A. JAMES CLARK SCHOOL OF ENGINEERING

Bhargav Dandamudi
115473225

ENPM 662 Final Project Report

Introduction to Robot Modeling

Modeling of 5DOF -Mitsubishi RV-M1 Robot Manipulator

Submitted by:
Bhargav Dandamudi
115473225

Bhargav Dandamudi
115473225

INDEX

List of Figures

Bhargav Dandamudi
115473225

Modeling of 5DOF -Mitsubishi RV-M1 Robot manipulator

1. Abstract

Virtual modeling Simulator of RV-M1 robot has been developed in the MATLAB environment. The virtual system performs forward kinematics and inverse kinematics in addition to providing the simulation of tracking a line and a random trajectory. All programs are developed in MATLAB environment. Specifically, goals achieved by the project are as follows:

1. Kinematic Analysis: Developed forward kinematic equations for RV M1 manipulator. Simulated RVM1 model using robotics toolbox in MATLAB and using DH parameters. Developed Simscape model of RVM1 manipulator.
2. Developed inverse kinematics of the model using DH parameters, solved the inverse kinematic problem in MATLAB using different algorithms. Calculated Error between expected positions and calculated positions from different algorithm solution for joint values. Also calculated time taken by the system to solve the inverse kinematics of the manipulator.
3. GUI Development: Developed a GUI figure using Matlab GUIDE to plot robot frames as per different joint values given as input.
4. Trajectory Planning: Developed a simulation program for RV M1 end effector to follow a defined trajectory using inverse kinematics and calculate RMS error between ideal trajectory and robot trajectory output.
5. Velocity Kinematics: Developed the robot Jacobian and found Singularities of the robot, and simulated these joint values in the developed GUI figure to check the output.

2. Introduction

Robot manipulators are widely used in production, processing, product transportation, domestic services and other fields. The manipulator is composed of a serial of rigid links connected to each other with revolute or prismatic joints. The kinematics analysis on robot manipulator is an important part of robot study. Simulation is a process of designing a model of a real system and performing experiments with this model to understand the behavior of the system. Exploring a virtual model under simulated environments is the best way to learn about a real system. A virtual robot system can help one fully understand the controls and working of a robot. The system may also be helpful to design the path and plan the trajectory of a robot in an industrial environment or other robotics application. A graphical interface will be developed using Matlab Guide to illustrate the forward and inverse kinematics, allowing student or researcher to have hands-on of virtual graphical model that fully describe both the robot's geometry and the robot's motion in its workspace before to tackle any real task.

3. Mitsubishi RV-M1 robot

The Mitsubishi RV-M1 robot with 5 degree of freedom (DOF) is an excellent educational robot platform which many laboratories are equipped with. Developed Virtual model of RV-M1 robot in the MATLAB Environment, which would represent the robot as a three-dimensional model. The Mitsubishi RV-M1 with a full five degrees of freedom (DOF) is completely user friendly. It has a lifting capacity of 1kg and has excellent speed and repeatability. RV-M1 can perform virtually any task, from picking and moving components to complex manipulation sequences. The RV-M1 robot dimensions in (mm) are shown in Fig.1 which has five revolute joints Its external geometrical structure is similar to that of a human arm.

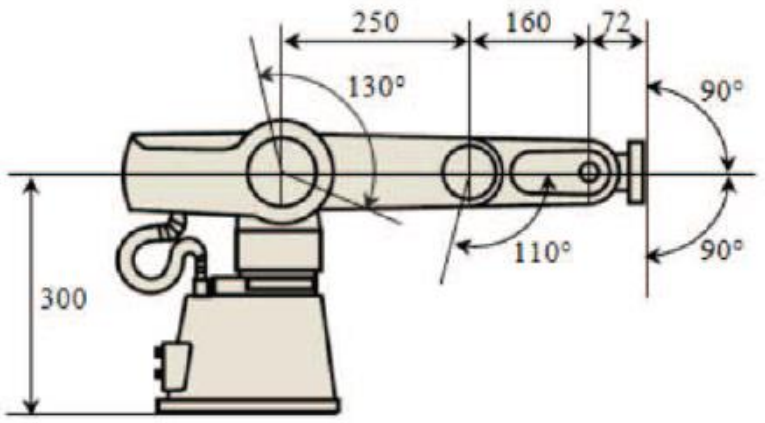


Figure 1. External Dimensions of the RV-M1 Manipulator.

Serial manipulators are robots made up by joining a number of rigid bodies. The joint limits in (°) and link dimensions in (mm) are described in Fig.1. All five joints are revolute, and described as follows:

Waist (J1): Rotation about $\mathbf{Z_1}$ with an angle θ_1
Shoulder (J2): Rotation about $\mathbf{Z_2}$with an angle θ_2
Elbow (J3): Rotation about $\mathbf{Z_3}$ with an angle θ_3
Wrist Pitch (J4): Rotation about $\mathbf{Z_3}$ with an angle θ_4
Wrist Roll (J5): Rotation about $\mathbf{Z_4}$ with an angle θ_5

The positive sense of revolution is clockwise. The Mitsubishi RV-M1 manipulator shown in Fig.2 has five degrees of freedom (5DOF). Following are the joint limits as described in:

$$-150° \leq \boldsymbol{\theta_1} \leq 150°$$
$$-30° \leq \boldsymbol{\theta_2} \leq 100°$$
$$-110° \leq \boldsymbol{\theta_3} \leq 0°$$
$$-90° \leq \boldsymbol{\theta_4} \leq 90°$$
$$-180° \leq \boldsymbol{\theta_5} \leq 180°$$

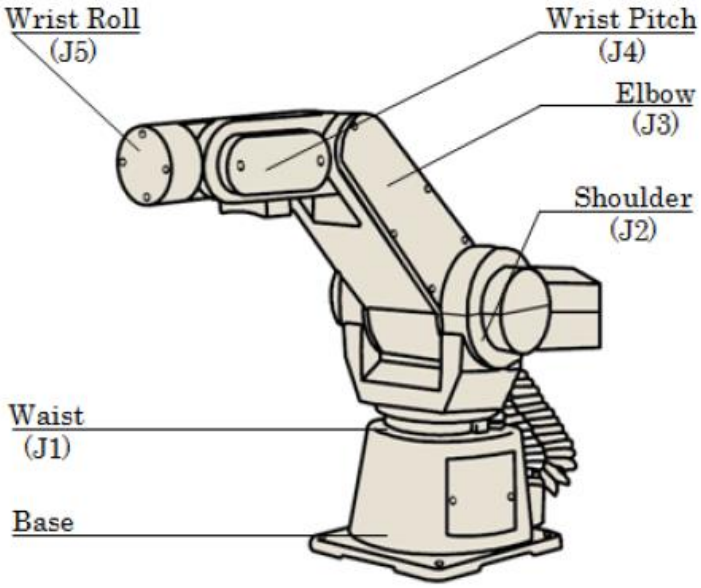


Figure 2. Structure of RVM1 Robot arm

Whole model is much flexible to be developed in Solidworks than Simscape. Solid shapes as in real RVM1 model can be developed in Solidworks when compared to use simple blocks in Simscape, later whole CAD model is imported in MATLAB Simscape. Block Diagram can be found in Appendix 2.2. We need to use "smimport('Assem1.xml')" command in MATLAB environment to import all the files from the folder to Simscape

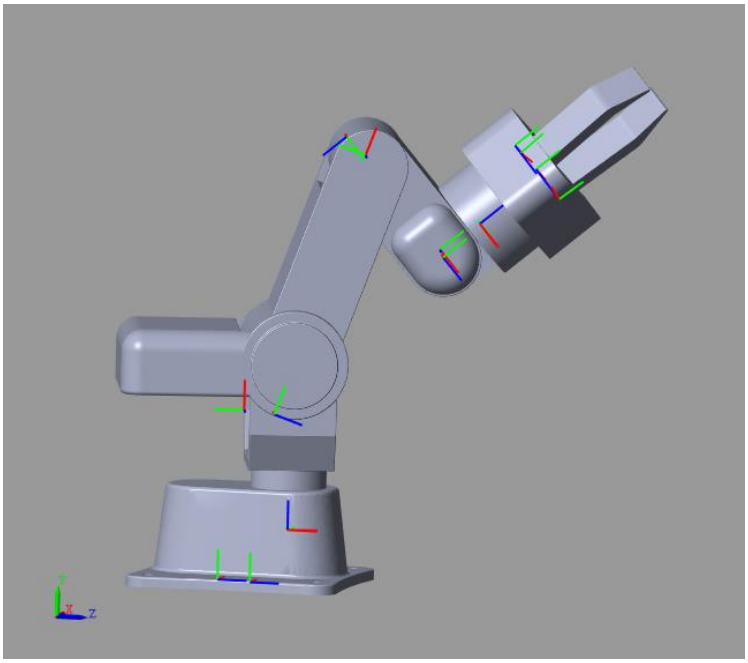


Figure 3. Simscape model of the RVM1 Model.

4. RVM1 Forward Kinematics:

The RV-M1 robot with 5DOF has all revolute joints. In the forward kinematics, the angles of the revolute joints are given. The task is to find the end-effectors' position and orientation. In the inverse kinematics, the task is to obtain the joint variables corresponding to a given end effectors' orientation and position. The inverse kinematics problem is much more complex than the forward kinematics. Denavit-Hartenberg (DH) notations are widely used to describe the kinematics model of a robot. If the DH parameters of a robot are known, we can easily simulate the forward kinematics of the robot. Every robot can be described by DH parameters. Frames for the DH parameters can be found in [1].

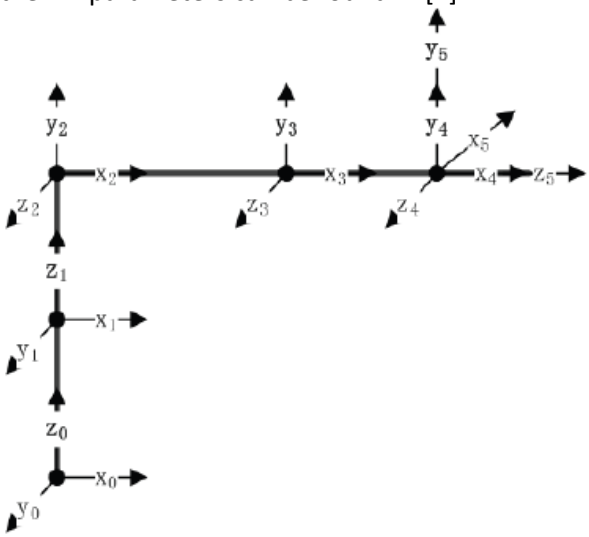


Figure 4. Coordinate Frames for RVM1 Manipulator.

All five joints of the RV-M1 robot are revolute and rotate about Z_i (i =1, 2,..5). The DH parameters of the RV-M1 robot are given in Figure 5, where the parameters θ_i, d_i ,a_i and α_i are called the joint angle, link offset, link length, and link twist, respectively. Usually, three of these four are fixed parameters while one is a variable. For a revolute joint, the joint variable is considered to be θ_i, while for a prismatic joint, it will be d_i. The RVM1 robot have all revolute joints, so the parameters d_i , a_i and α_i are fixed and only θ_i is variable. The joint limits of the RV-M1 robot are shown in the last column of Figure 5.

