Name: Megha Veerendra

NYU ID: mv1807

Fundamentals of Robotics

Project Report

Question 1

The joint velocities can be calculated using two methods: inverse Jacobian and Jacobian transpose.

I first computed the Jacobian using the DH parameters that were provided. The detailed computation is in the **direct_kin.m** file. I directly entered the corresponding formulas in the Simulink blocks.

a) Jacobian inverse method to compute the joint positions.

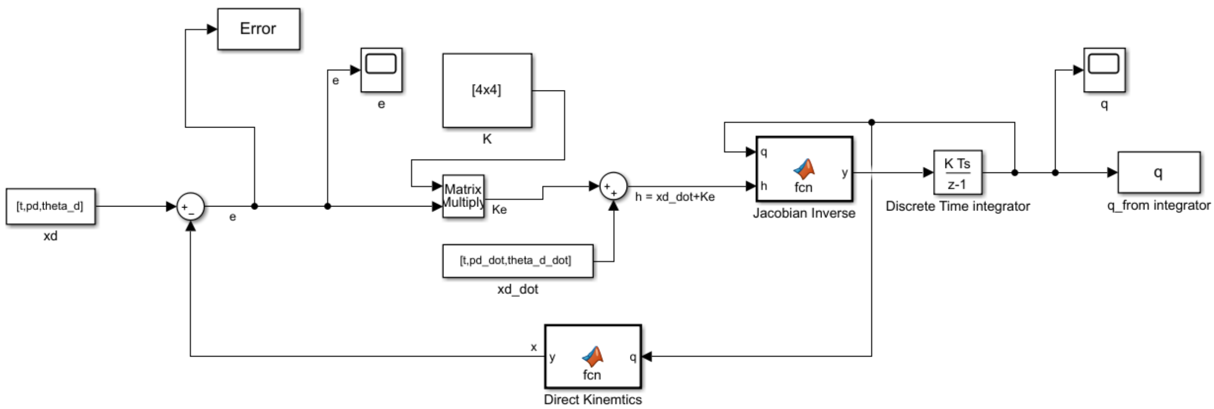

Fig 1: Simulink model for Jacobian Inverse method

The model follows the schematic shown in Fig 2.

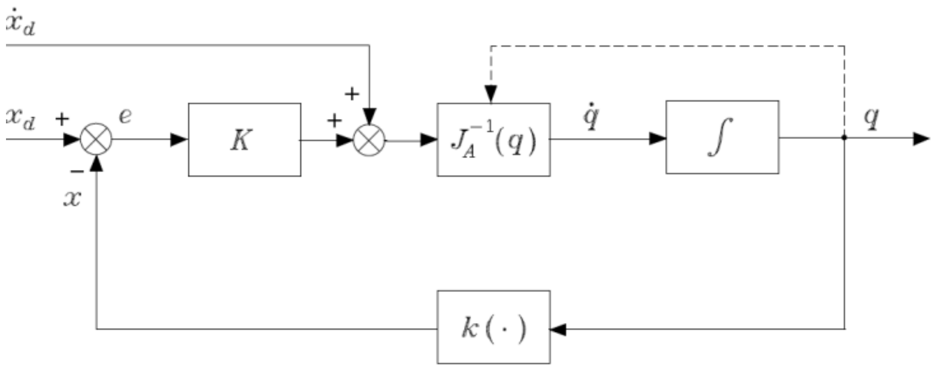

Fig 2: Jacobian Inverse Schematic

- The parameter x_d = [pd,theta_d] which is obtained from the data provided in the matlab file kinematic_traj.m.
- $\dot{x}_d$ = [pd_dot,theta_d_dot] which is again information provided in kinematic_traj.m.
- x is obtained from the direct kinematic block. Initially this value is obtained from the initial conditions which is provided in the integrator block as q0. This data comes from kinematic_traj.m. Initially, q0 passes through the direct kinematics block to give x i.e the position and orientation in the operational space.
- The error between the desired and obtained positions i.e e = xd-x,
- The error 'e' is fed to the gain block which is a 4x4 diagonal matrix with 10 as the diagonal elements. The 'K' matrix represents the gain which amplifies the error.
- Following this the amplified gain is summed with $\dot{x}_d$ and fed to the Jacobian Inverse block.
- The Jacobian Inverse block contains formula of the jacobian inverse. The detailed calculation of the Jacobian and its inverse is in direct_kin.m file. The output of the Jacobian Inverse block is the '$\dot{q}$' vector which is a column vector that contains the joint velocities.
- The '$\dot{q}$' vector is fed to the integration block where Euler integration is performed. After integration, 'q' is obtained which is a column vector containing the joint positions.
- The direct kinematics block takes 'q' as the input and gives as output the operation space vectors which is given by the 'x' vector.
- The process repeats for 4 seconds which is the length of the simulation.

