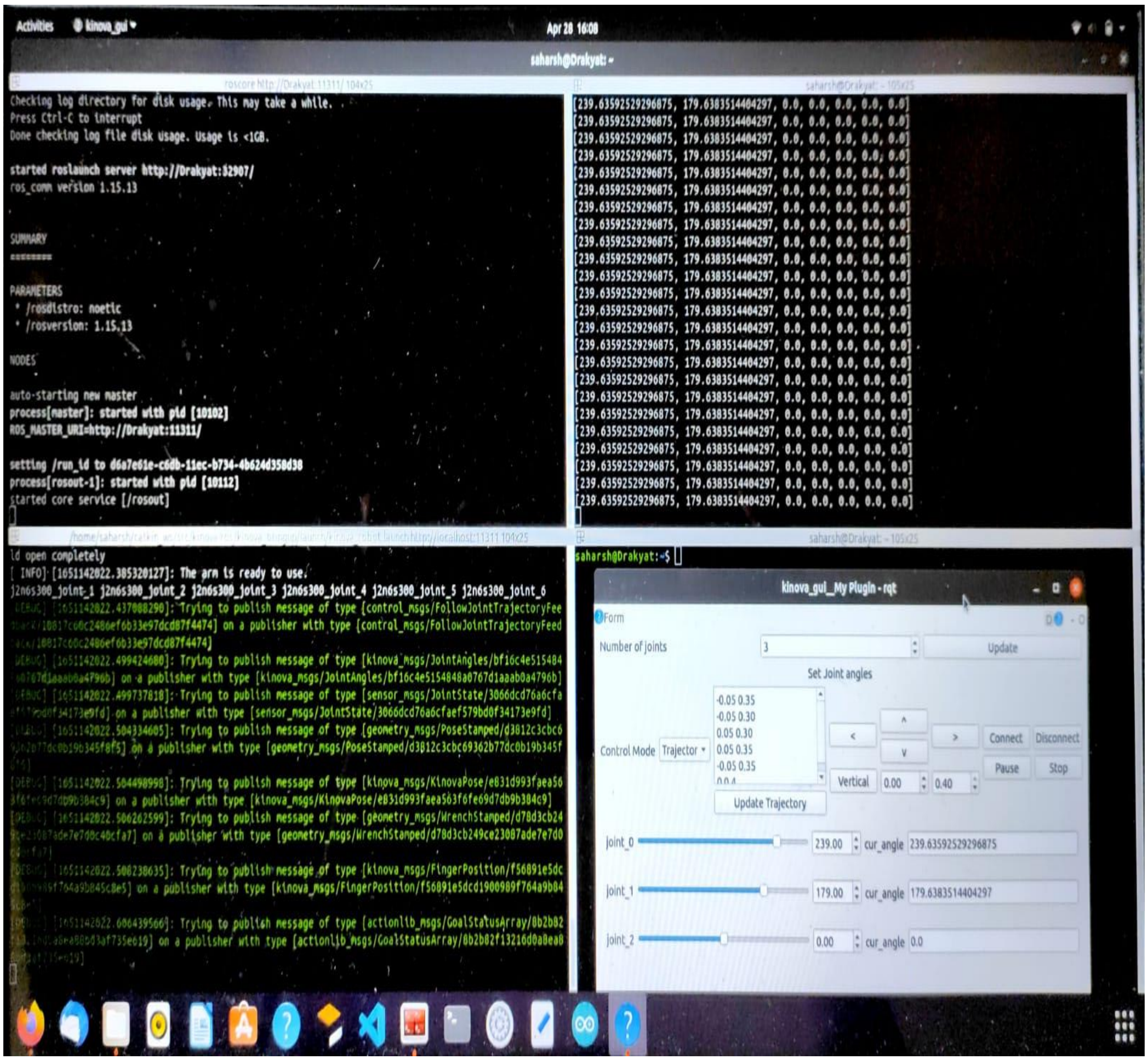
Features included in the GUI: Firstly, the GUI created is generalized for until four degrees of freedom. This was done since few of the actuators in the lab was not working and hence it was necessary to include the other DOFs as well in order to produce the required results. Here, once we update the number of joints, the equivalent number of sliders will appear as shown in Fig 36. These sliders can be used to increase and decrease the angles, etc. Additionally, there are four different control modes which were put in namely:

(i) Position Control
(ii) Velocity Control
(iii) Cartesian Control
(iv) Trajectory Control

The four buttons connect, disconnect, start and stop are self-explanatory. Position control is the control via which the absolute angles of the joints are given as input through the slider or through the box provided to the side of it. Once the start button is pressed, the manipulator moves to the angles input by the user. The second control mode provided is velocity control. In this type of control, the velocity of the manipulator's joint angles is given as input through the slider or the dialogue box near it to. The maximum angular velocity the actuator can obtain is limited to 60 degrees/sec in either direction. Once the slider is set to different angular velocities and the start is pressed, the actuator starts rotating in the velocity. A stop button is also provided in order to halt the manipulator if anything unexpected happens. The third control provided is cartesian control. In this type of control, the arrow marks provided in the GUI can be used. This kind of control makes the end effector go in any of the four directions i.e., up, down, right and left. The inverse kinematics are calculated accordingly and the joint angles are changed such that it follows a straight-line pattern. Finally, trajectory control is also provided. This uses the big dialogue box provided in the GUI. In case of trajectory control, we can give a set of coordinates that are reachable within the workspace. Once the start is pressed, the manipulator moves according the set of coordinates in the same given order. The vertical option is provided under cartesian control in order to bring it to the home position whenever necessary. The two dialogue boxes that are provided next to the vertical button shows us the position of the end effector with respect to the origin of the manipulator. These four controls methods can be used to move the manipulator in the required ways and obtain the required trajectories for achieving the spray-painting objective.

In order to initiate the control, roscore has to be initialised first. Then, the Kinova launch file has to be launched that was created to interact with the hardware. Finally, the rosrun command has to be used in order to open the GUI created. We can also subscribe to various other topics provided by Kinova in order to keep a check on the parameters of the actuators such as temperature of the actuator, velocity, position, etc. All the codes are written using C++ and Python. It is due this flexibility and the ability to interact with the hardware, ROS comes in handy for this project.

The workspace after doing the above-mentioned steps and after opening the GUI looks like as shown in Fig 37.



```
Activities   kinova_gui ▾                     Apr 28 16:08
saharsh@Drakyat: ~
```

```
roscore http://Drakyat:11311/ 104x25
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://Drakyat:32907/
ros_comm version 1.15.13


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.13

NODES

auto-starting new master
process[master]: started with pid [10102]
ROS_MASTER_URI=http://Drakyat:11311/

setting /run_id to d6a7e61e-c6db-11ec-b734-4b624d358d38
process[rosout-1]: started with pid [10112]
started core service [/rosout]
```

```
saharsh@Drakyat: ~ 105x25
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
[239.63592529296875, 179.6383514404297, 0.0, 0.0, 0.0, 0.0, 0.0]
```

```
/home/saharsh/catkin_ws/src/kinova-ros/kinova_bringup/launch/kinova_robot.launch http://localhost:11311 104x25
ld open completely
[ INFO] [1651142022.385320127]: The arm is ready to use.
j2n6s300_joint_1 j2n6s300_joint_2 j2n6s300_joint_3 j2n6s300_joint_4 j2n6s300_joint_5 j2n6s300_joint_6
[DEBUG] [1651142022.437088290]: Trying to publish message of type [control_msgs/FollowJointTrajectoryFee
dback/10817c60c2486ef6b33e97dcd87f4474] on a publisher with type [control_msgs/FollowJointTrajectoryFeed
back/10817c60c2486ef6b33e97dcd87f4474]
[DEBUG] [1651142022.499424680]: Trying to publish message of type [kinova_msgs/JointAngles/bf16c4e515484
8a0767d1aaab0a4796b] on a publisher with type [kinova_msgs/JointAngles/bf16c4e5154848a0767d1aaab0a4796b]
[DEBUG] [1651142022.499737818]: Trying to publish message of type [sensor_msgs/JointState/3066dcd76a6cfa
ef579bd0f34173e9fd] on a publisher with type [sensor_msgs/JointState/3066dcd76a6cfaef579bd0f34173e9fd]
[DEBUG] [1651142022.504334605]: Trying to publish message of type [geometry_msgs/PoseStamped/d3812c3cbc6
9362b77dc0b19b345f8f5] on a publisher with type [geometry_msgs/PoseStamped/d3812c3cbc69362b77dc0b19b345f
8f5]
[DEBUG] [1651142022.504498998]: Trying to publish message of type [kinova_msgs/KinovaPose/e831d993faea56
3f6fe69d7db9b384c9] on a publisher with type [kinova_msgs/KinovaPose/e831d993faea563f6fe69d7db9b384c9]
[DEBUG] [1651142022.506262599]: Trying to publish message of type [geometry_msgs/WrenchStamped/d78d3cb24
9ce23087ade7e7d0c40cfa7] on a publisher with type [geometry_msgs/WrenchStamped/d78d3cb249ce23087ade7e7d0
c40cfa7]
[DEBUG] [1651142022.508238635]: Trying to publish message of type [kinova_msgs/FingerPosition/f56891e5dc
d1900989f764a9b845c8e5] on a publisher with type [kinova_msgs/FingerPosition/f56891e5dcd1900989f764a9b84
5c8e5]
[DEBUG] [1651142022.606439566]: Trying to publish message of type [actionlib_msgs/GoalStatusArray/8b2b82
f13216d0a8ea88bd3af735e619] on a publisher with type [actionlib_msgs/GoalStatusArray/8b2b82f13216d0a8ea8
8bd3af735e619]
```

```
saharsh@Drakyat: ~ 105x25
saharsh@Drakyat:~$
```