Joint	θ_i *(deg)*	d_i *(mm)*	a_i *(mm)*	α_i *(deg)*	*Joint Range*
1	θ_1	152	0	90	-150~150
2	θ_2	0	250	0	-30~100
3	θ_3	0	160	0	-110~0
4	θ_4 +90	0	0	90	0~180
5	θ_5 - 90	72	0	0	-180~180

Figure 5. DH parameters of RVM1 manipulator

Calculating Positions for each joint to calculate end effector position using given theta values (Joint angles). According to DH coordinate system, the homogeneous transformation matrix. the relative position of a link (i-1) with respect to an adjacent link (i) and vice versa, is defined by forward/backward homogeneous transformation matrices made of two translations and two rotations along and about the local X and Z axes, and defined as follows:

$$^{i-1}_{i}\boldsymbol{T} = Transl_X[a_{i-1}].Rot_X[\alpha_{i-1}].Transl_Z[d_i].Rot_Z[\theta_i] \quad \text{..... (1)}$$

$$^{i}_{i-1}\boldsymbol{T} = Trans_Z[-d_i].Rot_Z[\theta_i].Trans_X[a_{i-1}].Rot_X[-\alpha_{i-1}] \quad \text{..... (2)}$$

Where:

$$^{i-1}_{i}\boldsymbol{T} = \begin{bmatrix} cos\theta_i & -sin\theta_i & 0 & a_{i-1} \\ cos\alpha_{i-1}sin\theta_i & cos\alpha_{i-1}cos\theta_i & -sin\alpha_{i-1} & -d_i sin\alpha_{i-1} \\ sin\alpha_{i-1}sin\theta_i & sin\alpha_{i-1}cos\theta_i & cos\alpha_{i-1} & d_i cos\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{..... (3)}$$

$$^{i}_{i-1}\boldsymbol{T} = \begin{bmatrix} cos\theta_i & cos\alpha_{i-1}sin\theta_i & sin\alpha_{i-1}sin\theta_i & -a_{i-1}cos\theta_i \\ -sin\theta_i & cos\alpha_{i-1}cos\theta_i & -sin\alpha_{i-1}cos\theta_i & a_{i-1}sin\theta_i \\ 0 & -sin\alpha_{i-1} & cos\alpha_{i-1} & -d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{..... (4)}$$

The position of the wrist in the reference coordinate frame described by the transformation matrix T05 is obtained by taking the dot product of all the transformation matrices up to the tool-frame.

$$^{0}_{5}\boldsymbol{T} = {}^{0}_{1}T.{}^{1}_{2}T.{}^{2}_{3}T.{}^{3}_{4}T.{}^{4}_{5}T \quad \text{..... (5)}$$

$$^{0}_{5}\boldsymbol{T} = \begin{bmatrix} t_{11} & t_{12} & t_{13} & \boldsymbol{t_{14}} \\ t_{21} & t_{22} & t_{23} & \boldsymbol{t_{24}} \\ t_{31} & t_{32} & t_{33} & \boldsymbol{t_{34}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{..... (6)}$$

Where the first 3x3 matrix represents the orientation of the wrist, and the vector $\boldsymbol{[\, t_{14}\ t_{24}\ t_{34}]^T}$ denotes the position $\boldsymbol{[\, x_w\ y_w\ z_w]^T}$ of the wrist in the reference frame in terms of joint variables. Including the gripper transformation matrix, given by:

$$^{5}_{6}\boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_G \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{..... (7)}$$

$$ {}^{0}_{6}T = {}^{0}_{5}T . {}^{5}_{6}T $$

$$ {}^{0}_{6}T = \begin{bmatrix} t_{11} & t_{12} & t_{13} & x_G \\ t_{21} & t_{22} & t_{23} & y_G \\ t_{31} & t_{32} & t_{33} & z_G \\ 0 & 0 & 0 & 1 \end{bmatrix} $$ …… (8)

Where the vector $[\, x_G \; y_G \; z_G]^T$ denotes the position of the gripper in the reference frame in terms of joint variables. Using Matlab for further calculation, we can find

$$ t_{11} = c_1 c_5 c_{234} - s_1 s_5 $$
$$ t_{12} = -c_1 s_5 c_{234} - s_1 c_5 $$
$$ t_{13} = c_1 s_{234} $$
$$ x_G = d_G t_{13} + x_W = c_1 (a_2 c_2 + a_3 c_{23} + d_G s_{234}) $$
$$ t_{21} = -s_1 c_5 c_{234} - c_1 s_5 $$
$$ t_{22} = s_1 s_5 c_{234} - c_1 c_5 $$
$$ t_{23} = -s_1 s_{234} $$
$$ y_G = d_G t_{23} + y_W = -s_1 (a_2 c_2 + a_3 c_{23} + d_G s_{234}) $$
$$ t_{31} = c_5 s_{234} $$
$$ t_{32} = -s_5 s_{234} $$
$$ t_{33} = -c_{234} $$
$$ z_G = d_G t_{33} + z_W = d_1 - d_G c_{234} + a_2 s_2 + a_3 s_{23} $$

With:

$$ \begin{cases} c_i = cos\theta_i \\ s_i = sin\theta_i \\ c_{ij} = cos(\theta_i + \theta_j) \\ s_{ij} = sin(\theta_i + \theta_j) \\ c_{ijk} = cos(\theta_i + \theta_j + \theta_k) \\ s_{ijk} = sin(\theta_i + \theta_j + \theta_k) \end{cases} $$

……(9)

5. Simulations of Forward Kinematics

There are two ways of doing forward kinematics:

1. Using Robotics Toolbox to create robot by creating rigid body chain, Refer to Appendix 2 Program 1 (RVM1Frames)as show in Figure 6

2. Use DH parameter and Homogenous transformation as discussed in 4. As show in Figure 7.

Using following theta values for plotting RVM1:

```
theta_1 = 0;
theta_2 = pi/6;
theta_3 = pi/2;
theta_4 = pi/3;
theta_5  = 0;
```

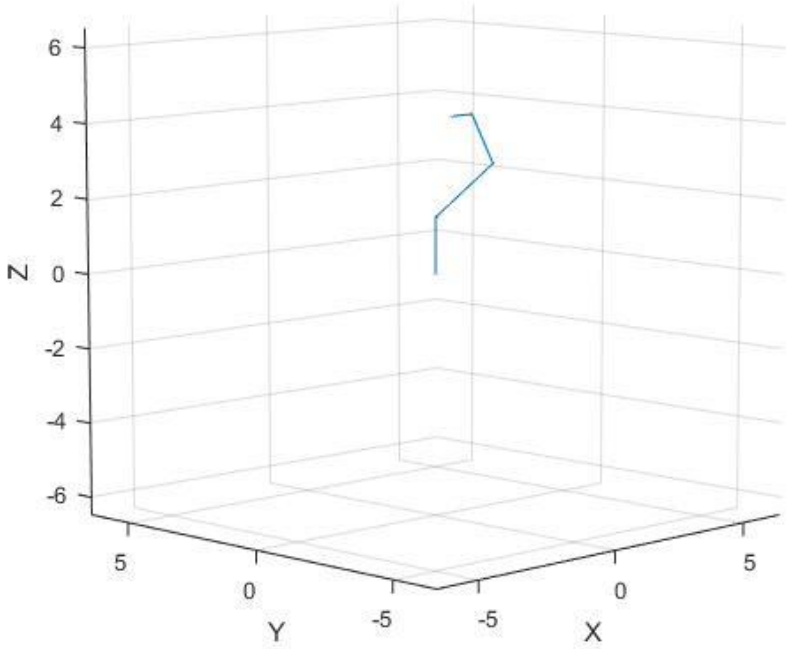


Figure 6. Robot Plot using Robotics Toolbox

Refer to Appendix program 1 to 6 for forward kinematics Matlab programs. We have divided all link lengths with 100 for better figure analyses in plots. Accordingly changed the dh parameters as well. Use Program 1 for plotting RVM1 using robotics tool box. Program 2 is snippet for calculating DH matrix as in Equation 4. Program 3 takes input as joint values (theta) and calculates the position of each joint, hence the end effector and gripper location. Figure 7 shows the output of program 4 output.

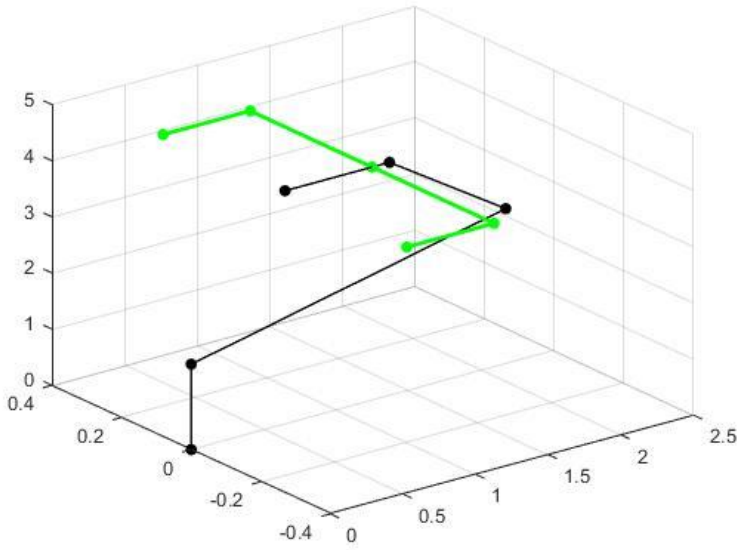


Figure 7. Plotting RVM1 and showing gripper using DH parameters and gripper locations.

Also, we calculated the error between positions being calculated separately by DH and robotics toolbox. Error came out to be 0 i.e., our calculations and simulations are perfectly correct, hence verified. Plotting them together demonstrates error as zero as show in figure 8.

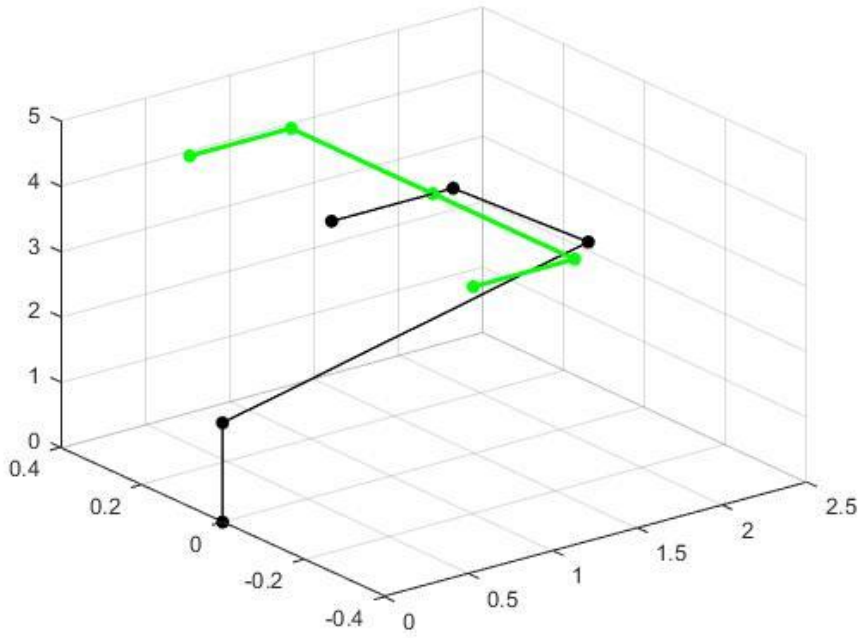


Figure 8. Plotting RVM1 using both techniques, and demonstrating Zero error (Program 5 in Appendix).

6. Inverse Kinematics

6.1 Calculating mathematically using geometry.

Having the expression of the end-effectors' coordinates with respect to the reference frame, the next step is to search the inverse relationship giving joint coordinates in terms of the end-effectors' pose.