- It is observed that the error becomes zero as the desired and obtained vectors become equal.

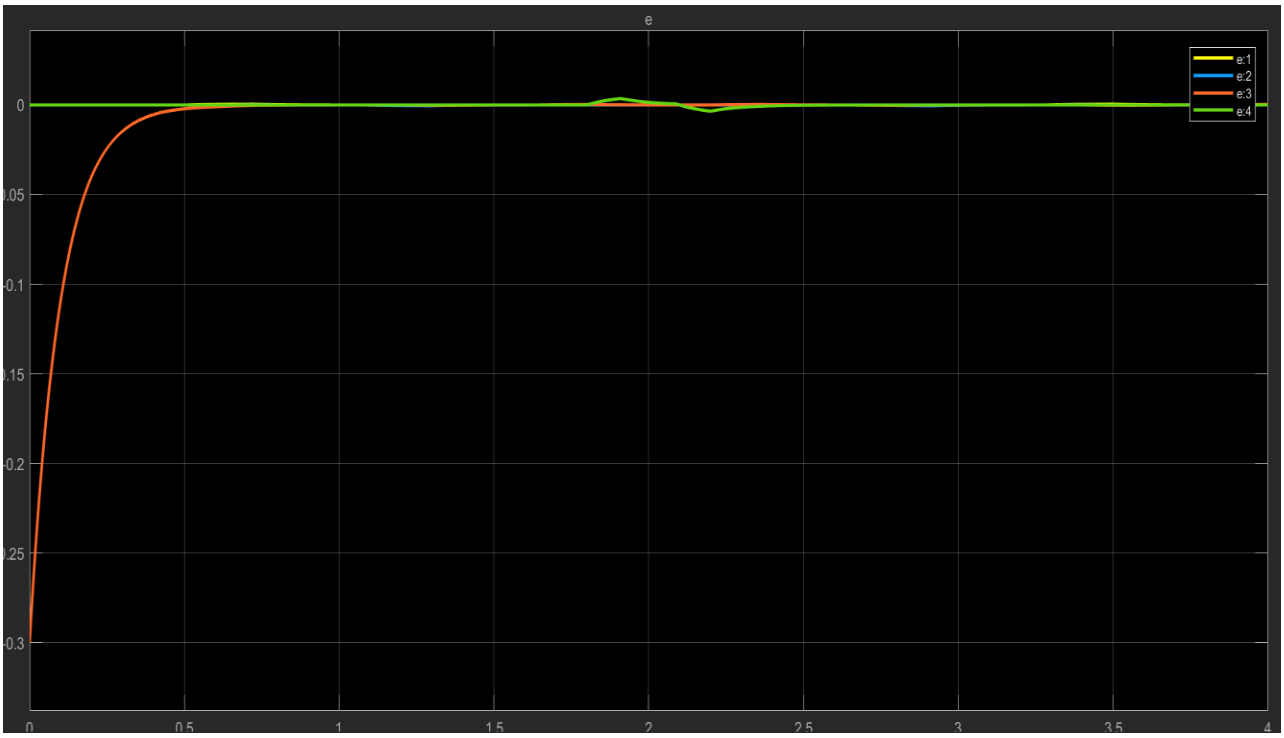

Fig 3: Graph showing Error in operational space parameters (e = xd - x) V/s Time

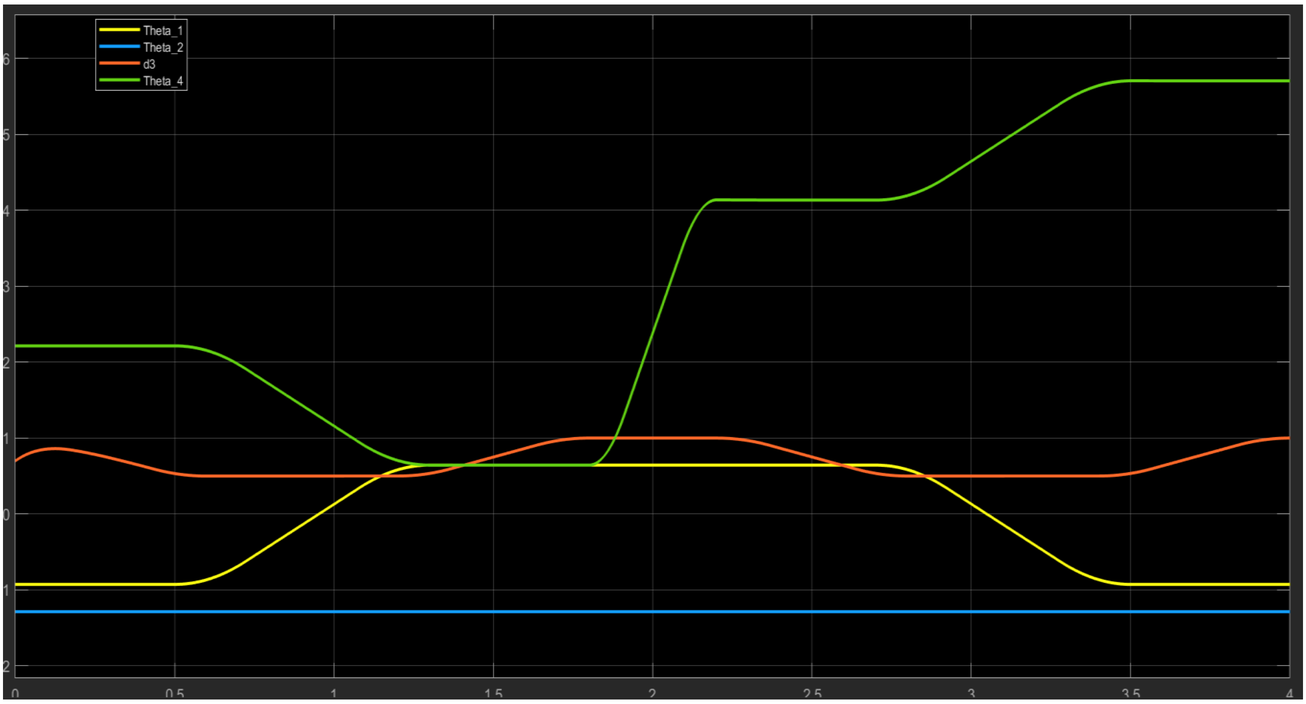

Fig 4: Graph showing the joint space parameters wrt time

b) Jacobian Transpose method to compute joint positions