kinova_gui_My Plugin - rqt

Form

Number of joints: 3 — Update

Set Joint angles

Control Mode: Trajector

```
-0.05 0.35
-0.05 0.30
0.05 0.30
0.05 0.35
-0.05 0.35
0.0 4
```

Update Trajectory

Vertical: 0.00 0.40

joint_0	239.00	cur_angle	239.63592529296875
joint_1	179.00	cur_angle	179.6383514404297
joint_2	0.00	cur_angle	0.0

Fig 37. Image of the workspace during the interaction with the hardware

In the above workspace shown in Fig 37, the top left terminal has been used to initiate the roscore. The bottom left terminal is used to launch the launch file which is required to interact with the controller of the hardware. The top right terminal is used to open the GUI. Also, the top right terminal will also provide the details of the angles of the actuators continuously until it is stopped. The bottom right of the screen is the GUI and the bigger version of it is shown in Fig 36.

7.3 Manipulator prototype testing

Once the whole setup is completed, the few experiments were carried out to test the movement of the manipulator. First, the manipulator was made to draw a simple parallelogram on the white A4 sheet in order to make sure that the manipulator is able to perform simple movements. The following set of images shows the steps through which the manipulator was successfully able to draw the parallelogram.

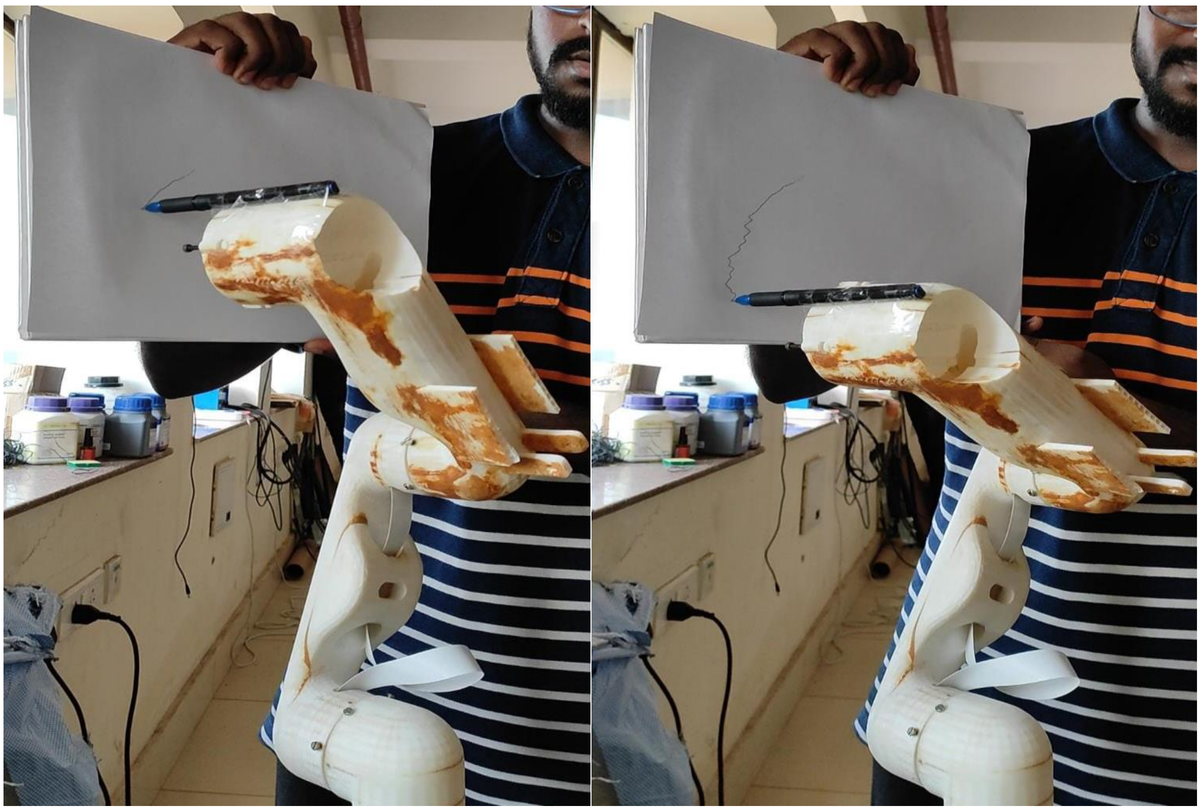


Fig 38. Steps 1 and 2 for drawing parallelogram

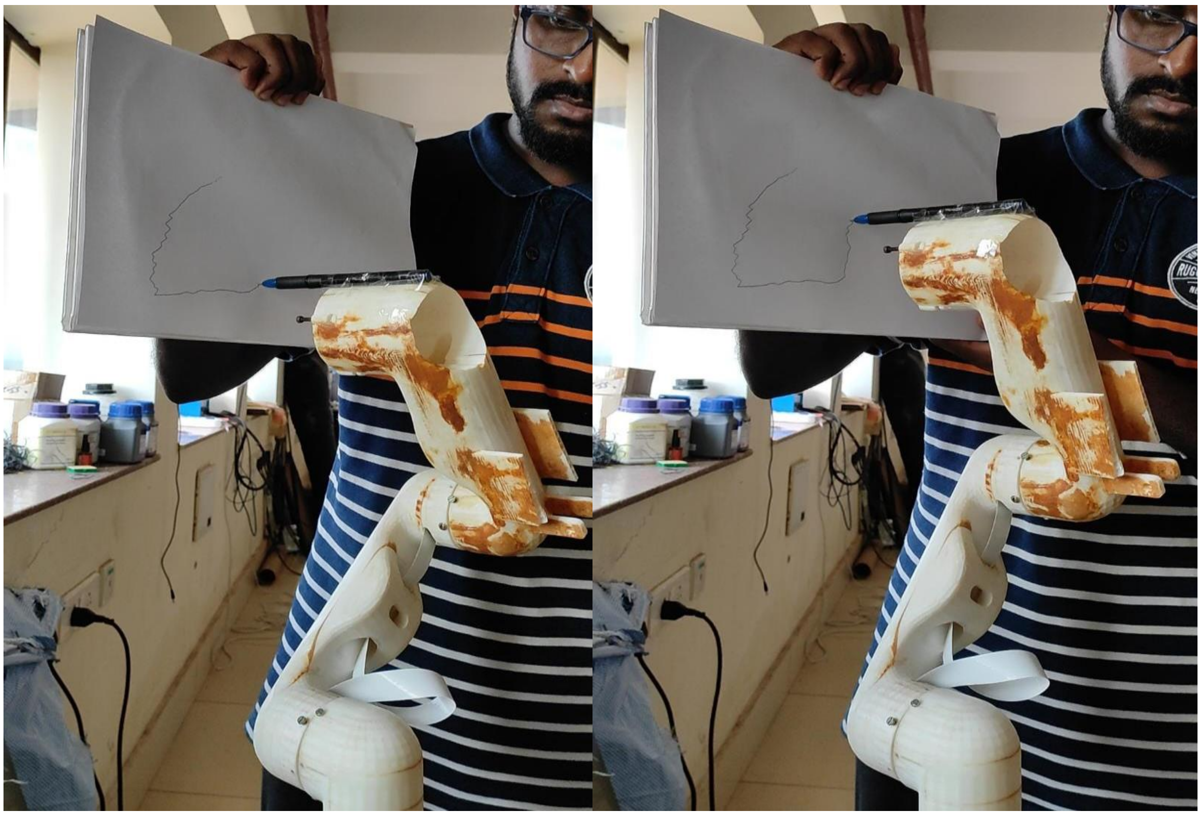


Fig 39. Steps 3 and 4 for drawing parallelogram

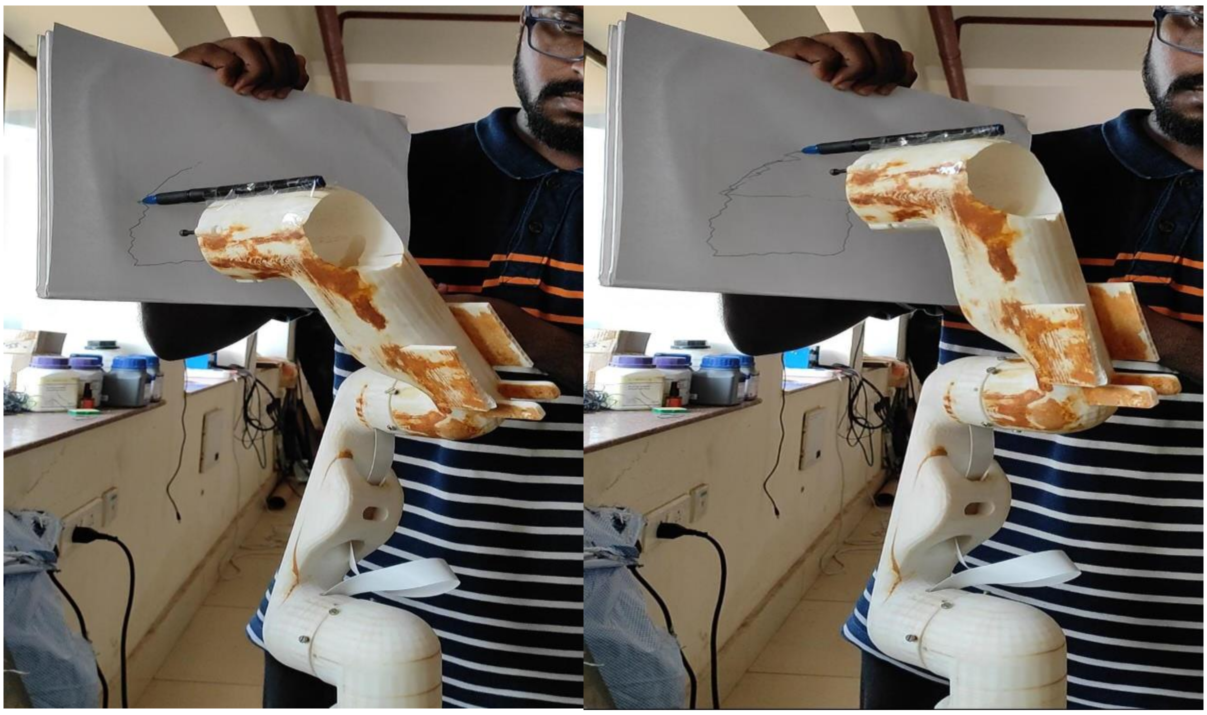


Fig 40. Steps 5 and 6 for drawing parallelogram

From the above steps, we can see that the manipulator is able to draw simple shapes. Now to increase the complexity, the coordinates were given such that it can write few alphabets. As expected, it was able to successfully implement “IIT” on the white sheet. The Fig 41 shows the parallelogram drawn by the manipulator and Fig 42 shows “IIT” which written by the manipulator in the similar steps as shown above.

