Foundations of Robotics Final Project

Gabriel Bronfman and Cinerita Andrandes

User Manual

The code is relatively simple to operate. All the functions live inside their own respective files, while the ‘main.m’ file is the primary location you can run and interact with the code. Within the main, you will find five primary sections. Below will be the functionality demonstrated within each primary section of the main code. For more information on the inputs and outputs of each function, check the documentation.

1. Load Prerequisites Display information

This section contains a single line, which imports the URDF file for a KUKA IIWA14 7-DOF manipulator into the code as a ‘RigidBodyTree’ object titled ‘robot’. The RigidBodyTree object allows Matlab to interpret a set of links and joints of specified length and type from the URD file for visualization purposes. It also captures a CAD file described by the URDF and attaches it to the specified locations for each joint and link.

2. Forward Kinematics

This section demonstrates the ForwardKinematicsCalculation and ForwardKinematicsDisplay functionality of the code package. Because the URDF file already knows how to rotate the robot according to inputs for each joint, there is a ForwardKinematicsCalculation function that defines the frame T^{B}_{EE} according to the DH table independently determined by the team. By changing the values defined in desJointAngles, you can control the visualized pose of the robot.

ForwardKinematicsDisplay uses the joint angles from desJointAngles to ‘move’ the ‘robot’ object into the desired configuration via a returned configuration object. You may use the last two lines to determine if the actual transform of the end-effector of the visualized robot is the same as the team’s computed transform for the same joint angles.

3. Inverse Kinematics

This section demonstrates the use of the inverseKinematics function included in the code package. By altering the values of desEEPos = [alpha beta gamma x y z] for the end effector of the manipulator, you can compute any required joint positions for the manipulator(written as a homogeneous transform of the end-effector in the base frame with XYZ euler angles [alpha beta gamma] and a linear translation of [x y z]. The code then uses forwardKinematicsDisplay to generate a configuration object for the RigidBodyTree object ‘Robot’ to display. There is a section that generates a T^{B}_{EE} exactly as defined by a pure homogeneous transform of XYZ euler angles and linear translation as specified by desEEPos. Then it takes the computed T^{B}_{EE} and compares it with the “true” target element by element, thereby creating a matrix that defines the accuracy of the calculation. Due to the nature of transcendental equations, sometimes the solver has issues with computing the correct set of joint angles. In this case simply recomputing will usually do the trick. There can be variations in the length of the time taken to solve the equations, which is also due to the inconsistencies of variable precision arithmetic solvers.

4. Inverse Kinematics using the Jacobian

This section demonstrates the use of the inverseKinematicsUsingJacobian function included in the code package. By altering the values of desiredPos = [alpha beta gamma x y z] for the end effector of the manipulator, you can compute any required joint positions for the manipulator(written as a homogeneous transform of the end-effector in the base frame with XYZ euler angles [alpha beta gamma] and a linear translation of [x y z]. This approach uses similar visualization techniques to the previous sections. To accomplish this task, the team used a simple gradient descent approach to map the error in the cartesian space into the joint space, then iterate by a step of .1. This approach can lead to some extremely long times for certain poses as the function might get stuck in a local minima for a long time because the error approaches zero, but doesn’t actually match the required 5% error margins for the code. The equation of use can be found in the description of the function.

5. Workspace Visualization

This section demonstrates the visualization of the workspace of a KUKA IIWA14 7-DOFManipulator. There is only one function-workspaceVisualizer-which accomplishes the entire task. One can read more about the function in the documentation.