We can clearly see from equation (9).

$$\boldsymbol{\theta_1} = atan2(x_G, -y_G) \quad \text{..... (10)}$$

From (6), the wrist coordinates in the reference frame are obtained:

$$\begin{aligned} \boldsymbol{x_W} &= t_{14} = c_1(a_2c_2 + a_3c_{23}) \\ \boldsymbol{y_W} &= t_{24} = -s_1(a_2c_2 + a_3c_{23}) \\ \boldsymbol{z_W} &= t_{34} = d_1 + a_2s_2 + a_3s_{23} \end{aligned} \quad \text{..... (11)}$$

Using 11, multiplying $\mathbf{x_w}$ with cosθ_1 and $\mathbf{y_w}$ by -sin(θ_1) adding up the resulting equations yields:

$$a_3c_{23} = x_Wc_1 - y_Ws_1 - a_2c_2$$

..... (12)

Squaring and adding we get

$$\alpha sin\theta_2 + \beta cos\theta_2 = \gamma$$

..... (13)

This is recurrent equation in kinematics of robot manipulators.

$$\boldsymbol{\theta_2} = atan2\left(\gamma, \pm\sqrt{r^2 - \gamma^2}\right) - atan2(\alpha, \beta) \quad \text{..... (14)}$$

With:

$$\begin{cases} \alpha = z_W - d_1 \\ \beta = x_Wc_1 - y_Ws_1 \\ \gamma = \dfrac{a_2^2 - a_3^2 - \alpha^2 - \beta^2}{2a_2} \\ r = \sqrt{\alpha^2 + \beta^2} \end{cases}$$

Also from (11) and (12)

$$\boldsymbol{\theta_3} = atan2(\beta - a_2c_2\,, \alpha - a_2s_2) - \boldsymbol{\theta_2} \quad \text{..... (15)}$$

To calculate θ_4, an angle φ is such that:

$$\varphi = \theta_2 + \theta_3 + \theta_4 \quad \text{..... (16)}$$

Introducing the angle φ is meant to keep the gripper orientation fixed during object manipulations but also to allow finding a closed form solution [15]. That means the gripper orientation is no longer relative to the elbow but to the ground. Given the angle φ , and having θ_2 and θ_3 , the angle θ_4 is deduced from (16):

$$\theta_4 = \varphi - \theta_2 - \theta_3 \quad \text{..... (17)}$$

All solutions and calculations are referred from [2].

6.2 Calculating position using Different Algorithms on Matlab.

L MATLAB® supports two algorithms for achieving an IK solution: the BFGS projection algorithm and the Levenberg-Marquardt algorithm. Both algorithms are iterative, gradient-based optimization methods that start from an initial guess at the solution and seek to minimize a specific cost function. If either algorithm converges to a configuration where the cost is close to zero within a specified tolerance, it has found a solution to the inverse kinematics problem. However, for some combinations of initial guesses and desired end effector poses, the algorithm may exit without finding an ideal robot configuration. To handle this, the algorithm utilizes a random restart mechanism. If enabled, the random restart mechanism restarts the iterative search from a random robot configuration whenever that search fails to find a configuration that achieves the desired end effector pose. These random restarts continue until either a qualifying IK solution is found, the maximum time has elapsed, or the iteration limit is reached.

6.2.1 BFGS Gradient Projection

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) gradient projection algorithm is a quasi-Newton method that uses the gradients of the cost function from past iterations to generate approximate second-derivative information. The algorithm uses this second-derivative information in determining the step to take in the current iteration. A gradient projection method is used to deal with boundary limits on the cost function that the joint limits of the robot model create. The direction calculated is modified so that the search direction is always valid.

This method is the default algorithm and is more robust at finding solutions than the Levenberg-Marquardt method. It is more effective for configurations near joint limits or when the initial guess is not close to the solution. If your initial guess is close to the solution and a quicker solution is needed, consider the Levenberg-Marquardt method.

Solver Parameters for BFGS Gradient Projection

The solver parameters for the BFGS algorithm have the following fields:

- MaxIterations — Maximum number of iterations allowed. The default is 1500.
- MaxTime — Maximum number of seconds that the algorithm runs before timing out. The default is 10.
- GradientTolerance — Threshold on the gradient of the cost function. The algorithm stops if the magnitude of the gradient falls below this threshold. Must be a positive scalar.
- SolutionTolerance — Threshold on the magnitude of the error between the end-effector pose generated from the solution and the desired pose. The weights specified for each component of the pose in the object are included in this calculation. Must be a positive scalar.
- EnforceJointLimits — Indicator if joint limits are considered in calculating the solution. JointLimits is a property of the robot model in robotics.RigidBodyTree. By default, joint limits are enforced.
- AllowRandomRestarts — Indicator if random restarts are allowed. Random restarts are triggered when the algorithm approaches a solution that does not satisfy the constraints. A randomly generated initial guess is used. MaxIteration and MaxTime are still obeyed. By default, random restarts are enabled.
- StepTolerance — Minimum step size allowed by the solver. Smaller step sizes usually mean that the solution is close to convergence. The default is 10–14.

6.2.2 Levenberg-Marquardt

The Levenberg-Marquardt (LM) algorithm variant used in the Inverse Kinematics class is an error-damped least-squares method. The error-damped factor helps to prevent the algorithm from escaping a local minimum. The LM algorithm is optimized to converge much faster if the initial guess is close to the solution. However, the algorithm does not handle arbitrary initial guesses well. Consider using this algorithm for finding IK solutions for a series of poses along a desired trajectory of the end effector. Once a robot configuration is found for one pose, that configuration is often a good initial guess at an IK solution for the next pose in the trajectory. In this situation, the LM algorithm may yield faster results. Otherwise, use the BFGS Gradient Projection instead.

Solver Parameters for Levenberg-Marquardt

The solver parameters for the LM algorithm have the following extra fields in addition to what the BFGS Gradient Projection method requires:

- ErrorChangeTolerance — Threshold on the change in end-effector pose error between iterations. The algorithm returns if the changes in all elements of the pose error are smaller than this threshold. Must be a positive scalar.
- DampingBias — A constant term for damping. The LM algorithm has a damping feature controlled by this constant that works with the cost function to control the rate of convergence. To disable damping, use the UseErrorDamping parameter.
- UseErrorDamping — 1 (default), Indicator of whether damping is used. Set this parameter to false to disable dampening.

7.Velocity Kinematics and Jacobian

As for position kinematics analysis being broken up into forward and inverse problems, so is the velocity analysis:

- Given the joint variables velocities, the 'end effectors' velocity is calculated.
- Given the end-effectors velocity, the joint variables velocities are calculated.

Let x be the vector of the end-effectors' position and θ_1 the vector of joint variables, the relationship between the time derivative of the two quantities is defined as follows:

$$\dot{x} = J\dot{\theta}$$

…… (18)

Where J is a linear transformation called the robot Jacobian, which allows us to go back and forth from joint velocities to Cartesian velocities. The Jacobian is calculated column by column using the generating vectors method, where ci(θ) is the ith column defined as:

$$c_i(\theta) = \begin{bmatrix} {}^{0}_{i-1}\hat{k} \times {}^{0}_{i-1}d_n \\ {}^{0}_{i-1}\hat{k} \end{bmatrix}$$

…….. (19)

Where X is the cross product and K cap and d_n denote respectively the joint axis (Z) unit vector and the position of the origin of the frame (i-1) expressed in the reference frame and d_n is the translational submatrix of T_n^{i-1} .

$${}^{0}_{i-1}\hat{k} = {}^{0}_{i-1}R.\,{}^{i-1}_{i-1}\hat{k}$$

…… (20)

Where R is the rotational submatrix of T and Kcap is the direction of the joint (i-1) unit vector expressed in the same frame. Hence, the robot Jacobian:

$$J = \begin{bmatrix} {}^{0}_{0}\hat{k} \times {}^{0}_{0}d_5 & {}^{0}_{1}\hat{k} \times {}^{0}_{1}d_5 & {}^{0}_{2}\hat{k} \times {}^{0}_{2}d_5 & {}^{0}_{3}\hat{k} \times {}^{0}_{3}d_5 & {}^{0}_{4}\hat{k} \times {}^{0}_{4}d_5 \\ {}^{0}_{0}\hat{k} & {}^{0}_{1}\hat{k} & {}^{0}_{2}\hat{k} & {}^{0}_{3}\hat{k} & {}^{0}_{4}\hat{k} \end{bmatrix}$$

…….. (21)

Through symbolic calculation using MATLAB and Mathematica, the columns of the Jacobian read:

$$c_1 = \begin{bmatrix} s_1(a_2c_2 + a_3c_{23} + d_Gs_{234}) \\ c_1(a_2c_2 + a_3c_{23} + d_Gs_{234}) \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}; c_2 = \begin{bmatrix} 0 \\ -a_2c_2 - a_3c_{23} - d_Gs_{234} \\ 0 \\ 0 \\ 0 \\ -1 \end{bmatrix}$$

$$c_3 = \begin{bmatrix} 0 \\ 0 \\ a_2c_1 + a_3c_{13} + d_Gs_{234} \\ -s_1 \\ -c_1 \\ 0 \end{bmatrix}; c_4 = \begin{bmatrix} 0 \\ 0 \\ a_3c_1 + d_Gs_{234} \\ -s_1 \\ -c_1 \\ 0 \end{bmatrix}; c_5 = \begin{bmatrix} 0 \\ 0 \\ d_Gs_1 \\ -s_1 \\ -c_1 \\ 0 \end{bmatrix}$$

…… (22)

To find the inverse relationship between joint velocities and the end-effectors' velocity, equation (18) is multiplied for both sides by the inverse of the Jacobian:

$$\dot{\theta} = J^{-1}\dot{x}$$

…… (23)

The first problem to arise is that the Jacobian matrix of the robot in question is non-square and thus non-invertible. An alternative is to use the Pseudo-inverse instead:

$$J^{+} = (J^{T}.J)^{-1}.J^{T}$$

…… (24)

And equation (23) becomes:

$$\dot{\theta} = J^{+}\dot{x}$$

…….. (25)

The use of the pseudo-inverse will find its mathematical benefits while facing singularities, for even if the Jacobian is singular, its pseudo-inverse can still be calculated. However, it is highly important in robotics to find the joint values that render the Jacobian singular and avoid them in real tasks.