- The Jacobian transpose method follows the same procedure as the Inverse Jacobian model except that the schematic is implemented as below in figure 4

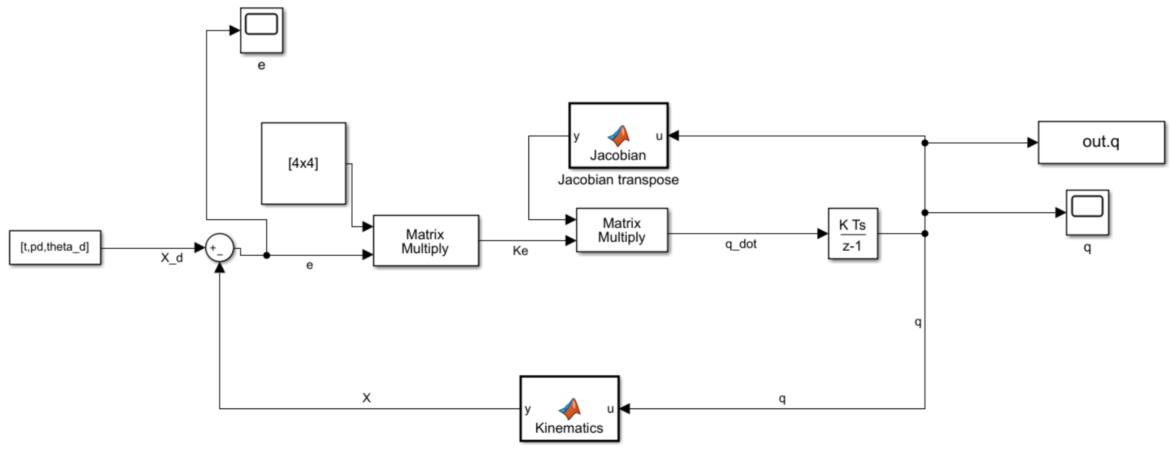

Fig 5: Simulink model for Jacobian Transpose method

The Simulink model for the Jacobian Transpose method follows the schematic shown in Fig 6.