Documentation

MakeTransform(theta,d,a,alpha)

This function accepts the 4 DH values for a given frame in a robot and returns a 4x4 matrix that represents the homogeneous transform of that joint frame expressed in the previous joint frame. The transformation is defined as follows:

$$\boxed{{}^{i-1}T_i = \left[\begin{array}{ccc|c} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ \hline 0 & 0 & 0 & 1 \end{array}\right]}$$

Example:

```
>> MakeTransform(0,1,pi/2,2)

ans =

    1.0000         0         0    1.5708
         0   -0.4161   -0.9093         0
         0    0.9093   -0.4161    1.0000
         0         0         0    1.0000
```

forwardKinematicsCalculation(angles)

This function accepts a list of angles from θ_1 to θ_7 and returns a 4x4 matrix with values of a homogeneous transform of the end-effector in the base frame for a KUKA IIWA-14 7-DOF manipulator. It takes advantage of the following equation to sequentially create T^{B}_{EE} :

$$T^{0}_{2} = T^{0}_{1} * T^{1}_{2}$$

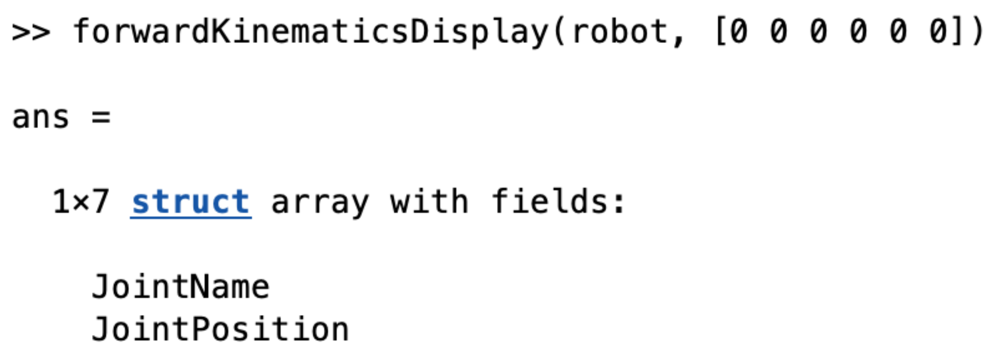
Example:

```
>> forwardKinematicsCalculation([pi/2 pi pi 0 pi pi/4 pi])

ans =

   -0.0000    1.0000    0.0000   -0.0000
    0.7071    0.0000   -0.7071   -0.0891
   -0.7071   -0.0000   -0.7071   -0.5491
         0         0         0    1.0000
```

forwardKinematicsDisplay(robot, jointAngles)

This function creates a configuration object for the RigidBodyTree object parameter ‘robot’ that represents the angles that are entered as the parameter ‘jointAngles’. The returned object has names and angles for each joint.

Example:



```
>> forwardKinematicsDisplay(robot, [0 0 0 0 0 0])

ans =

  1×7 struct array with fields:

    JointName
    JointPosition
```

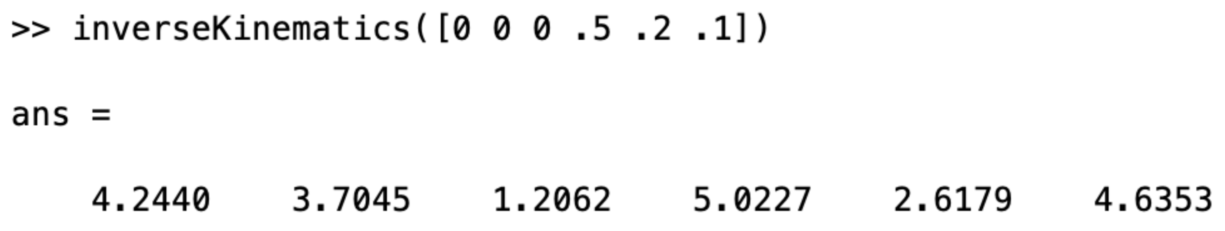
inverseKinematics(pose)