8. Singularities

Typically, there are two kinds of singularities to deal with in robot arms modelling, external and internal. External singularities occur when the position of the object to manipulate is at the boundary of the workspace or completely out of reach. Internal singularities are those who make the robot unable to reach an object even if it is within the workspace. If the Jacobian matrix was square, then it would have

been easy to identify singularities by putting the determinant of the Jacobian equal to zero. In our case, the Jacobian is a non-square matrix.
For the first set of singularities, the position vector denoted vector [xG yG zG]^T defined by equations (8) is needed. By partially differentiating the vector in question with respect to joint variables, the 3x4 Jacobian of the position vector is obtained and being defined as:

$$\hat{J} = \begin{bmatrix} \frac{\partial x_G}{\partial \theta_i} \\ \frac{\partial y_G}{\partial \theta_i} \\ \frac{\partial z_G}{\partial \theta_i} \end{bmatrix} ; \quad i = 1,2,3,4$$

…… (26)

This gives:

$$\hat{J} = \begin{bmatrix} \hat{j}_{11} & \hat{j}_{12} & \hat{j}_{13} & \hat{j}_{14} \\ \hat{j}_{21} & \hat{j}_{22} & \hat{j}_{23} & \hat{j}_{24} \\ \hat{j}_{31} & \hat{j}_{32} & \hat{j}_{33} & \hat{j}_{34} \end{bmatrix}$$

…… (27)

The rank of **J** is 3, which means that **J** has 3 linearly independent rows. The elements of the Matrix in equation (30) are given by:

$$\hat{j}_{11} = -s_1(a_2c_2 + a_3c_{23} + d_Gs_{234})$$
$$\hat{j}_{12} = -c_1(a_2s_2 + a_3s_{23} - d_Gc_{234})$$
$$\hat{j}_{13} = -c_1(a_3s_{23} - d_Gc_{234})$$
$$\hat{j}_{14} = d_Gc_1c_{234}$$
$$\hat{j}_{21} = -c_1(a_2c_2 + a_3c_{23} + d_Gs_{234})$$
$$\hat{j}_{22} = s_1(a_2s_2 + a_3s_{23} - d_Gc_{234})$$
$$\hat{j}_{23} = s_1(a_3s_{23} - d_Gc_{234})$$
$$\hat{j}_{24} = -d_Gs_1c_{234}$$
$$\hat{j}_{31} = 0$$
$$\hat{j}_{32} = d_Gs_{234} + a_2c_2 + a_3c_{23}$$
$$\hat{j}_{33} = d_Gs_{234} + a_3c_{23}$$
$$\hat{j}_{34} = d_Gs_{234}$$

Let's divide the Jacobian matrix in 3 parts as below:

$$\hat{J}_1 = \begin{bmatrix} \hat{j}_{11} & \hat{j}_{12} & \hat{j}_{13} \\ \hat{j}_{21} & \hat{j}_{22} & \hat{j}_{23} \\ \hat{j}_{31} & \hat{j}_{32} & \hat{j}_{33} \end{bmatrix}, \hat{J}_2 = \begin{bmatrix} \hat{j}_{11} & \hat{j}_{12} & \hat{j}_{14} \\ \hat{j}_{21} & \hat{j}_{22} & \hat{j}_{24} \\ \hat{j}_{31} & \hat{j}_{32} & \hat{j}_{34} \end{bmatrix}, \hat{J}_3 = \begin{bmatrix} \hat{j}_{11} & \hat{j}_{13} & \hat{j}_{14} \\ \hat{j}_{21} & \hat{j}_{23} & \hat{j}_{24} \\ \hat{j}_{31} & \hat{j}_{33} & \hat{j}_{34} \end{bmatrix}$$

…….. (28)

By calculating the determinants of each J above sub matrices equal to zero, simultaneous solutions are searched. This yields to the following pair solution:

$$\{(\sin\theta_3 = 0) \wedge (\cos\theta_4 = 0)\}$$

…… (29)

Amongst all the values of θ_3 and θ_4 that satisfy equation (29), only those lying within the joint limits range are of interest for us. Taking into account the third and fourth joint limits, equation (29) gives the first singularity set:

$$\left\{\theta_3 = 0 \wedge \theta_4 = \frac{\pi}{2}\right\}$$

…….. (30)

The other set of singularities is found by fixing one joint at a time and varying all the others, beginning from joint2 to joint4. By substituting the lower limit of θ_2 in submatrices in equation (28), then solving for the determinants equal to zero, the second singularity occurs at:

$$\left\{\theta_2 = -\frac{\pi}{6} \wedge \theta_4 = \frac{\pi}{2}\right\}$$

…… (31)

This procedure is repeated for all joints up to the fourth one, each time substituting both the lower and upper limits of the joint in question and varying the others.

This can be easily implemented in Developed GUI which will be further discussed in section 10.

9. Inverse Kinematics, Line and Trajectory Follower Simulations

Refer to Program 8 in appendix, it calculates the array of joint values using points as input the system. Inverse kinematics solution is found using Levenberg Marquardt Algorithm for quicker results. We can change it to BFGS Algorithm by un commenting the following line.

A line segment is plotted using the start and end points given as input to the system. We can use program 7. Sample points are inside comments, we can use them to visualize the line. This program is being used

in Program 8 for input values. Program 10 is used to plot all the positions calculated using inverse kinematics solution parameters as input to robotic toolbox.

User can directly use program 12 for each plot, Outputs for

```
startPoint = [1.1,0,1.3];
endPoint = [2.3,0,3.33];
```

is as shown below:

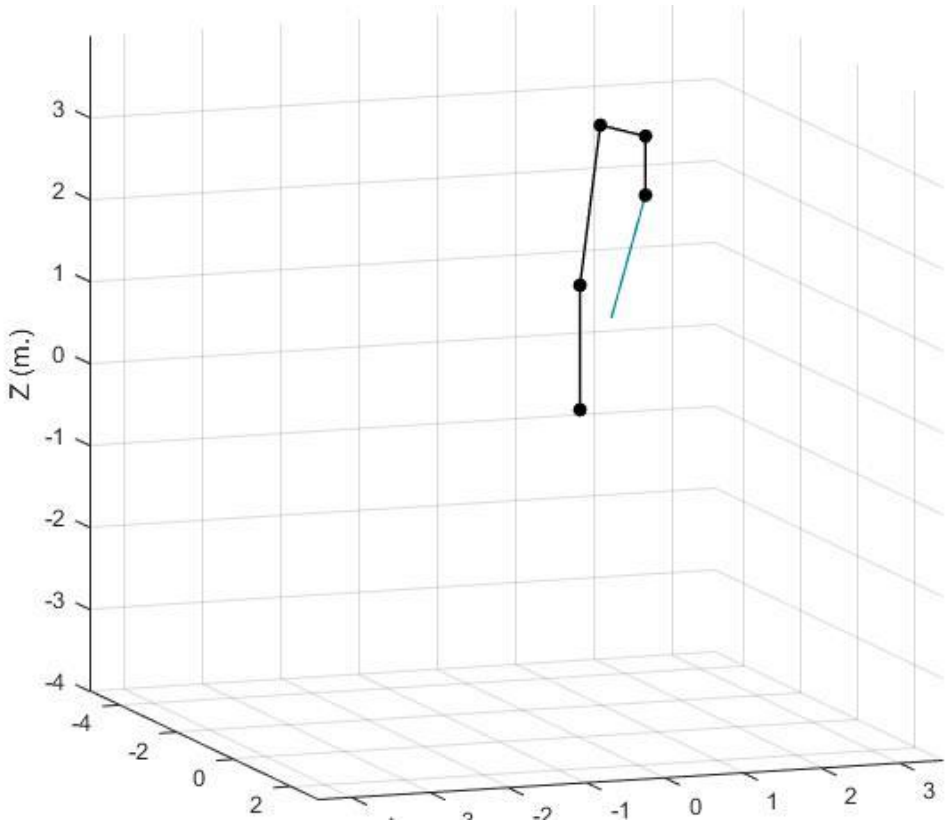


Figure 9. Demonstrating RVM1 following a line segment joining two points (Program 12 in Appendix))

We can easily make the observation that the time taken to calculate the inverse kinematics is a lot longer than calculating forward kinematics. For precise calculation of time we are using tic tac login program 12 and same for following trajectory.

Note: All points in line segment and trajectory must lie inside the workspace of the RVM1 robot. Robot will stretch out completely trying to reach the point but won't be able to reach the exact coordinate which is out of reach, we can demonstrate its singularity position using Program 15 GUI.

We can similarly track a random trajectory, for fun part let's plot letter D as shown in figure 10.

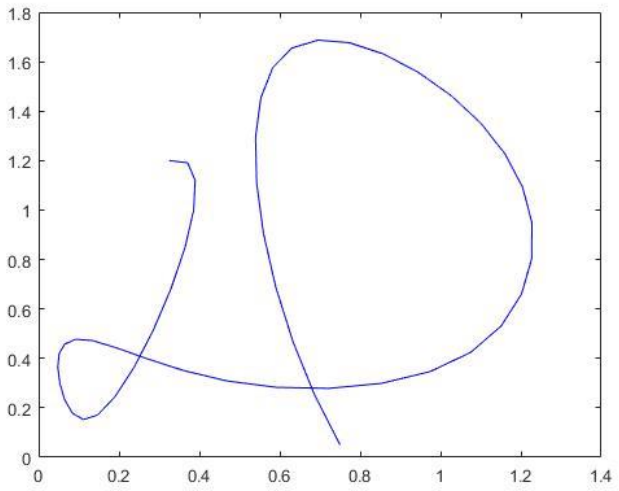


Figure 10 Trajectory which needs to plotted by RVM1

Plotting output as follows

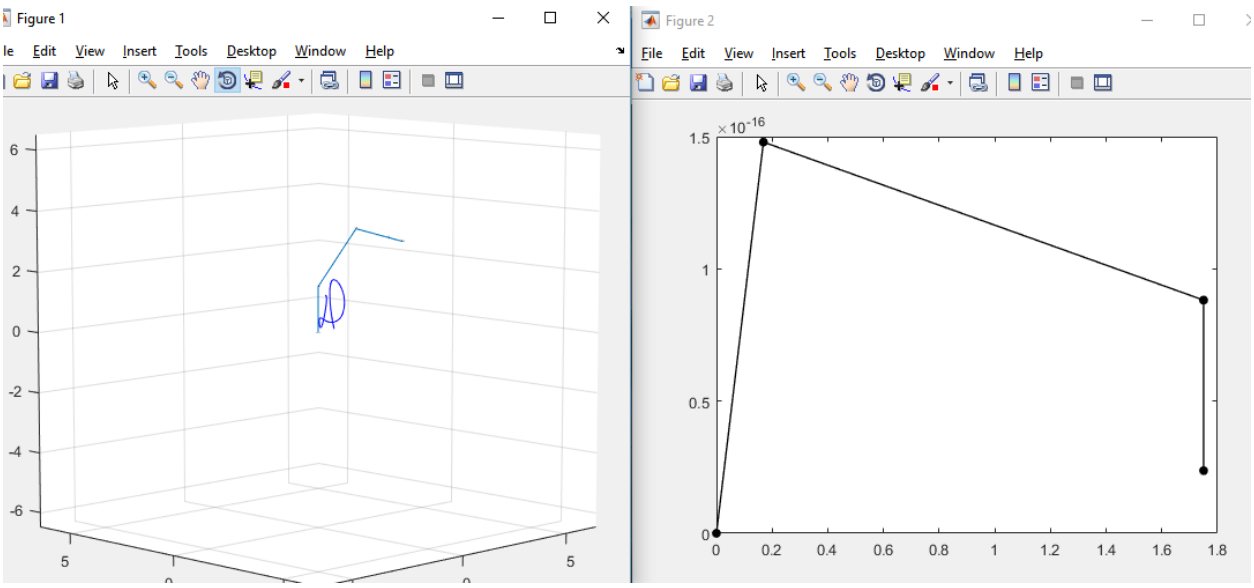


Figure 10 Robot Final position and Initial position

Time delay for each plot is kept high, so that user can notice how it is plotting each position values calculated by programs. Time taken to track the trajectory is about 240 sec on an average, it works on Levenberg-Marquardt algorithm.

10. Graphic User Interface

GUI is developed using MATLAB Guide toolbox. We can enter all the Joint values inside the edit boxes, (refer to program 14 and 15 in Appendix). We can plot our manipulator depending on theta values given to it, also to bring the RVM1 to its home position we can use Home button.