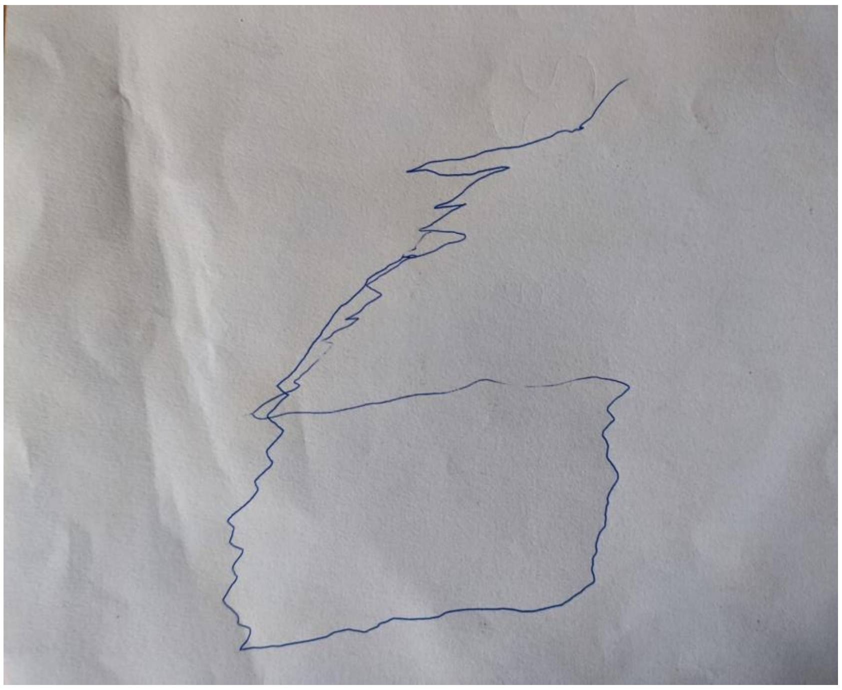


Fig 41. Parallelogram drawn by the manipulator

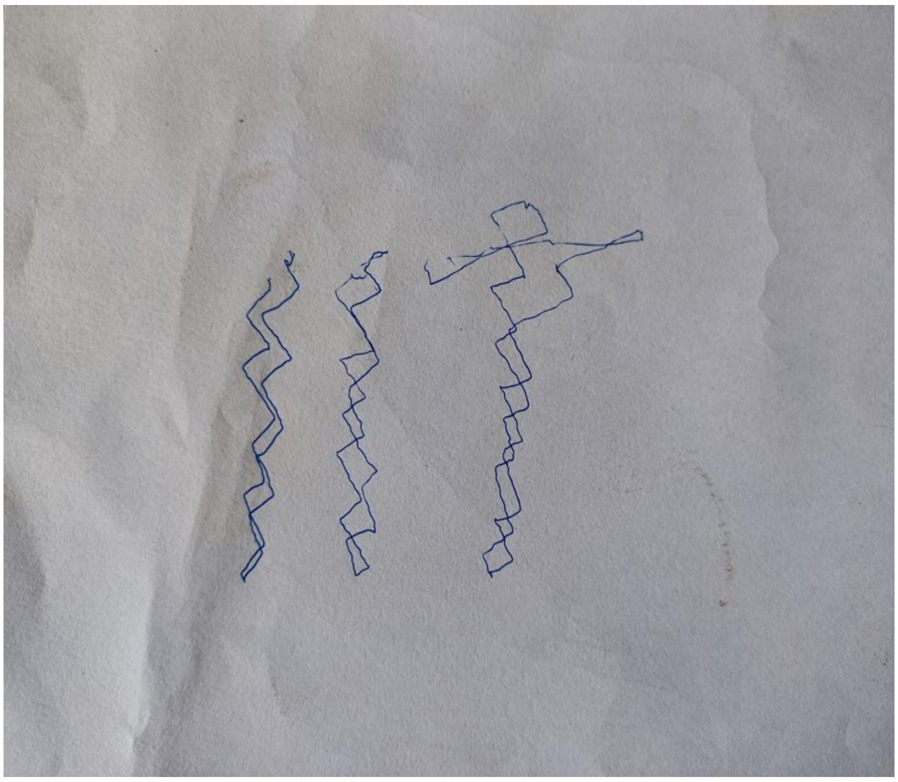


Fig 42. "IIT" written by the manipulator

In Fig 41, there is a diagonal line at the top of the page. This line is drawn due to the movement of manipulator from the rest position to the starting position of the manipulator and going back to the initial position. In the actual case, this should not happen because the fourth DOF will be adjusting the pen/spray-painting device such that during the non-useful motion, the device is not in contact with the surface.