This function returns a 1x6 array that represents the joint angles of the first six joints of a KUKA IIWA-14 7-DOF manipulator as defined by pose = [alpha beta gamma x y z] evaluated as a homogeneous transform of the end-effector frame in the base frame with XYZ euler angles [alpha beta gamma] and a linear translation of [x y z]. The function takes advantage of Variable Precision Arithmetic Solving, a method that numerically calculates the solution to equations without the need of a formal analytical solution. First, the solver symbolically creates the T_{EE}^{B} using variables theta1 - theta7. It substitutes theta7 for 0 to reduce complexity, and then attempts to numerically solve for the following six equations:

$$\begin{cases} \beta = \mathrm{Atan2}(-r_{31}, \sqrt{r^2{}_{11} + r^2{}_{21}}) \\ \alpha = \mathrm{Atan2}(r_{21} / c\beta, r_{11} / c\beta) \\ \gamma = \mathrm{Atan2}(r_{32} / c\beta, r_{33} / c\beta) \end{cases} .$$

X = r(1,4);
Y = r(2,4);
Z = r(3,4)

Where r is the matrix T^{B}_{EE} . This function will throw an exception if you try to compute a point outside of its reachable workspace. It will timeout if the point is within your reachable workspace but the orientation is unachievable.
Example:



```
>> inverseKinematics([0 0 0 .5 .2 .1])

ans =

    4.2440    3.7045    1.2062    5.0227    2.6179    4.6353
```

makeTransformOfEEinB(angles)

Function that is wrapped (and more aptly named by) forwardKinematicsCalculation. It is the function that does the work of forwardKinematicsCalculation, and contains the DH parameters of a KUKA IIWA-14 7-DOF manipulator. The math and functionality are outlined in forwardKinematicsCalculation. It only exists for some debugging convenience throughout the development process and it doesn't really serve any purpose in the final project. You are welcome to use it for the exact same purpose as forwardKinematicsCalculation, except note that it requires seven joint angles.

makeTransformWithDesiredOrientation([alpha beta gamma x y z])

Function that returns a 4x4 matrix that represents the homogeneous transform described as the 3x3 rotation matrix in the below equation, a fourth column defined by the position vector [x;y;z], and by a fourth row [0 0 0 1]. This function defines the TRUE answer for a desired position and orientation of the manipulator's end effector, and is used to calculate the accuracy of the computed joint angles for the same desired position and orientation.

$$= \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}$$

inverseKinematicsUsingJacobian(robot, [alpha beta gamma x y z])

This function returns a 1x6 array that represents the joint angles of the first six joints of a KUKA IIWA-14 7-DOF manipulator as defined by pose = [alpha beta gamma x y z] evaluated as a homogeneous transform of the end-effector frame in the base frame with XYZ euler angles [alpha beta gamma] and a linear translation of [x y z]. The function takes advantage of gradient descent using the below equation. By taking the error in the cartesian space, mapping it into the joint space, and then incrementally moving towards the goal.

$$Q_{new} = Q_{old} + \Delta x * J * K$$

Here, Δx is the error between the current end-effector position and the desired, J is the geometric jacobian of the pose of the robot, and K is .1.

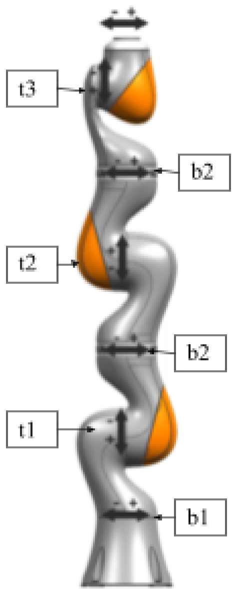
workspaceVisualizer(robot)

For the workspace calculation the URDF file is first uploaded to the workspace. The angle limits for each joint are specified as follows:

Angles t1,t2,t3 = 210° to -30°
Angles b1,b2,b3=180° to -180°

Further the link lengths are specified as defined by the manufacturer as follows:

l1=0.360
l2=0.420
l3=0.400
l4=0.276



A for-loop is then run to find the transformation matrices of each joint frame. The function ‘Maketranform’ is used, wherein the inputs for theta are random values within the specified joint limits. The final transformation matrix is found from which the values for x,y and z are plotted.
The resulting plot gives us the workspace of the robot.