MAE 547: Modelling and Control of Robots
Project: Robotics Toolbox

By: Vivek Verma, Kamlesh Rathinam, Jayasurya Sevalur Mahendran, Asjad Malik, Manaswini Ayalasomayajula

Steps to Run the File

1) Start MATLAB
2) Download and Run PeterCorke's Robotics Toolbox version 9.1
3) Select the MAINFILE.m
4) Run the MAINFILE.m file and then select Add to path
5) Select the function which you need to perform (Click on homogenous transformation to get the transformation matrix, for further help, regarding how to run the code, refer the documentation below)

1) Homogenous Transformation Documentation

What is Homogenous Transformation?

Homogenous Transformation Matrix defines a location (position and orientation) with respect to a reference frame. Consider the fact that any configuration can be achieved from the initial configuration by first rotating, and then translating.

How to run this code?

User needs to enter the three angles of rotation in degrees and select one among two frames, ZYX or ZYZ.
Also, besides this user also needs to enter the translation along X-axis, Y-axis and Z-axis.
All five functions Transformation Mapping, Transformation Operator, Euler angles, description of the frame, rotation matrix will open in ONE single SubGUI.

Input

Three angles and the frame in which transformation is to be done (RPY or ZYZ frames). These parameters are specified by the user in the GUI window.
Angles specified are in degrees.

Output

The transformation matrix Transformation Mapping, Transformation Operator, Euler angles, description of the frame, rotation matrix is displayed in the GUI window.
A plot and animation of the final transformation is displayed.

Flow of Program

1) The angles entered by the user are taken as the inputs. These inputs are plugged into the homogenous transformation equations.
2) This value is then displayed in the GUI Window.

$$\boldsymbol{T}_e^b(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{n}_e^b(\boldsymbol{q}) & \boldsymbol{s}_e^b(\boldsymbol{q}) & \boldsymbol{a}_e^b(\boldsymbol{q}) & \boldsymbol{p}_e^b(\boldsymbol{q}) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2) Euler Angles Documentation

What are Euler Angles?
The Euler angles are three angles introduced by Leonhard Euler to describe the orientation of a rigid body with respect to the fixed coordinate system. They can also represent the orientation of a mobile frame of reference in physics or the orientation of a general basis in 3-D linear algebra.
Euler angles are typically denoted as ϕ, θ, ψ

How to run this code?
User needs to enter the transformation matrix in the GUI Window and click on Euler angles. User gets two sets of equivalent Euler angles (because one transformation matrix can give two sets of Euler angles).

Input
Transformation Matrix from the GUI window.

Output
Two sets of equivalent Euler angles in degrees.

Flow of Program
1) The transformation matrix entered by the user are taken as inputs, these values are plugged into to equations to get Euler angles.
2) This value is then displayed in the GUI Window.

4) Forward Kinematics Documentation

What is Forward Kinematics?
Forward Kinematics refers to the use of the kinematic equations of a robot to compute the position of the end-effector from the specified values for the joint parameters. Given the joint angle values, forward kinematics equations calculate the robot's end-effector location in the coordinate space.

How to run this code?

Enter the joint angles in radians in the GUI Window and click Get End Effector Positions. You can see the plot where you can teach the robot by using the sliders on the window.

Input
Joint angles from the GUI window.

Output
End effector co-ordinates.

Flow of Code

1) The inputs taken from the user are plugged into the general formula to calculate the transformation matrix for every joint.
2) The links are created using ''Link'' function PeterCorke's robotics toolbox, these are assembled using ''Serial Link''. This creates our Robot.
3) The function R.teach, this opens up the plot which lets the user teach the robot using sliders available in the plot.

$$\boldsymbol{A}_i^{i-1}(q_i) = \boldsymbol{A}_{i'}^{i-1}\boldsymbol{A}_i^{i'} = \begin{bmatrix} c_{\vartheta_i} & -s_{\vartheta_i}c_{\alpha_i} & s_{\vartheta_i}s_{\alpha_i} & a_i c_{\vartheta_i} \\ s_{\vartheta_i} & c_{\vartheta_i}c_{\alpha_i} & -c_{\vartheta_i}s_{\alpha_i} & a_i s_{\vartheta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

5) Inverse Kinematics Documentation

What is Inverse Kinematics?
Inverse kinematics is a method that helps define the motion of a robot to reach a desired location. Given the robot's end-effector location, inverse kinematics equations calculate the joint angles required to move the end-effector to that location.