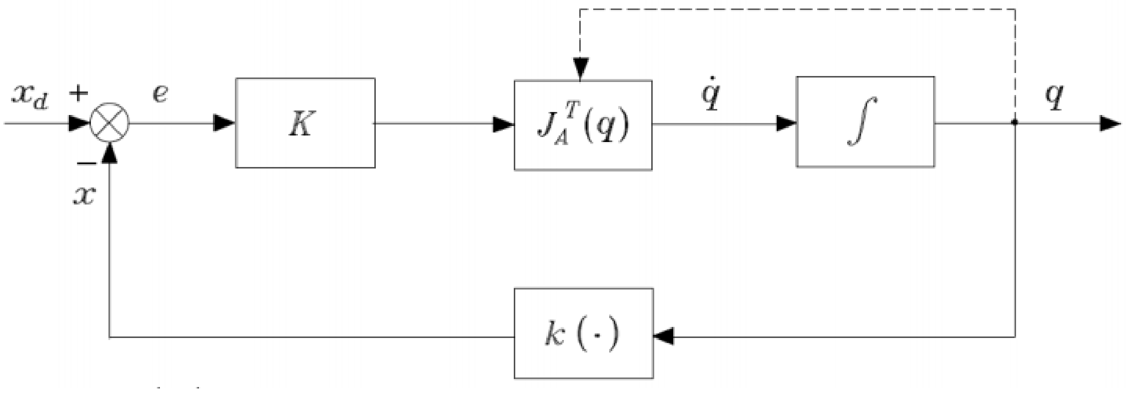

Fig 6: Jacobian Transpose Algorithm Schematic

The output of the simulation is as follows:

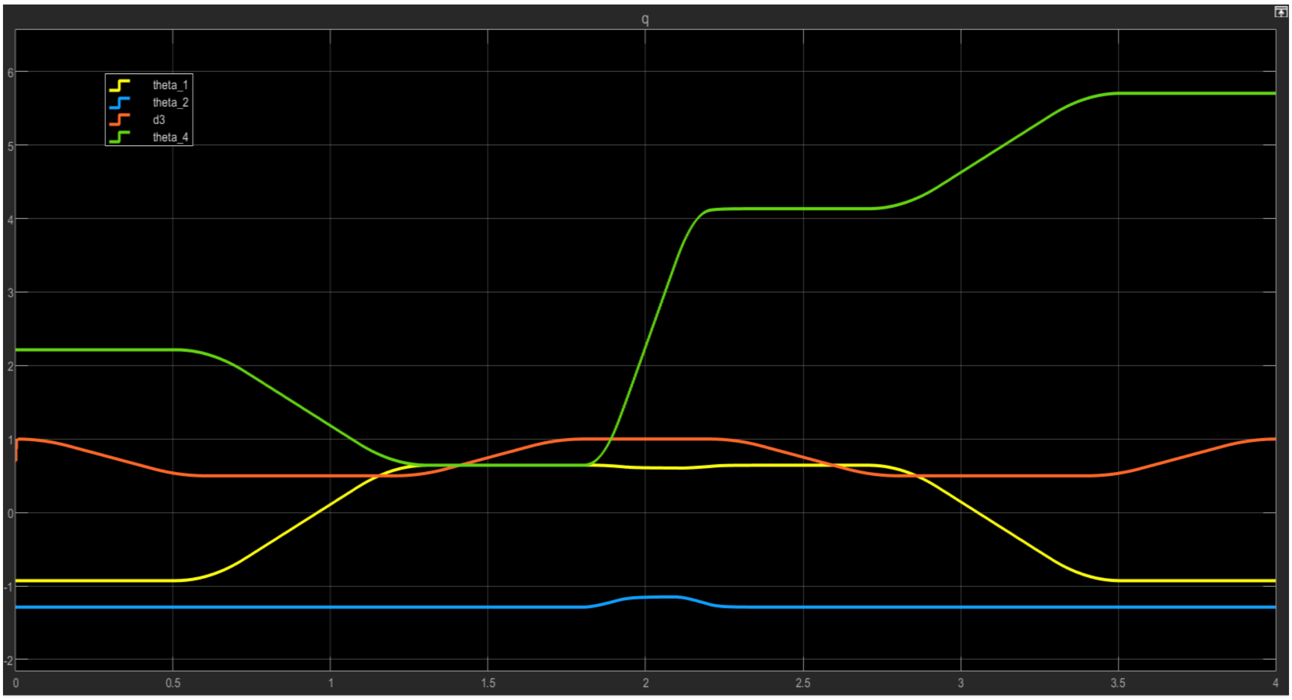

Fig 7: Variation of joint space parameters with time

- It is observed that the error in the desired and observed trajectories converges to zero, this is proof of the manipulator following the trajectory.