Output when we run the program15:

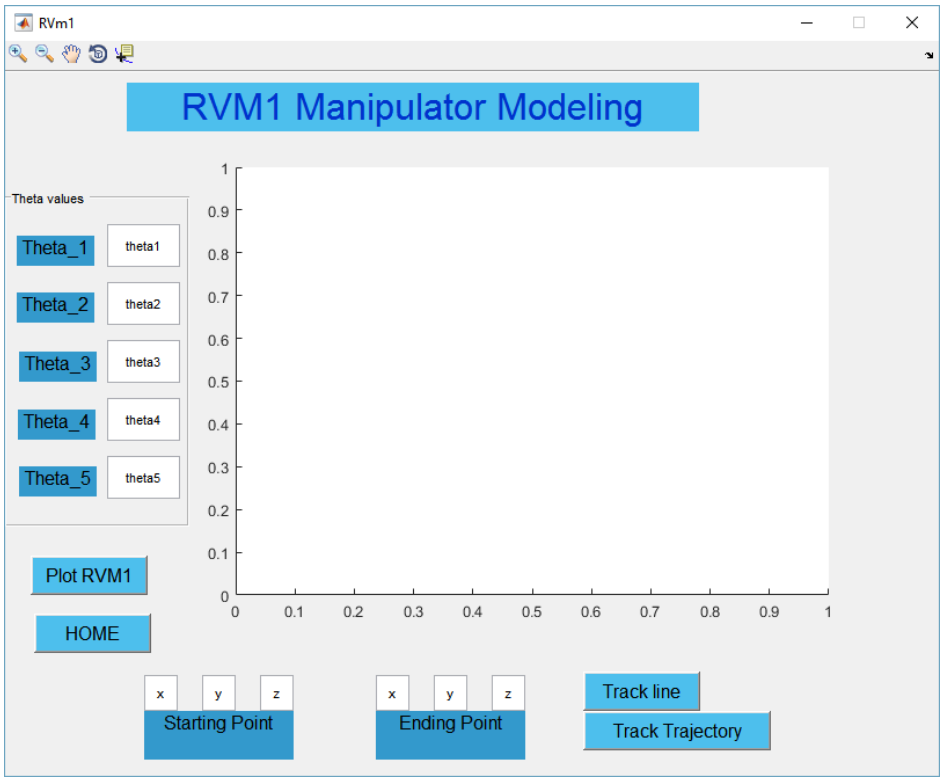


Figure 11. GUI Screen without any input.

We need to draw the gui as shown above and change the callback functions accordingly in Matlab editor. We can tag each block as per need and write its callback function. Associated all the programs developed so far to push buttons tags, so that whenever they are used it calls the program we need. One can used [9] for more detail knowledge on developing GUI using Matlab.

We Can enter values of starting Point and Ending point (one must enter values within robot workspace), GUI will plot the respective plots when Track line push button is pressed, Also Track the trajectory push button moves the robot end effector in the given trajectory.

Also for analyzing the plotted graphs, we can zoon in, zoom out and pan the plotted figures.
Further figures demonstrate the use of GUI and how it plots the RVM1 end effector.

When we input values of theta and press Plot RVM1 push button:

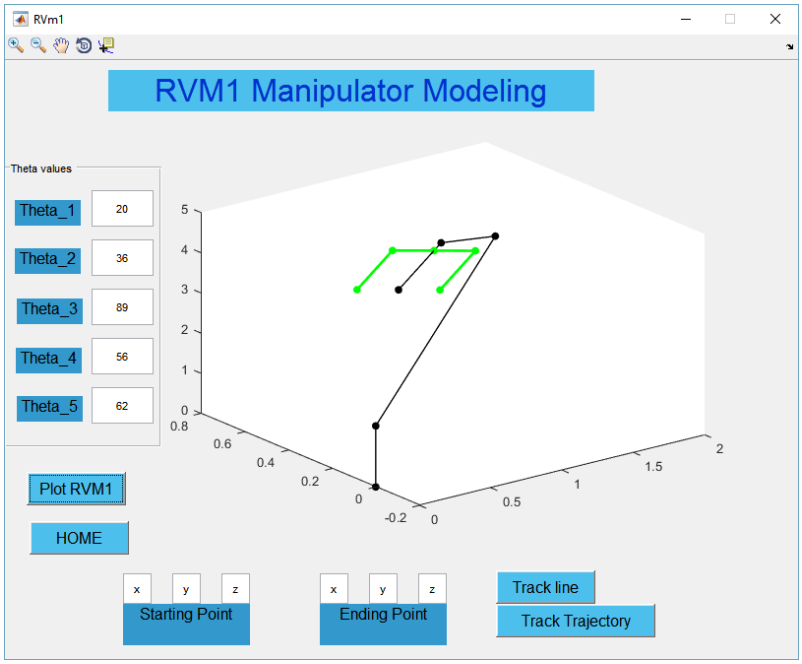


Figure 12. Demonstrating use of Plot RVM1 as per given Joint Values

When input value for Starting and Ending point and press Track line:

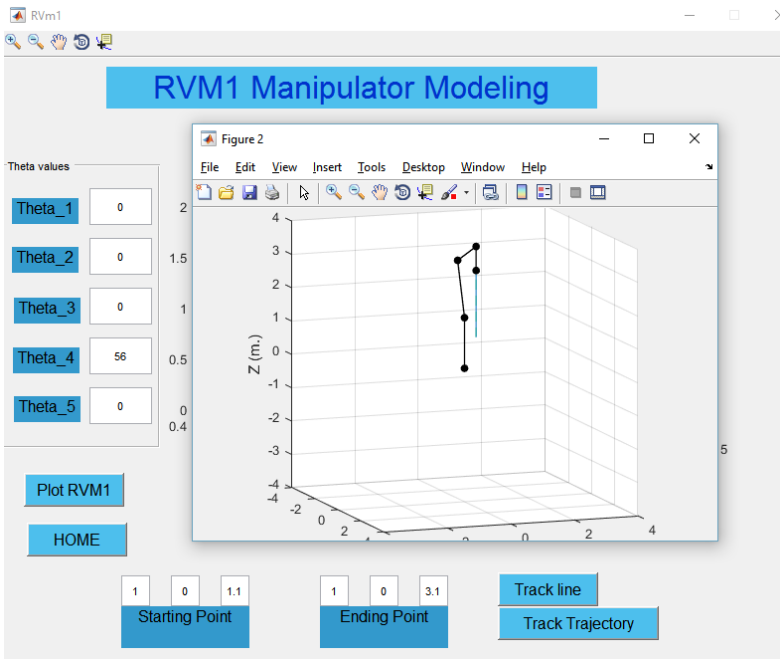


Figure 13. Demonstrating use of Track line Push button

Similarly, one can use Track Trajectory push button and see how it tracks the trajectory.

11. Conclusion

The kinematic model of the Mitsubishi RV-M1 robot with 5 DOF is developed with user friendly GUI. The forward and inverse kinematic equations are obtained. The interactive simulation system based on MATLAB is built, which is very useful to give a geometrical insight of the robot in its workspace, and allows the students or researcher to visualize the joints and movements of the robot. Simulation results verify the effectiveness of kinematics equations. The performance investigation of this method to the real RV-M1 robot will be done as a future scope of this work. Also developed solid parts in Solidworks can be implemented in Robotics Toolbox to demonstrate real model of RVM1 than stick figure. Future scope of the project can be as follows:

1. Commercial simulation software for robot system are very expensive. Therefore, a kinematics simulation of Mitsubishi RV-M1 robot using MATLAB can be very useful. This software enables a user to learn the kinematics of a Mitsubishi robot by displaying the three-dimensional robot model. This project can be used in class to explain forward and inverse kinematics, trajectory tracking and to explain singularities.

2. Virtual Reality software can be developed for better visualization of RVM1 robot.

3. The embedded remote access control and monitoring with virtual reality modelling is also another further research that will be put under consideration. With a virtual model at hand, future work is aimed on visual serving and hardware-in-loop control for robotic arm that utilized in assembly processes.

4.Simulation to find optimum path to reach a specific point in given workspace of Robot.

12. Appendix

Matlab Codes, Simulations and Simscape Model

12.1 Matlab Codes
List of Programs:

Program 1. Plot Robot Frames in MATLAB
Program 2. To compute DH Matrix
Program 3. Calculate all homogeneous and position of the gripper Program 4. Plot robot using DH parameters
Program 5. Final Forward kinematics file calling all required functions for plotting stick figures and Inverse kinematics for positions of gripper/end effector.
Program 6. Calculate error between the robot from robotics tool box and robot plotted using DH parameters.
Program 7. Plot Line trajectory as per given points (Start Point and End Point)
Program 8. Solve Inverse kinematics of the robot using Levenberg Marquardt and BFGS Gradient Projection Algorithm.
Program 9. To find joint values of the robot when we give trajectory as input to the file.
Program 10. To plot the points with graphic delay for line
Program 11. To plot the points calculated from trajectory
Program 12. To track the line as per the start and end points give as input.
Program 13. To track the trajectory as defined:
Program 14. Draw GUI figure as show in figure below
Program 15. To Develop GUI for Plot/Tracking functions
Program 16. Trajectory snippet used for plotting random trajectory

Program 1. Plot Robot Frames in MATLAB

```
function [robot] = RVM1Frames(theta_1, theta_2, theta_3, theta_4, theta_5)
% Function plot_manip will plot the manipulator based of DH parameters
% using functions from Robotics System toolbox.
% The function essentially helps in the visualization of the frames

robot = robotics.RigidBodyTree;

%% DH Params
dhparams = [ 0      pi/2    1.52        theta_1;
             2.50      0          0     theta_2;
             1.60      0          0     theta_3;
             0      pi/2          0     theta_4+(pi/2);
```

```
            0      0          .72        theta_5-(pi/2)];
%% Create the Rigid body chain
%used matlab official site for reference

body1 = robotics.RigidBody('body1');
jnt1 = robotics.Joint('jnt1','revolute');
setFixedTransform(jnt1,dhparams(1,:),'dh');
body1.Joint = jnt1;

body2 = robotics.RigidBody('body2');
jnt2 = robotics.Joint('jnt2','revolute');
body3 = robotics.RigidBody('body3');
jnt3 = robotics.Joint('jnt3','revolute');
body4 = robotics.RigidBody('body4');
jnt4 = robotics.Joint('jnt4','revolute');
body5 = robotics.RigidBody('body5');
jnt5 = robotics.Joint('jnt5','revolute');
bodyEndEffector = robotics.RigidBody('endeffector');


setFixedTransform(jnt2,dhparams(2,:),'dh');
setFixedTransform(jnt3,dhparams(3,:),'dh');
setFixedTransform(jnt4,dhparams(4,:),'dh');
setFixedTransform(jnt5,dhparams(5,:),'dh');


body2.Joint = jnt2;
body3.Joint = jnt3;
body4.Joint = jnt4;
bodyEndEffector.Joint = jnt5;

addBody(robot,body1,'base')
addBody(robot,body2,'body1')
addBody(robot,body3,'body2')
addBody(robot,body4,'body3')
addBody(robot,bodyEndEffector,'body4')

% Display
figure
h1 = axes;
% Create the configuration for the Robot
config = homeConfiguration(robot);

% Assign the theta parameters to the config struct
config(1).JointPosition = dhparams(1,4);
config(2).JointPosition = dhparams(2,4);
config(3).JointPosition = dhparams(3,4);
config(4).JointPosition = dhparams(4,4);
config(5).JointPosition = dhparams(5,4);

show(robot, config);
hold on
% Flip the axes to make our frame assignment consistent with the MATLAB
% convention
```

```
set(h1, 'Ydir', 'reverse')
set(h1, 'Xdir', 'reverse')
hold on
```

Program 2. To compute DH Matrix

```
function A = compute_dh_matrix(a, alpha, d, theta)

A = [cos(theta) -sin(theta)*cos(alpha) sin(theta)*sin(alpha) a*cos(theta);...
    sin(theta) cos(theta)*cos(alpha) -cos(theta)*sin(alpha) a*sin(theta);...
    0 sin(alpha) cos(alpha) d;...
    0 0 0 1];

end
```