How to run this code?
Enter the end effector co-ordinates of X-axis, Y- axis and Z-axis in the GUI Window and click Joint angles.

Input
End effector co-ordinates from the GUI window.

Output
Joint angles in radians.

Flow of the Code

1) The inputs from the GUI window in the form of Euler angles are plugged into inverse kinematics function.
2) The final output of the code gives the joint angles in degrees and this is displayed in the GUI window.

6) Workspace Documentation

What is workspace?

The workspace of robot manipulator is defined as the set of points that can be reached by its end-effector. In other words, the workspace of a robot is the space in which the mechanism is working (purely and simple). Despite this definition is still widespread, several authors also refer to workspace as work volume or work envelope.

Input variables

Link lengths an: These values are constant and do not change. They are obtained from the DH parameter table given by the user.

Revolute joints angle limits tn: The limits are directly entered by the user (in radians)

Prismatic joints distance limits dn: The limits are directly entered by the user (in meters)

Output variables

None.

Functions called

“ar_o_no()”: Used to extract the exact number of variables being used by the matlabFunction equations

Output

A 3D graph of the workspace of the given robot based on the given joint limits and equations.

Flow of program / Concept of Workspace and algorithms used

1) The inputs tn and dn vary within the limits given. They are concatenated into an ‘char’ array.

“ndgrid” MATLAB function is used to plot a n-dimensional objects on an N-dimensional grid.

2) The px, py, and pz are equated to their respective equations and these position coordinates will be used to plot the points in the 3D space.

3) Function “ar_o_no()” has input variables of MATLAB Function and returns the variables being used by the equation. This function is necessary to be able to use only the required number of elements or variables from the q array.

For example: if a MATLAB Function equation has only two variables out of the 6 variables in the robot arm for computation of that particular coordinate position, only the two variables must be inputted to the ndgrid function to span the space within those limits.

4) The values of the variables and their limits are plugged into the given equations for the robot in x, y and z coordinate positions to give the span of the equations in their respective coordinate.

5) These matrices are then entered as the “ndgrid” input parameters to then obtain the 3D figure of the workspace.

7) Differential Kinematics Documentation

Forward kinematic equations are functions used to relate joint variables to end-effector positions and orientations with respect to a fixed coordinate system. The Jacobian matrix linearly relates joint velocities to end-effector velocities and can be useful in several aspects of robotics: planning and execution of smooth paths, determination of singular configurations, derivation of dynamic equations of motion, and transformation of forces and torques from the end effector to the manipulator joints. When a manipulator is in singular configuration, certain directions of motion are unattainable, or a degree or degrees of freedom are lost.

How to run this code?

Enter the DH Parameters and type of joint (prismatic or revolute) and Select Jacobian, this will display the Jacobian Matrix in the Window.

For Singularities, click on Get Singularities, this will give dependent joints, equation for singularities and angles on which singularities occur.

Input

DH Parameters and type of joint from the GUI window.

Output

Jacobian Matrix, dependent joints, equation for singularities and angles on which singularities occur, joints which are dependent.

Flow of Code

DH parameters are taken as inputs, new links with the new parameters are formed and a new robot object is created called ‘’R’’.

1) We obtain symbolic jacobian matrix through the code.
2) Using the jacob0 function from the Peter Corke robotic toolbox, we obtain jacobian matrix.
3) Singularities equation is obtained from the symbolic jacobian we have derived.

