Abstract

In this exercise you learn how to calculate the forward and inverse kinematics of an ABB robot arm. A Matlab visualization of the robot arm is provided. You will have to implement the tools to compute the forward and inverse kinematics in your Matlab script. During this you will exercise how to work with different representations of the end-effector orientation and how to check whether your implementations are correct. In the end, you will apply inverse-kinematics in order to have the robot follow a desired path with the end-effector.

1 Introduction

The following exercise is for the ABB IRB 120 depicted in figure 2. It is a 6-link robotic manipulator with a fixed base. During the exercise you will implement a lot of different Matlab functions - test them carefully since the following tasks are often dependent on them. To help you with this, we have provided the script prototypes at bitbucket.org/ethz-asl-lr/robotdynamics_exercise1 together with a visualizer of the manipulator.
2 Forward Kinematics

Throughout this document, we will employ $I$ for denoting the inertial world coordinate system (coordinate system $P_0$ in figure 2) and $E$ for the coordinate system attached to the end-effector (coordinate system $P_6$ in figure 2).

Exercise 2.1
Define the generalized coordinates for the ABB IRB120. Please remember that generalized coordinates should be complete (the configuration of the robot should be fully specified by the generalized coordinates) and independent (no coordinate is a pure function of the others).

Solution 2.1
The generalized coordinates can be chosen as the single joint angles between subsequent links. Any other complete and independent linear combination of the single joint angles is also a valid solution.

\[ \vec{\theta} = (\theta_1, \ldots, \theta_6)^T \in \mathbb{R}^{6\times1} \] (1)

Exercise 2.2
Find the homogenous transformations matrices $T_{k-1,k}(\theta_k)$, $\forall k = 1, \ldots, 6$. Additionally, find the constant homogeneous transformations between the inertial frame and frame 0 ($T_{0I}$) and between frame 6 and the end-effector frame ($T_{6E}$). Please implement the following functions:

1 function $T_{0I} = \text{getTransformI0}()$
2 % Input: void
3 % Output: homogeneous transformation Matrix from the inertial ... frame 0 to frame I. $T_{I0}$
4 end
5

Figure 1: ABB IRB 120 with coordinate systems and joints

Solution 2.2

Remember that a homogeneous transformation matrix is expressed in the form:

$$\mathbf{T}_{hk}(\theta_k) = \begin{bmatrix} \mathbf{R}_{hk}(\theta_k) & h\mathbf{r}_{hk}(\theta_k) \\ 0_{1 \times 3} & 1 \end{bmatrix}$$  \hspace{1cm} (2)$$

For the ABB IRB 120, each $\mathbf{T}_{hk}(\theta_k)$ is composed by an elementary rotation a single joint axis and a translation defined by the manipulator kinematic parameters. By defining the elementary rotation matrices about each axis as:

$$\mathbf{R}_z(\alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \hspace{1cm} (3)$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \hspace{1cm} (4)$$

$$\mathbf{R}_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{bmatrix} \hspace{1cm} (5)$$

one can write:
\[
T_{01}(\theta_1) = \begin{bmatrix}
R_z(\theta_1) & a_{01}(\theta_1) & 0 \\
0_{1 \times 3} & 1
\end{bmatrix}
\] (6)

\[
T_{12}(\theta_2) = \begin{bmatrix}
R_y(\theta_2) & 0 & a_{12}(\theta_2) \\
0_{1 \times 3} & 1
\end{bmatrix}
\] (7)

\[
T_{23}(\theta_3) = \begin{bmatrix}
R_y(\theta_3) & 0 & 2a_{23}(\theta_3) \\
0_{1 \times 3} & 1
\end{bmatrix}
\] (8)

\[
T_{34}(\theta_4) = \begin{bmatrix}
R_x(\theta_4) & 3a_{34}(\theta_4) & 0 \\
0_{1 \times 3} & 1
\end{bmatrix}
\] (9)

\[
T_{45}(\theta_5) = \begin{bmatrix}
R_y(\theta_5) & 4a_{45}(\theta_5) & 0 \\
0_{1 \times 3} & 1
\end{bmatrix}
\] (10)

\[
T_{56}(\theta_6) = \begin{bmatrix}
R_x(\theta_6) & 5a_{56}(\theta_6) & 0 \\
0_{1 \times 3} & 1
\end{bmatrix}
\] (11)