Program 3. Calculate all homogeneous and position of the gripper

```
function [pos] = RVM1_fk( theta_1, theta_2, theta_3, theta_4, theta_5)
% The input to the function will be the joint
%angles of the robot in radians, and the distance between the gripper pads in
inches.
%The output contains 11 positions of various points along the robot arm

%% DH Parameters assuming home position as zero i.e, theta=0

dhparams = [ 0      pi/2    1.52        theta_1;
             2.50      0          0     theta_2;
             1.60     0         0        theta_3;
             0      pi/2         0      theta_4+pi/2;
             0      0          .72       theta_5-pi/2];

% Extract the ai, alphai, di and thetai from the dhparams matrix

l = .70;
w = .60;
%
grip_1 = [-l/2 0 0];
grip_2 = [-l/2 0 -w];
grip_3 = [0 0 -w] ;
grip_4 = [l/2 0 -w];
grip_5 = [l/2 0 0];


link_1 = dhparams(1,:);
link_2 = dhparams(2,:);
link_3 = dhparams(3,:);
link_4 = dhparams(4,:);
link_5 = dhparams(5,:);
```

```
a_1 = link_1(1); alpha_1 = link_1(2); d_1 = link_1(3); theta1 = link_1(4);
a_2 = link_2(1); alpha_2 = link_2(2); d_2 = link_2(3); theta2 = link_2(4);
a_3 = link_3(1); alpha_3 = link_3(2); d_3 = link_3(3); theta3 = link_3(4);
a_4 = link_4(1); alpha_4 = link_4(2); d_4 = link_4(3); theta4 = link_4(4);
a_5 = link_5(1); alpha_5 = link_5(2); d_5 = link_5(3); theta5 = link_5(4);

% Compute the Ai realtive frames from one joint to other
A01 = compute_dh_matrix(a_1, alpha_1, d_1, theta1);
A12 = compute_dh_matrix(a_2, alpha_2, d_2, theta2);
A23 = compute_dh_matrix(a_3, alpha_3, d_3, theta3);
A34 = compute_dh_matrix(a_4, alpha_4, d_4, theta4);
A45 = compute_dh_matrix(a_5, alpha_5, d_5, theta5);


% Compute position of each frame w.r.t the ground frame
pos = zeros(6, 3);
A02 = A01 * A12;
A03 = A02 * A23;
A04 = A03 * A34;
A05 = A04 * A45;

%calculating positions of all points
pos(1,:) = [0;0;0];
pos(2,:) = A01(1:3,4);
pos(3,:) = A02(1:3,4);
pos(4,:) = A03(1:3,4);
pos(5,:) = A04(1:3,4);
pos(6,:) = A05(1:3,4);

grip_1_pos = (A05 * [grip_1'; 1])';
grip_2_pos = (A05 * [grip_2'; 1])';
grip_3_pos = (A05 * [grip_3'; 1])';
grip_4_pos = (A05 * [grip_4'; 1])';
grip_5_pos = (A05 * [grip_5'; 1])';

%gripper positions
pos(7,:) = grip_1_pos(1:3);
pos(8,:) = grip_2_pos(1:3);
pos(9,:) = grip_3_pos(1:3);
pos(10,:) = grip_4_pos(1:3);
pos(11,:) = grip_5_pos(1:3);

end
```

Program 4. Plot robot using DH parameters

```
function plot_robot_using_DH(pos,theta_1, theta_2, theta_3, theta_4, theta_5)
% Function plot_fk takes the Forward Kinematics solution(pos - 11 x 3 matrix)
computed and plots
% these positions it like a stick figure for the manipulator using plot3
% function
```

```
plot3([pos(1, 1) pos(2, 1) pos(3, 1) pos(4, 1) pos(5, 1) pos(6, 1)]',...
      [pos(1, 2) pos(2, 2) pos(3, 2) pos(4, 2) pos(5, 2) pos(6, 2)]',...
      [pos(1, 3) pos(2, 3) pos(3, 3) pos(4, 3) pos(5, 3) pos(6, 3)]', 'k.-
','linewidth',1,'markersize',20);

hold on
plot3([pos(7, 1), pos(8, 1), pos(9, 1), pos(10, 1), pos(11, 1)]',...
      [pos(7, 2), pos(8, 2), pos(9, 2), pos(10, 2), pos(11, 2)]',...
      [pos(7, 3), pos(8, 3), pos(9, 3), pos(10, 3), pos(11, 3)]','g.-
','linewidth',2,'markersize',20)

end
```

Program 5. Final Forward kinematics file calling all required functions for plotting stick figures and Inverse kinematics for positions of gripper/end effector.

```
%RVM1_forwardninversekinematics
theta_1 = 0;
theta_2 = pi/6;
theta_3 = pi/2;
theta_4 = pi/3;
theta_5  = 0;

robot = RVM1Frames(theta_1, theta_2, theta_3, theta_4, theta_5);
%show(robot);
hold on;
pos = RVM1_fk(theta_1, theta_2, theta_3, theta_4, theta_5);
plot_robot_using_DH(pos);

err = compare_DH_toolbox(robot, pos);

hold on
```

Program 6.Calculate error between the robot from robotics tool box and robot plotted using DH parameters.

```
function err = compare_DH_toolbox(robot, pos)
% function compare.m takes the rigid body tree robot and 11x3 position
%
theta_1 = 0;
theta_2 = pi/4;
theta_3 = pi/3;
theta_4 = pi/4;
theta_5  = 0;
```

```
pos = RVM1_fk(theta_1, theta_2, theta_3, theta_4, theta_5);

robot = RVM1Frames(theta_1, theta_2, theta_3, theta_4, theta_5);
configuration = randomConfiguration(robot);


configuration(1).JointPosition = theta_1;
configuration(2).JointPosition = theta_2;
configuration(3).JointPosition = theta_3;
configuration(4).JointPosition = theta_4;
configuration(5).JointPosition = theta_5;

transform = getTransform(robot,configuration,'endeffector');
err=norm(transform(1:3,4)'-pos(6,:))

%err = 0;

end
```

Program 7. Plot Line trajectory as per given points (Start Point and End Point)

```
function [points] = traj_line(startPoint, endPoint, resolution)
% Function traj_line_generator.m generates points on a
% straight line between start and end points at a specific resolution
% Resolution is a fraction value between 0 and 1
%n is number of points

% startPoint = [7, 15, 25];
% endPoint = [27, 0, 0];
% resolution = 0.1;

p1 = startPoint;
p2 = endPoint;
line_dir = (p2-p1);
points = zeros(floor(1 / resolution),3);
index = 1;

for t = 0 : resolution : 1
    point = p1 + t * line_dir;
    points(index,:) = point;
    index = index + 1;

end
```

Program 8. Solve Inverse kinematics of the robot using Levenberg Marquardt and BFGS Gradient Projection Algorithm.

```
function [posArray, time] = RVM1_iksolve(robot, startPoint, endPoint,
referenceEndPose)

p1 = startPoint;
p2 = endPoint;
line_dir = (p2-p1);
H = trvec2tform([0 0 0]) * eul2tform([0 0 pi], 'ZYX');

resolution = 0.1;
index = 1;

initialguess = robot.homeConfiguration;
field1 = 'MaxIterations';  value1 = 1500;
field2 = 'SolutionTolerance'; value2 = 0.01;
s = struct(field1, value1, field2, value2);

ik = robotics.InverseKinematics('RigidBodyTree',robot, 'SolverParameters',
s,'SolverAlgorithm','LevenbergMarquardt');
%ik = robotics.InverseKinematics('RigidBodyTree',robot, 'SolverParameters',
s);
%we can use either of above commands depending on algorithm needed

points = traj_line(startPoint, endPoint, resolution);
%define points from the line equation

posArray = zeros(6,3,size(points,1));
time_each = zeros(size(points,1),1);

m = size(points,1);

for i = 1:m
    point = points(i,:);
    H(1:3,4) = point';
    %weights = ones(1,6);
    weights = [0.1 0.1 0.1 1.2 1.2 1];
    tform = H;

   [configSol, time_each(index)] = solve_ik(ik, tform, weights,
initialguess);

   jointValues = [configSol(1).JointPosition, configSol(2).JointPosition,
configSol(3).JointPosition,...
                           configSol(4).JointPosition,
configSol(5).JointPosition];
    pos = RVM1_fk( jointValues(1), jointValues(2), jointValues(3),
jointValues(4), jointValues(5));

    posArray(:,:,index) = pos(1:6,:);
    index = index + 1;
    initialguess = configSol;
end

time = sum(time_each);
```

```
posArray(6,1,:)= posArray(6,1,:)-0.72;
posArray(6,3,:)= posArray(6,3,:)-0.72;
%
end
function [configSol, time] = solve_ik(ik, tform, weights, initialguess)
    tic
    [configSol,~] = step(ik,'endeffector',tform,weights,initialguess);
    time = toc;

 % sometimes it takes lot of time , for observation purposes we can use
 % time as calculated above. we can find time needed for different
 % algorithm
%     jointValues = [configSol(1).JointPosition, configSol(2).JointPosition,
configSol(3).JointPosition,...
%                                 configSol(4).JointPosition,
configSol(5).JointPosition, configSol(6).JointPosition];
end
```

Program 9. To find joint values of the robot when we give trajectory as input to the file.

```
function [posArray, time] = RVM1_ik_traj_solve(robot, startPoint, endPoint,
referenceEndPose)

x=[13 14 6  2  5 20 42 49 39 25 22 30];
y=[48 31 7 13 19 12 16 39 61 66 40  2];
n=length(x);
t=1:n;
tt=linspace(t(1),t(n),50);
xx=spline(t,x,tt);
yy=spline(t,y,tt);

H = trvec2tform([0 0 0]) * eul2tform([0 0 pi], 'ZYX');
index = 1;

initialguess = robot.homeConfiguration;
field1 = 'MaxIterations';  value1 = 1500;
field2 = 'SolutionTolerance'; value2 = 0.01;
s = struct(field1, value1, field2, value2);

ik = robotics.InverseKinematics('RigidBodyTree',robot, 'SolverParameters',
s,'SolverAlgorithm','LevenbergMarquardt');
%ik = robotics.InverseKinematics('RigidBodyTree',robot, 'SolverParameters',
s);
%we can use either of above commands depending on algorithm needed

%points = traj_line(startPoint, endPoint, resolution);
%define points fromthe line equation
```

```
%plot(xx,yy,'b')
points =[ xx',yy',tt'];

posArray = zeros(6,3,size(points,1));
time_each = zeros(size(points,1),1);

m = size(points,1);

for i = 1:m
    point = points(i,:);
    H(1:3,4) = point';
    %weights = ones(1,6);
    weights = [0.1 0.1 0.1 1.2 1.2 1];
    tform = H;

   [configSol, time_each(index)] = solve_ik(ik, tform, weights,
initialguess);

   jointValues = [configSol(1).JointPosition, configSol(2).JointPosition,
configSol(3).JointPosition,...
                           configSol(4).JointPosition,
configSol(5).JointPosition];
    pos = RVM1_fk( jointValues(1), jointValues(2), jointValues(3),
jointValues(4), jointValues(5));

    posArray(:,:,index) = pos(1:6,:);
    index = index + 1;
    initialguess = configSol;
end
time = sum(time_each);

posArray(6,1,:)= posArray(6,1,:)-0.72;
posArray(6,3,:)= posArray(6,3,:)-0.72;
%
end
function [configSol, time] = solve_ik(ik, tform, weights, initialguess)
    tic
    [configSol,~] = step(ik,'endeffector',tform,weights,initialguess);
    time = toc;

%     jointValues = [configSol(1).JointPosition, configSol(2).JointPosition,
configSol(3).JointPosition,...
%                                 configSol(4).JointPosition,
configSol(5).JointPosition, configSol(6).JointPosition];
end
```