4) Singularities equation requires determinant of the jacobian matrix, determinant is calculated using our function, ''my deter''.
5) The singularities also gives dependent links, this is obtained from the function from Jsingu.

Equation for jacobian

$$J(q) = \begin{bmatrix} z_0 \times (p_3 - p_0) & z_1 \times (p_3 - p_1) & z_2 \times (p_3 - p_2) \\ z_0 & z_1 & z_2 \end{bmatrix}$$

8) Inverse Differential kinematics and inverse kinematics using Jacobian Documentation

What is inverse differential kinematics?

Resolves end-effector velocity into velocities of individual joints.

How to run this code?

User needs to enter Euler Angles in radians and co-ordinates in X, Y and Z axis in the GUI Window and select Get Joint Velocities, joint velocities will be displayed.

Input

Euler Angles and co-ordinates in X, Y and Z axis from the GUI window.

Output

Joint Velocities

Flow of Code

1) The inputs from the GUI Window is taken and plugged into the code.
2) The end-effector velocities are calculated, which are displayed in the GUI Window.

9) Manipulator Dynamics and Control Documentation

Manipulator dynamics is concerned with the equations of motion, the way in which the manipulator moves in response to torques applied by the actuators, or external forces.

Derivation of the dynamic model of a manipulator plays an important role for simulation of motion, analysis of manipulator structures, and design of control algorithms. Simulating manipulator motion allows control strategies and motion planning techniques to be tested without the need to use a physically available system. The analysis of the dynamic model can be helpful for mechanical design of prototype arms. Computation of the forces and torques required for the execution of typical motions provides useful information for designing joints, transmissions and actuators.

Input variables

DH parameter matrix is the required input from which the code will extract the number of links, d, a and alpha to build links along with mass, inertia, gravity, etc.

The user will have to input the following data from the GUI:

Mass

Inertia

Gear ratio

To obtain the matrix of the pose of the robot over time T, and the joint velocities; joint torques, initial joint position and initial joint angles are supposed to be entered by the user.

Functions called

Jacobians function:

We obtain the necessary Jacobian computation matrices from the function Jacobians.

This function requires two inputs:

DH parameter matrix

Transformation matrices

It will use the number of links from the DH matrix and the required columns and matrix elements from the various transformation matrices to obtain Jp and Jo.

Equations_of_motion function:

Manipulator inertia matrix (G)

i = R.inertia(**q**) is the symmetric joint inertia matrix (NxN) which relates joint torque to joint acceleration for the robot at joint configuration **q**.

If **q** is a matrix (KxN), each row is interpretted as a joint state vector, and the result is a 3d-matrix (NxNxK) where each plane corresponds to the inertia for the corresponding row of **q**.

Friction force (F)

tau = R.friction(qd) is the vector of joint friction forces/torques for the robot moving with joint velocities qd.

The friction model includes:

- Viscous friction which is a linear function of velocity.
- Coulomb friction which is proportional to sign(QD).

Coriolis matrix (C)

C = R.coriolis(q, qd) is the Coriolis/centripetal matrix (NxN) for the robot in configuration q and velocity qd, where N is the number of joints. The product C*qd is the vector of joint force/torque due to velocity coupling. The diagonal elements are due to centripetal effects and the off-diagonal elements are due to Coriolis effects. This matrix is also known as the velocity coupling matrix, since it describes the disturbance forces on any joint due to velocity of all other joints.

If q and qd are matrices (KxN), each row is interpretted as a joint state vector, and the result (NxNxK) is a 3d-matrix where each plane corresponds to a row of q and qd.

C = R.coriolis(qqd) as above but the matrix qqd (1x2N) is [q qd].

Joint viscous friction is also a joint force proportional to velocity but it is eliminated in the computation of this value.

Gravity load on joints (G)

> **taug** = R. gravload(**q**) is the joint gravity loading (1xN) for the robot R in the joint configuration **q** (1xN), where N is the number of robot joints. Gravitational acceleration is a property of the robot object.
>
> If **q** is a matrix (MxN) each row is interpreted as a joint configuration vector, and the result is a matrix (MxN) each row being the corresponding joint torques.
>
> **taug** = R.gravload(**q**, **grav**) as above but the gravitational acceleration vector **grav** is given explicitly.