In both Fig 41 and 42, the vertical lines are drawn in a zig zag fashion. This can arise due to two different reasons. Firstly, it can be due to internal difference in the angular velocities of the actuators. Secondly and most importantly, it happens due to its inherent nature of joint space control action i.e., whenever two waypoints are given, it tries to go such that the movement in the actuators is minimum and the time taken is minimum. In order to make it a straight line, it has been coded such that whenever two waypoints are given, when it is required to move in a straight line, using linspace command, multiple waypoints between the two given waypoints are created automatically and the manipulator is made to go through all the waypoints. This is the reason why the zig zag movement of end effector happens. This can be removed by making a greater number of waypoints between two waypoints but it will take a bit longer to compute and execute the motion.

CHAPTER 8:

CONCLUSION

Firstly, an in-depth literature review was done on various manipulators which are currently present in the industries. There were few techniques for trajectory planning using Machine Learning which I will be trying to implement in the next phase of the project.

Secondly, design of the spray-painting manipulator was done using SolidWorks based on the dimensions and the model of the manipulator which is currently present in the lab. A separate base was designed according to constraints of the manipulator which will later be 3D printed. This model developed will further be useful when imported into MATLAB using SimScape multibody link later to visualize the exact trajectory. Consequently, after designing the model, the kinematics of the model was studied using MATLAB. The forward and inverse kinematics was studied and was used in further analysis. Then, the differential motion was studied by calculating the Jacobian matrix which will help us knowing the velocities of different links present in the manipulator. Finally, using MATLAB, the trajectory planning algorithms were made and two ways of trajectory planning was studied, they are task space and joint space. Later, different types of trajectories were studied including trapezoidal, cubic and quintic. After doing this analysis, the following is considered the best way to move forward with the exact trajectory planning scenario for the spray-painting manipulator.

As the path traversed is very important in spray painting process, because we need to follow the surface at a given fixed distance to output optimum paint on the surface and time taken for the generation of such trajectory is not that important, from the results obtained, task space trajectory system will be the most suited one for the spray-painting manipulator case. Secondly, regarding which type of trajectory will be best suited for the case of spray painting, polynomial trajectories will be the best suited. This is because the polynomial trajectories enable us to constrain the velocities and allows us to have a complete control on the trajectory which is required for the spray-painting manipulator.

Once the trajectory techniques are shortlisted, I learnt all the required things for making simulations using ROS. The main part of the spray-painting process, i.e., trajectory generation for the required shapes and letters was done using Rviz. A new and efficient way to calculate the trajectory using a regression model was done and the mean absolute error obtained is around 0.95168 which is pretty accurate. Once this is done, the work on the hardware part started.

The complete hardware was setup using the 3D printed arms and the actuators and controller bought from Kinova. Now, the ROS codes required to make the actuators work was written in Python and a GUI was also created in order to make the software interact with the controller. This GUI contains various rostopics which are provided by Kinova that is used to control the actuators. Once, the software part is done, the integration between the hardware and the software part started. The connections between the hardware and the software were made and was successful.

After successful connections, various tests were carried out in order to check whether the hardware worked as expected when changes were made in the software. The final GUI that was created has four different types of control modes. They are position control, velocity control, cartesian control and trajectory control. Finally, in order to test these cases, few experiments were carried out successfully out of which the noticeable ones are the ability of the manipulator to draw shapes such as rectangles and parallelograms and also to write various letters. This brings the project to a successful end as the manipulator is able to achieve the desirable results as expected and can be employed in the spray-painting process.

CHAPTER 9:

FUTURE SCOPE

This manipulator can be used with the help of AR/VR technology and can be controlled by our hand as per our hand movements. With this, we can extend this as an industrial manipulator that can be controlled via the master-slave mechanism.