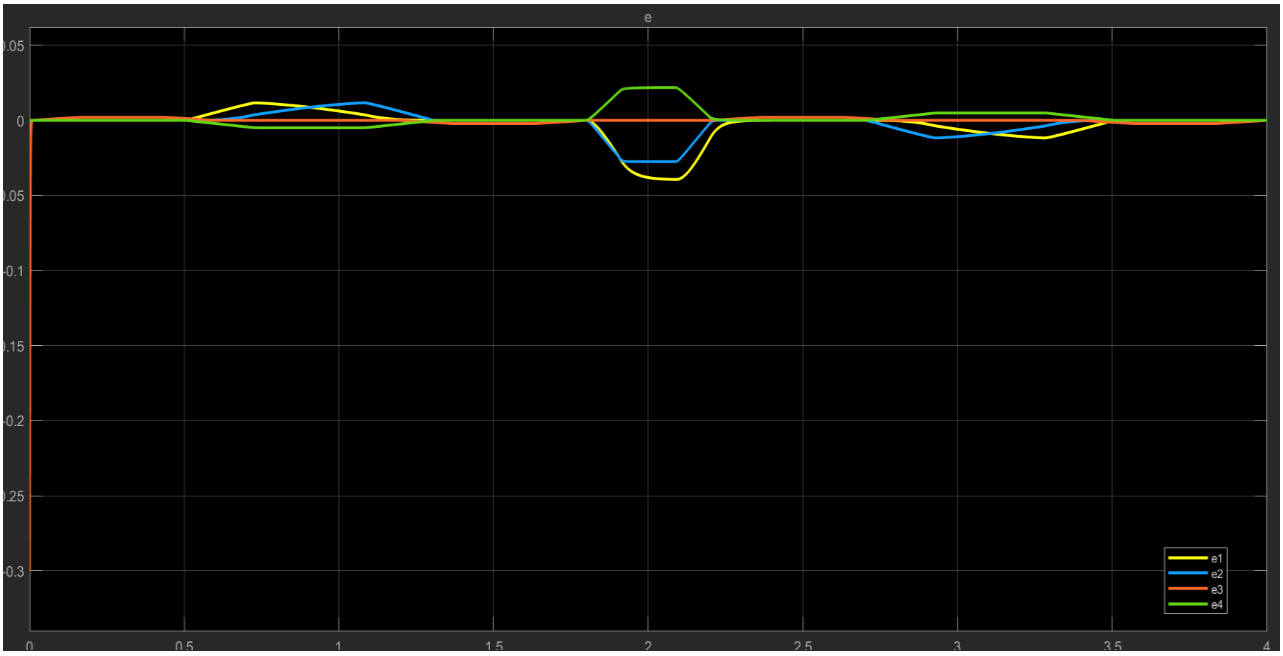

Fig 8: Graph showing Error in operational space parameters (e = xd-x) V/s Time

Question 2

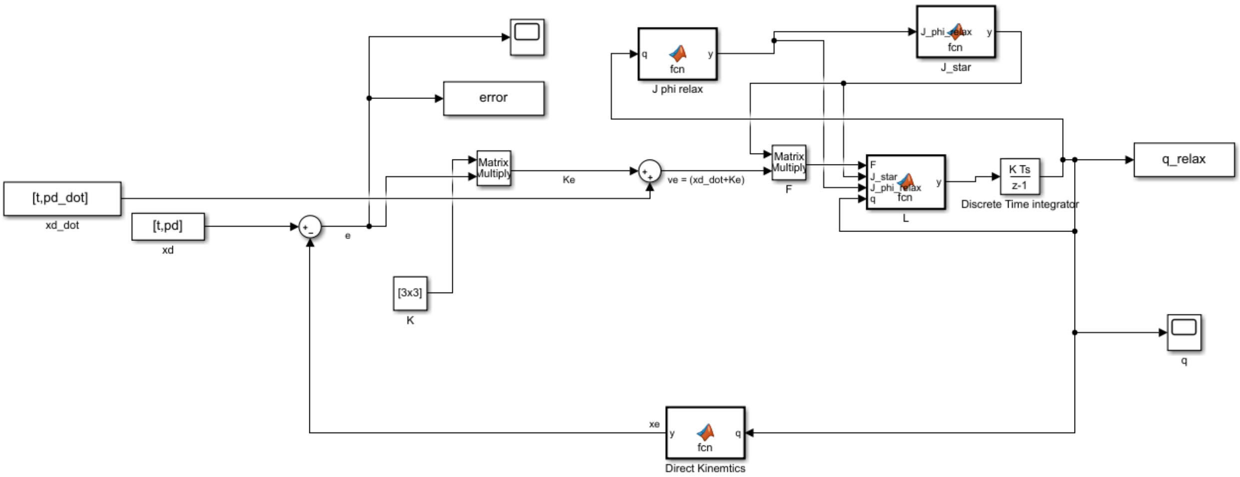

Fig 9: Simulink model for computing joint positions with relaxation of phi using Inverse Jacobian algorithm

- This question requires the relaxation of the phi component of the joint space. This creates a redundancy for the robot.
- To relax phi from the operational space, we simply remove the section of the Jacobian that is related to the movement of phi. The new Jacobian is now a 3 x 4 matrix.
- I then computed the Jacobian pseudo inverse using the new Jacobian this done in matlab function block J_star .
- In order to obtain the maximum distance from end joints. We compute joint velocities with respect to joint limits. This is given by partially differentiating w(q) w.r.t joint space variables (q) i.e $\frac{\partial w(q)}{\partial q}$.
- In block F of the matlab function, I compute the $\dot{q}_0$ vector by multiplying with vector $\frac{\partial w(q)}{\partial q}$ a constant gain k0.
- Finally, I compute $\dot{q}$ by multiplying J_star with J, subtracting it from identity matrix I, then I multiply this with $\dot{q}_0$ and then add to this the F matrix which is a product of J_star and v_e .
- Using the redundant manipulator in the system (i.e., theta 4), we use Jacobian inverse algorithm to compute joint positions for the given trajectory. The same schematic as Fig 2 is followed.

.