Program 10. To plot the points with graphic delay for line

```
function plot_inv(posArray, startPoint, endPoint)
% function plot_inverse_output displays the performance of inverse
% kinematics solution through animation
% startPoint - first point on the line
% endPoint - last point on the line
% posArray - 7x3xn matrix containing position of all the seven points on
% the manipulator across all the n steps

[~, ~, m] = size(posArray);
pause on;  % Set this to on if you want to watch the animation
GraphingTimeDelay = .51; % The length of time that Matlab should pause
between positions when graphing, if at all, in seconds.
% Setup plot
figure(2)
scale_f = 1;
axis vis3d
axis(scale_f*[-4 4 -4 4 -4 4])
grid on
view(70,10)
xlabel('X (m.)')
ylabel('Y (m.)')
zlabel('Z (m.)')
% Plot robot initially
hold on
hrobot = plot3([posArray(1, 1,1) posArray(2, 1,1) posArray(3, 1,1)
posArray(4, 1,1) posArray(5, 1,1) posArray(6, 1,1)]',...
      [posArray(1, 2, 1) posArray(2, 2,1) posArray(3, 2,1) posArray(4, 2,1)
posArray(5, 2,1) posArray(6, 2,1)]',...
      [posArray(1, 3, 1) posArray(2, 3, 1) posArray(3, 3, 1) posArray(4, 3,
1) posArray(5, 3, 1) posArray(6, 3, 1)]', 'k.-
','linewidth',1,'markersize',20);
  hold on
plot3([startPoint(1), endPoint(1)]', [startPoint(2), endPoint(2)]',
[startPoint(3), endPoint(3)]', 'g', 'linewidth', 1, 'LineStyle', '--')
pause(GraphingTimeDelay);

actualPose = posArray(6,:,1);

for i = 2 : m
    cla(hrobot)
    set(hrobot,'xdata',[posArray(1, 1,i) posArray(2, 1, i) posArray(3, 1, i)
posArray(4, 1, i) posArray(5, 1, i) posArray(6, 1, i)]',...
        'ydata',[posArray(1, 2, i) posArray(2, 2, i) posArray(3, 2, i)
posArray(4, 2, i) posArray(5, 2, i) posArray(6, 2, i) ]',...
        'zdata',[posArray(1, 3, i) posArray(2, 3, i) posArray(3, 3)
posArray(4, 3, i) posArray(5, 3, i) posArray(6, 3, i)]');
    actualPose = [actualPose;posArray(6,:,i)];
    hold on
    plot3([startPoint(1), endPoint(1)]', [startPoint(2), endPoint(2)]',
[startPoint(3), endPoint(3)]', 'g', 'linewidth', 1, 'LineStyle', '--')
    plot3(actualPose(:,1), actualPose(:,2), actualPose(:,3))
    pause(GraphingTimeDelay);
```

```
end
```

Program 11. To plot the points calculated from trajectory

```
function plot_inv_traj_output(posArray, startPoint, endPoint)
% function plot_inverse_output displays the performance of inverse
% kinematics solution through animation
% startPoint - first point on the line
% endPoint - last point on the line
% posArray - 7x3xn matrix containing position of all the seven points on
% the manipulator across all the n steps

[~, ~, m] = size(posArray);
pause on;  % Set this to on if you want to watch the animation
GraphingTimeDelay = .7; % The length of time that Matlab should pause between
positions when graphing, if at all, in seconds.

% Setup plot
figure(2)
scale_f = 1;
axis vis3d
axis(scale_f*[-4 4 -4 4 -4 4])
grid on
view(70,10)
xlabel('X (m.)')
ylabel('Y (m.)')
zlabel('Z (m.)')

% Plot robot initially
hold on
hrobot = plot3([posArray(1, 1,1) posArray(2, 1,1) posArray(3, 1,1)
posArray(4, 1,1) posArray(5, 1,1) posArray(6, 1,1)]',...
      [posArray(1, 2, 1) posArray(2, 2,1) posArray(3, 2,1) posArray(4, 2,1)
posArray(5, 2,1) posArray(6, 2,1)]',...
      [posArray(1, 3, 1) posArray(2, 3, 1) posArray(3, 3, 1) posArray(4, 3,
1) posArray(5, 3, 1) posArray(6, 3, 1)]', 'k.-
','linewidth',1,'markersize',20);
  hold on
%plot3([startPoint(1), endPoint(1)]', [startPoint(2), endPoint(2)]',
[startPoint(3), endPoint(3)]', 'g', 'linewidth', 1, 'LineStyle', '--')

pause(GraphingTimeDelay);

actualPose = posArray(6,:,1);

for i = 2 : m
    cla(hrobot)
    set(hrobot,'xdata',[posArray(1, 1,i) posArray(2, 1, i) posArray(3, 1, i)
posArray(4, 1, i) posArray(5, 1, i) posArray(6, 1, i)]',...
        'ydata',[posArray(1, 2, i) posArray(2, 2, i) posArray(3, 2, i)
posArray(4, 2, i) posArray(5, 2, i) posArray(6, 2, i) ]',...
        'zdata',[posArray(1, 3, i) posArray(2, 3, i) posArray(3, 3)
posArray(4, 3, i) posArray(5, 3, i) posArray(6, 3, i)]');
    actualPose = [actualPose;posArray(6,:,i)];
    hold on
```

```
    plot3([startPoint(1), endPoint(1)]', [startPoint(2), endPoint(2)]',
[startPoint(3), endPoint(3)]', 'g', 'linewidth', 1, 'LineStyle', '--')
    plot3(actualPose(:,1), actualPose(:,2), actualPose(:,3))
    pause(GraphingTimeDelay);
end
```

Program 12. To track the line as per the start and end points give as input.

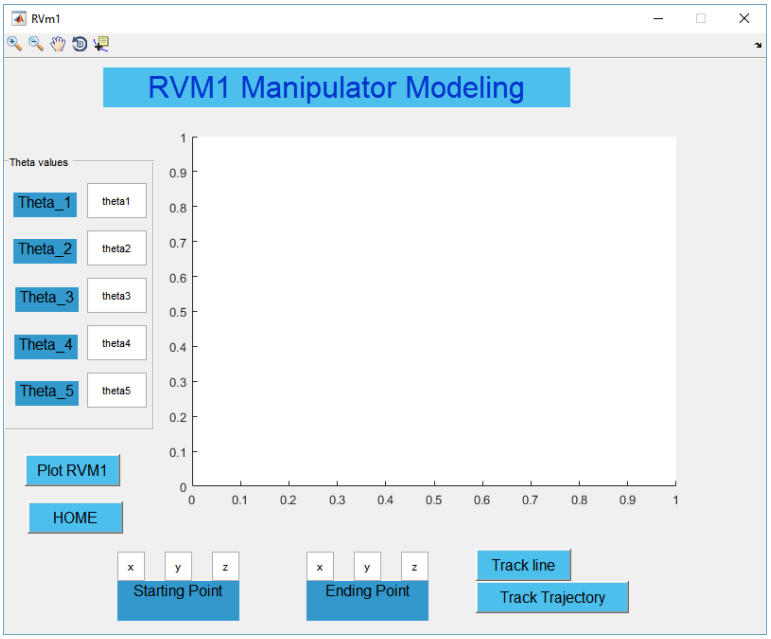
```
%This program will run all the codes and follow the line as per
%starting and end point given to it

theta_1 = 0;
theta_2 = pi/4;
theta_3 = -pi/3;
theta_4 = 0;
theta_5  = pi/4;
theta_6 = 0;

startPoint = [1.1,0,1.3];
endPoint = [2.3,0,3.33];

robot = RVM1Frames(theta_1, theta_2, theta_3, theta_4, theta_5);

robotConfig = robot.homeConfiguration();
robotConfig(1).JointPosition = theta_1;
robotConfig(2).JointPosition = theta_2;
robotConfig(3).JointPosition = theta_3;
robotConfig(4).JointPosition = theta_4;
robotConfig(5).JointPosition = theta_5;
endEffectorPose = getTransform(robot, robotConfig, 'endeffector');

[posArray, time] = RVM1_iksolve(robot, startPoint, endPoint,
endEffectorPose);

plot_inv(posArray, startPoint, endPoint);
hold on
traj_plot()
```

Program 13. To track the trajectory as defined:

```
%This program will run all the codes and follow the random designed
trajectory

theta_1 = 0;
theta_2 = pi/4;
theta_3 = -pi/3;
theta_4 = 0;
theta_5  = pi/4;
theta_6 = 0;
```

```
startPoint = [1.1,0,1.3];
endPoint = [2.3,0,3.33];

robot = RVM1Frames(theta_1, theta_2, theta_3, theta_4, theta_5);

robotConfig = robot.homeConfiguration();
robotConfig(1).JointPosition = theta_1;
robotConfig(2).JointPosition = theta_2;
robotConfig(3).JointPosition = theta_3;
robotConfig(4).JointPosition = theta_4;
robotConfig(5).JointPosition = theta_5;
endEffectorPose = getTransform(robot, robotConfig, 'endeffector');

[posArray, time] = RVM1_ik_traj_solve(robot, startPoint, endPoint,
endEffectorPose);

plot_inv_traj_output(posArray, startPoint, endPoint)
```