The equation of motion would thus be:

$$\mathbf{T - J^{*}he = B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F(\dot{q}) + G(q)}$$

Output variables

Using functions from the Peter Corke toolbox we also obtain the symmetric joint space inertia matrix (B), Coriolis and centripetal effects matrix (C), viscous and Coulomb friction matrix (F) and gravity loading matrix (G).

From the Jacobian function we obtain Jp and Jo.

Flow of program/algorithms used

Equations of motion function

1) After the necessary inputs are obtained, new links with the new parameters are formed and a new robot object is created called “myrobot”.
2) Using the “fdyn” function from the Peter Corke robotic toolbox, we obtain q, q_dot and torque T.

3) Using these values, we use other inbuilt function to obtain the coefficient matrices of the equation of motion.

Jacobian function:

1) The number of links of the robot is obtained from the DH parameters that is passed as an input argument to the function.
2) The values of z and p are stored in a 3D array that will contain these values for all links. They are equated to the corresponding columns and rows.
3) Using the previous variables, we compute them as necessary and store them in Jp and Jo depending on the link type.

$$\boldsymbol{J}_{Pj}^{(\ell_i)} = \begin{cases} \boldsymbol{z}_{j-1} & \text{for a } \textit{prismatic} \text{ joint} \\ \boldsymbol{z}_{j-1} \times (\boldsymbol{p}_{\ell_i} - \boldsymbol{p}_{j-1}) & \text{for a } \textit{revolute} \text{ joint} \end{cases}$$

$$\boldsymbol{J}_{Oj}^{(\ell_i)} = \begin{cases} \boldsymbol{0} & \text{for a } \textit{prismatic} \text{ joint} \\ \boldsymbol{z}_{j-1} & \text{for a } \textit{revolute} \text{ joint.} \end{cases}$$

10) Manipulator Control Documentation

What is Manipulator control?

The basic concept of the hybrid position/force controller is to decouple the position and force control problems into subtasks via a task space formulation. It’s the determination of the time history of the generalized forces to be developed by the joint actuators.

How to Run the code?

1) In the main GUI, select the motion control, the command window prompts up where the user is required to enter the desired values and the final time of simulation, upon that Simulink pops up, the user is now required to RUN the simulation, and after that the user needs to return back to the command window and TYPE “1” to simulate. After this the plots will pop up.
2) For Force Control, select the Force control, the command window prompts up where the user is required to enter the desired values and the final time of simulation, upon that Simulink pops up, the user is now required to RUN the simulation, and after that the user needs to return back to the command window and TYPE “1” to simulate. After this the plots will pop up.
3) In Force Control, the user is supposed to enter the Link mass, motor mass, gravity matrix gear ratio and by default the value are stored if the user does not have the value a simulation will occur with the pre stored values.

Inputs

Xd, ẋd, ẍd are taken as inputs from the GUI window.

Output

Plots which represent that the joint will become stable over the time “t” and the force required which is required by the joint to from one point to another.

Contributions of Each Member to the Project

1) Vivek Verma: Homogenous Transformations, Euler Angles, Differential Kinematics, Manipulator Control, GUI Implementation
2) Kamlesh Rathinam: Forward Kinematics, Inverse Kinematics, Inverse Differential Kinematics and Inverse Kinematics using Jacobian, GUI Implementation
3) Manaswini Ayalasomayajula : Workspace, Manipulator Dynamics
4) Asjad Malik: Homogenous Transformations, Euler Angles, Differential Kinematics, Manipulator Control, GUI Design
5) Jayasurya Sevalur Mahendran: Forward Kinematics, Inverse Kinematics, Inverse Differential Kinematics and Inverse Kinematics using Jacobian, Manipulator Dynamics, GUI Implementation