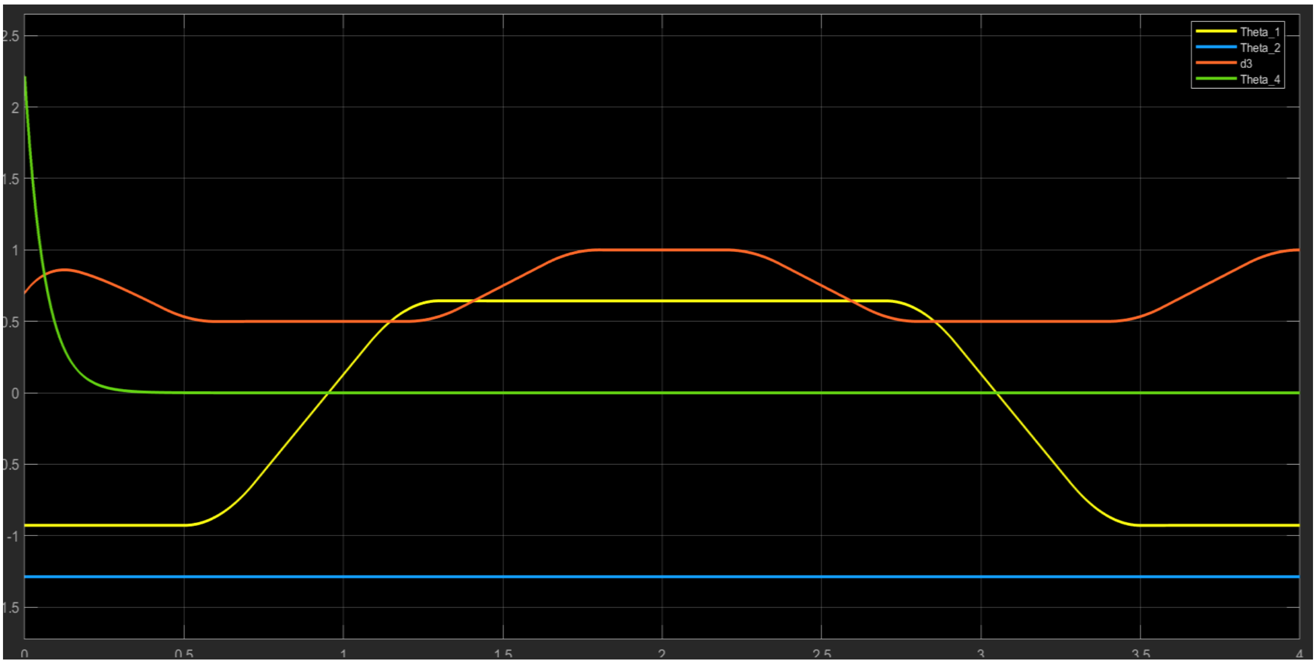

Fig 10: Graph showing the evolution of joint space parameters wrt time

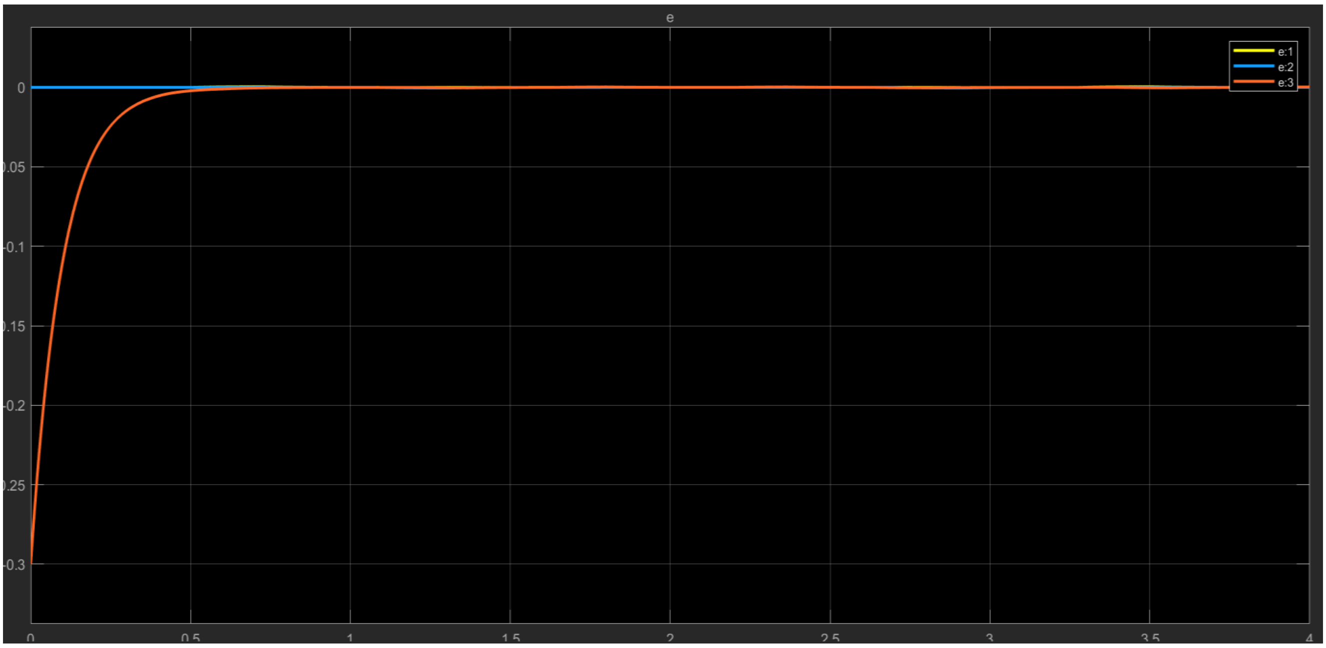

Fig 11: Graph showing Error in operational space parameters (e = xd - x) V/s Time