Finally, the constant homogeneous transformations \(T_{IE}\) and \(T_{6E}\) are simply the identity matrix \(I_{4 \times 4}\).
Exercise 2.3
Find the end-effector position \( \mathbf{r}_{IE} = \mathbf{r}_{IE}(\mathbf{\theta}) \). Please implement the following function:

```matlab
function \( \mathbf{r} = \text{jointToPosition}(x) \)
% Input: joint angles
% Output: position of end-effector w.r.t. inertial frame. \( \mathbf{r}_{IE} \)
end
```

Solution 2.3
The end-effector position is given by the direct kinematics, represented in matrix form by the homogeneous transformation \( \mathbf{T}_{IE}(\mathbf{\theta}) \), which can be found by successive concatenation of coordinate frame transformations.

\[
\mathbf{T}_{IE}(\mathbf{\theta}) = \mathbf{T}_{I0} \cdot \left( \prod_{k=1}^{6} \mathbf{T}_{k-1,k} \right) \cdot \mathbf{T}_{6E} = \begin{bmatrix} \mathbf{R}_{IE} & \mathbf{i} \mathbf{r}_{IE}(\mathbf{\theta}) \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \tag{12}
\]

The end-effector position can then be found by selecting the fourth column of \( \mathbf{T}_{IE}(\mathbf{\theta}) \).

\[
\begin{bmatrix} \mathbf{r}(\mathbf{\theta}) \\ 1 \end{bmatrix} = \mathbf{T}_{IE}(\mathbf{\theta}) \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{13}
\]
Exercise 2.4

What is the end-effector position for \( \vec{\theta} = \begin{pmatrix} \pi/6 \\ \pi/6 \\ \pi/6 \\ \pi/6 \\ \pi/6 \\ \pi/6 \end{pmatrix} \) ? Use matlab to display it.

Solution 2.4

From the direct kinematics equations found earlier, it is:

\[
\vec{r}_{IE} = \vec{r}_{IE}(\vec{\theta}) = \begin{pmatrix} 0.2948 \\ 0.1910 \\ 0.2277 \end{pmatrix}.
\] (14)

Exercise 2.5

Find the analytical expression of the end-effector linear velocity \( \vec{v}_{IE} = \vec{v}_{IE}(\vec{\theta}) = \frac{d\vec{r}_{IE}(\vec{\theta})}{dt} \).

Solution 2.5

Consider the coordinate frames shown in Fig. 2. Frame 0 is fixed, while frame 1 has a linear velocity \( \vec{v}_{01} \) and angular velocity \( \vec{\omega}_{1} \). Consider a point \( P \) that is fixed in
The linear velocity $v^P_{02}$ of point $P$ with respect to the fixed frame $0$ is given by:

$$v^P_{02} = v_{01} + \omega_1 \times r_{12}.$$  

(15)

If point $P$ is fixed in frame 1, it is $r_{12} = 0$. With this result in mind, consider now a planar two link robot arm with two revolute joints. The coordinate frames are chosen as in Fig.2. Reasoning as in Eq.(??), the linear velocity at the end of the kinematic chain can be found by propagating the velocity contributions from the fixed frame $0$. Hence, one has:

$$v_{01} = v_0 + \omega_0 \times r_{01}$$

$$v_{02} = v_{01} + \omega_1 \times r_{12}$$

$$v_{0E} = v_{02} + \omega_2 \times r_{2E}$$

(16)

Figure 3: The kinematic structure of a planar two link robot arm.

Combining these results with the fact the frame $0$ is fixed (i.e. $\omega_0 = 0$, $v_0 = 0$), the end-effector linear velocity is given by:

$$v_{0E} = \omega_1 \times r_{12} + \omega_2 \times r_{2E}$$

(17)

This result can be extended to the case of the ABB IRB 120, yielding:

$$I \ddot{r}^E_{1E} = I \omega_1 \times I r_{12} + I \omega_2 \times I r_{23} + \cdots + I \omega_5 \times I r_{56} + I \omega_6 \times I r_{6E}$$

$$= I \ddot{v}_{01} + I \ddot{v}_{12} + \cdots + I \ddot{v}_{56} + I \ddot{v}_{6E}$$

(18)