All the default shapes required, alphabets and numbers can be saved in the form of coordinates so that it will be easy for us to retrieve and use for spray painting.

Wireless control of the manipulator can be implemented by using Bluetooth or Wifi modules.

Though the current prototype needs human interaction for the control, it can be made fully autonomous with the incorporation of artificial intelligence and machine learning algorithms in the future. These are not incorporated in the current prototype due to hardware limitations and compatibility issues.

CHAPTER 10:

REFERENCES

1.Shabeeb, A. H. (n.d.). *Simulation of Spray Painting Using Articulated-Arm Robot*. https://www.researchgate.net/publication/311935812

2. Prakash Gujela, O., Gujela, V., Gujela, D. P., & Professor, A. (2007). Six DOF Spray Painting Robot Analysis. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering (An ISO, 3297*(9). https://doi.org/10.15662/IJAREEIE.2015.0409061

3. Madhuraghava, P., Fakruddin Basha, D., Reddy, V. S., & Sunil, N. (n.d.). *Modelling and Structural, Analysis of a 6-DOF Robot Spray Coating Manipulator*. https://doi.org/10.9790/1813-0701014856

4. Trigatti, G., Boscariol, P., Scalera, L., Pillan, D., & Gasparetto, A. (2018). A new path-constrained trajectory planning strategy for spray painting robots - rev.1. *International Journal of Advanced Manufacturing Technology*, *98*(9–12), 2287–2296. https://doi.org/10.1007/s00170-018-2382-2

5. Wang, Y. A., Xie, X. P., & Lu, X. H. (2020). Design of a double-nozzle air spray gun and numerical research in the interference spray flow field. *Coatings*, *10*(5). https://doi.org/10.3390/COATINGS10050475

6. Patel, R., Zeinali, M., & Passi, K. (2020, November). Deep Learning-based Robot Control using Recurrent Neural Networks (LSTM; GRU) and Adaptive Sliding Mode Control. https://doi.org/10.11159/cdsr21.113

7. Lees, D. S., & Chirikjian', G. S. (1996). *A Combinatorial Approach to Trajectory Planning for Binary Manipulators*.

8. Liu, G., Sun, X., Liu, Y., Liu, T., Li, C., & Zhang, X. (2021). Automatic spraying motion planning of a shotcrete manipulator. *Intelligent Service Robotics*. https://doi.org/10.1007/s11370-021-00348-9

9. Macfarlane, S., & Croft, E. A. (2003). Jerk-bounded manipulator trajectory planning: Design for real-time applications. *IEEE Transactions on Robotics and Automation*, *19*(1), 42–52. https://doi.org/10.1109/TRA.2002.807548

10. Zhang, S., Dai, S., Zanchettin, A. M., & Villa, R. (2020). Trajectory planning based on non-convex global optimization for serial manipulators. *Applied Mathematical Modelling*, *84*, 89–105. https://doi.org/10.1016/j.apm.2020.03.004

11. Wang, J., & Lei, X. (2018). On-Line Kinematical Optimal Trajectory Planning for Manipulator. *IHMSC 2018*, *1*, 319–323. https://doi.org/10.1109/IHMSC.2018.00080

12. Arunadevi, M., Jadhav, P. D., Ruchik M, A., Kishore Gandhi, P., Siddu K, V., & Vishnu Prasad, S. (2021). Machine Learning approach for Inverse Kinematics in Trajectory Planning of Pioneer 2 Manipulator with Cubic Spline Interpolation. *Proceedings - 5th International Conference on Computing Methodologies and Communication, ICCMC 2021*, 807–813. https://doi.org/10.1109/ICCMC51019.2021.9418318

13. Zha, X. F. (n.d.). *Optimal pose trajectory planning for robot manipulators*. www.elsevier.com/locate/mechmt

14. MATLAB Robotics Toolbox

15. https://en.wikipedia.org/wiki/Robot_kinematics

16. https://spotify.github.io/coordinator/

17. Drive link for all the codes and ROS packages used in the project:
https://drive.google.com/drive/folders/1K3Pm1HXFOVJD6r0fe2O37g_17-mnkTUA?usp=sharing