Program 14. Draw Gui figure as show in figure below



Program 15. To Develop GUI for Plot/Tracking functions

```
function varargout = RVm1(varargin)
```

```
% RVM1 MATLAB code for RVm1.fig
%      RVM1, by itself, creates a new RVM1 or raises the existing
%      singleton*.
%      H = RVM1 returns the handle to a new RVM1 or the handle to
%      the existing singleton*.
%
%      RVM1('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in RVM1.M with the given input arguments.
%
%      RVM1('Property','Value',...) creates a new RVM1 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before RVm1_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to RVm1_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help RVm1

% Last Modified by GUIDE v2.5 17-Dec-2017 14:23:53

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @RVm1_OpeningFcn, ...
                   'gui_OutputFcn',  @RVm1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before RVm1 is made visible.
function RVm1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to RVm1 (see VARARGIN)

% Choose default command line output for RVm1
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes RVm1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = RVm1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
theta_1 =str2double(get(handles.edit7,'string'))*pi/180;
theta_2 =str2double(get(handles.edit8,'string'))*pi/180;
theta_3 =str2double(get(handles.edit9,'string'))*pi/180;
theta_5 =str2double(get(handles.edit10,'string'))*pi/180;
theta_4 =str2double(get(handles.edit11,'string'))*pi/180;
cla;
pos = RVM1_fk(theta_1, theta_2, theta_3, theta_4, theta_5);
plot_robot_using_DH(pos);

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a
double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a
double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a
double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a
double


% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit6_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a
double


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

startPoint(1) =str2double(get(handles.edit1,'string'));
startPoint(2) =str2double(get(handles.edit2,'string'));
startPoint(3) =str2double(get(handles.edit3,'string'));
endPoint(1)= str2double(get(handles.edit4,'string'));
endPoint(2)= str2double(get(handles.edit5,'string'));
endPoint(3)= str2double(get(handles.edit6,'string'));
theta_1 =str2double(get(handles.edit7,'string'))*pi/180;
theta_2 =str2double(get(handles.edit8,'string'))*pi/180;
theta_3 =str2double(get(handles.edit9,'string'))*pi/180;
theta_5 =str2double(get(handles.edit10,'string'))*pi/180;
theta_4 =str2double(get(handles.edit11,'string'))*pi/180;

robot = RVM1Frames(theta_1, theta_2, theta_3, theta_4, theta_5);

robotConfig = robot.homeConfiguration();
robotConfig(1).JointPosition = theta_1;
robotConfig(2).JointPosition = theta_2;
robotConfig(3).JointPosition = theta_3;
robotConfig(4).JointPosition = theta_4;
robotConfig(5).JointPosition = theta_5;
endEffectorPose = getTransform(robot, robotConfig, 'endeffector');

[posArray, time] = RVM1_iksolve(robot, startPoint, endPoint,
endEffectorPose);
axes(handles.axes1);
```

```
plot_inv(posArray, startPoint, endPoint);
hold on

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
theta_1 =str2double(get(handles.edit7,'string'))*pi/180;
theta_2 =str2double(get(handles.edit8,'string'))*pi/180;
theta_3 =str2double(get(handles.edit9,'string'))*pi/180;
theta_5 =str2double(get(handles.edit10,'string'))*pi/180;
theta_4 =str2double(get(handles.edit11,'string'))*pi/180;
startPoint = [1.1,0,1.3];
endPoint = [2.3,0,3.33];
cla;
robot = RVM1Frames(theta_1, theta_2, theta_3, theta_4, theta_5);

robotConfig = robot.homeConfiguration();
robotConfig(1).JointPosition = theta_1;
robotConfig(2).JointPosition = theta_2;
robotConfig(3).JointPosition = theta_3;
robotConfig(4).JointPosition = theta_4;
robotConfig(5).JointPosition = theta_5;
endEffectorPose = getTransform(robot, robotConfig, 'endeffector');

[posArray, time] = RVM1_ik_traj_solve(robot, startPoint, endPoint,
endEffectorPose);

plot_inv_traj_output(posArray, startPoint, endPoint);




function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a
double


% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit8_Callback(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit8 as text
%        str2double(get(hObject,'String')) returns contents of edit8 as a
double


% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit9_Callback(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit9 as text
%        str2double(get(hObject,'String')) returns contents of edit9 as a
double


% --- Executes during object creation, after setting all properties.
function edit9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit10_Callback(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit10 as text
%        str2double(get(hObject,'String')) returns contents of edit10 as a
double

% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit11_Callback(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text
%        str2double(get(hObject,'String')) returns contents of edit11 as a
double

% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in home.
function home_Callback(hObject, eventdata, handles)
% hObject    handle to home (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
theta_1 =0;
theta_2 =0;
theta_3 =0;
theta_5 =0;
theta_4 =0;
```

```
cla;
pos = RVM1_fk(theta_1, theta_2, theta_3, theta_4, theta_5);
 plot_robot_using_DH(pos);
```

Program 16. Trajectory snippet used for plotting random trajectory

```
function traj_plot()
x=[13 14 6  2  5 20 42 49 39 25 22 30];
y=[48 31 7 13 19 12 16 39 61 66 40  2];
n=length(x);
t=1:n;
tt=linspace(t(1),t(n),50);
xx=spline(t,x,tt)/40;
yy=spline(t,y,tt)/40;

plot(xx,yy,'b')
%axis('equal'),axis('off')
end
```

2.2 Matlab Simscape Model

1. Simscape Block Diagram is as follows:

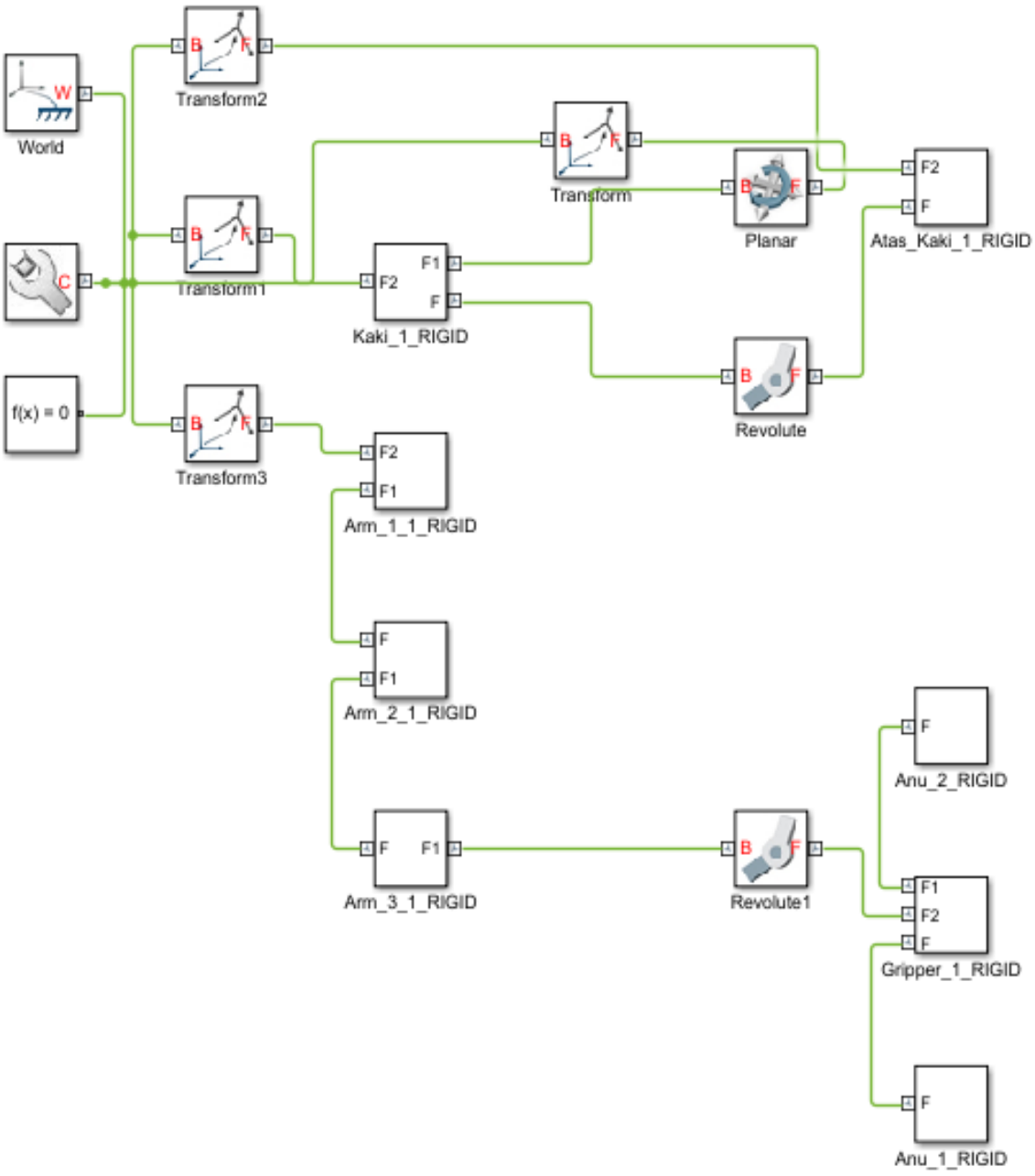


Figure 14. Simscape Block Diagram

2. Simscape Model:

Figure 15. Simscape Simulation Output

3. Solidworks Cad model

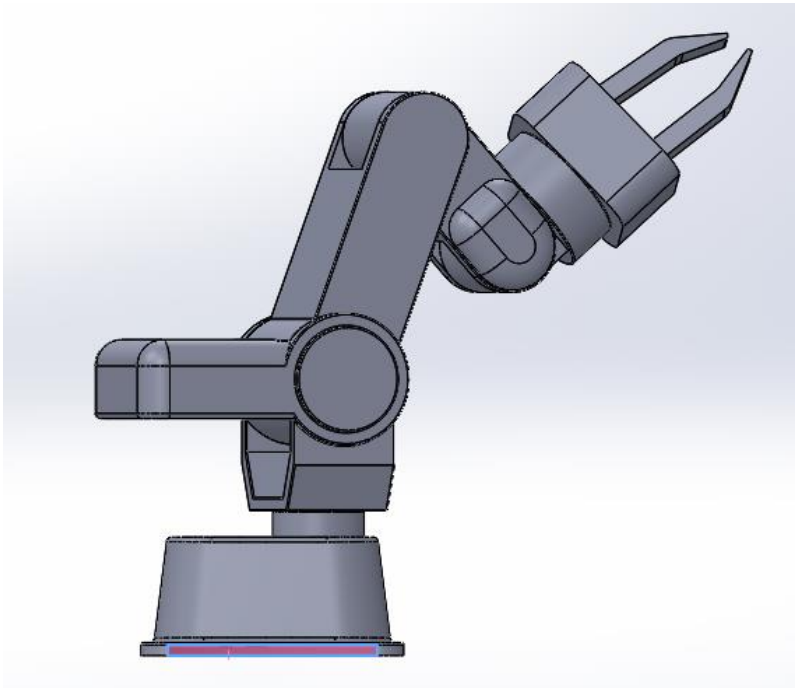


Figure 16. Solidworks Assembly Output.

13 References

[1]. Model and Simulation of the Mitsubishi RV-M1 Robot using MATLAB
Haibin Zhao, Zhiguo Lu, Chong Liu, Hong Wang School of Mechanical Engineering and Automation Northeastern University in 2016 IEEE International Conference on Signal and Image Processing.

[2]. R.N. Jazar, Theory of Applied Robotics, Second Edition: Springer.
New York Dordrecht Heidelberg, 2010.

[3] Badreddine, Hassan, Stefan Vandewalle, and Johan Meyers. "Sequential Quadratic Programming (SQP) for Optimal Control in Direct Numerical Simulation of Turbulent Flow." *Journal of Computational Physics*. 256 (2014): 1–16. doi:10.1016/j.jcp.2013.08.044.

[4] Bertsekas, Dimitri P. *Nonlinear Programming*. Belmont, MA: Athena Scientific, 1999.

[5] Goldfarb, Donald. "Extension of Davidon's Variable Metric Method to Maximization Under Linear Inequality and Equality Constraints." *SIAM Journal on Applied Mathematics*. Vol. 17, No. 4 (1969): 739–64. doi:10.1137/0117067.

[6] Nocedal, Jorge, and Stephen Wright. *Numerical Optimization*. New York, NY: Springer, 2006.

[7] Sugihara, Tomomichi. "Solvability-Unconcerned Inverse Kinematics by the Levenberg–Marquardt Method." *IEEE Transactions on Robotics* Vol. 27, No. 5 (2011): 984–91. doi:10.1109/tro.2011.2148230.

[8] Zhao, Jianmin, and Norman I. Badler. "Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures." *ACM Transactions on Graphics* Vol. 13, No. 4 (1994): 313–36. doi:10.1145/195826.195827.

[9] https://www.mathworks.com/discovery/matlab-gui.html

[10] https://www.mathworks.com/help/robotics/ug/inverse-kinematics-algorithms.html

[11] A virtualRV-M1robotsystem Rajesh Kumar a,n, ParveenKalra b, NeelamR.Prakash c a Mechanical EngineeringDepartment,RoomNo.403,AcademicBlock2,U.I.E.T.,PanjabUniversity,Chandigarh160014, India.

[12] Explicit Kinematic Model of the Mitsubishi RV-M1 Robot Arm
Yacine Benbelkacem and Rosmiwati Mohd-Mokhtar, Member, IEEE