Exercise 2.6

Find the end-effector rotation matrix $R_{IE} = R_{IE}(\dot{\theta})$. Please implement the following function:

```matlab
1 function R = jointToRotMat(x)
2 % Input: joint angles
3 % Output: rotation Matrix of the end-effector. R_IE
4 end
```
Solution 2.6
From the structure of the direct kinematics equations found earlier, it follows that the end-effector rotation matrix is obtained by extracting the first three rows and the first three columns from $\mathbf{T}_{IE}(\vec{\theta})$. This operation can be compactly written in matrix form:

$$
\mathbf{R}_{IE}(\vec{\theta}) = [\mathbf{I}_{3\times3} \quad \vec{0}_{3\times1}] \cdot \mathbf{T}_{IE}(\vec{\theta}) \cdot [\mathbf{I}_{3\times3} \quad \vec{0}_{1\times3}].
$$

(19)

Exercise 2.7
Find the quaternion representing the attitude of the end-effector $\mathbf{q}_{IE} = \mathbf{q}_{IE}(\vec{\theta})$. Please also implement the following function:

- Two functions for converting from quaternion to rotation matrices and vice-versa. Test these by converting from quaternions to rotation matrices and back to quaternions.
- The quaternion multiplication $\mathbf{q} \otimes \mathbf{p}$ (unfortunately the quaternion implementation in the aerospace toolbox of Matlab uses Hamilton convention, hence use the definition from the lecture slides, where the JPL [1] convention has been used).
- The function rotating a vector with a given quaternion. This can be implemented in different ways which can be tested with respect to each other. The easiest way is to transform the quaternion to the corresponding rotation matrix (by using the function from above) and then multiply the matrix with the vector to be rotated.

Also check that your two representations for the end-effector orientation match with each other. In total you should write the following five functions:

```
function R = jointToRotMat(x)
% Input: joint angles
% Output: rotation Matrix of the end-effector. R_{EI}
T01 = getTransformI0();
T12 = jointToTransform01(x(1));
T23 = jointToTransform23(x(2));
T34 = jointToTransform23(x(3));
T45 = jointToTransform23(x(4));
T56 = jointToTransform23(x(5));
T6E = getTransform6E();
TIE = T01*T12*T23*T34*T45*T56*T6E;
R = T_{IE}(1:3,1:3);
end
```

```
function q = jointToQuat(x)
% Input: joint angles
% Output: quaternion representing the orientation of the ...
end
```

```
function R = quatToRotMat(q)
% Input: quaternion [w x y z]
% Output: corresponding rotation matrix
```
function q = rotMatToQuat(R)
% Input: rotation matrix
% Output: corresponding quaternion [w x y z]
end

function q_{AC} = quatMult(q_{AB},q_{BC})
% Input: two quaternions to be multiplied
% Output: output of the multiplication
end

function B_{2r} = rotVecWithQuat(q_{BA},A_{r})
% Input: the orientation quaternion and the coordinate of the vector to be mapped
% Output: the coordinates of the vector in the target frame
end

Solution 2.7

\[
R_{IE}(\vec{\theta}) = R_{I0} \cdot \left( \prod_{k=1}^{6} R_{k-1,k} \right) \cdot R_{6E} \tag{20}
\]

\[
q(\vec{\theta}) = \begin{pmatrix}
\sqrt{(1 + T)/4} \\
(R_{23} - R_{32})/(4 \cdot q_{1}) \\
(R_{31} - R_{13})/(4 \cdot q_{1}) \\
(R_{12} - R_{21})/(4 \cdot q_{1})
\end{pmatrix} \tag{21}
\]

where \( T = \text{Trace}(R_{IE}). \)

function q = jointToQuat(x)
% Input: joint angles
% Output: quaternion representing the orientation of the end-effector. q_{EI}
end

R = jointToRotMat(x);
q = rotMatToQuat(R);
end

function R = quatToRotMat(q)
% Input: quaternion [w x y z]
% Output: corresponding rotation matrix
q = q(:);
R = (2*q(1)*eye(3) - 2*q(1)*skewMatrix(q(2:end)) + ... 2*q(2:end)*q(2:end)');
end

function q = rotMatToQuat(R)
% Input: rotation matrix
% Output: corresponding quaternion [w x y z]
q = JPL(q);
q1 = sqrt((1+trace(R))/4);
det = [q1; (R(2,3)-R(3,2))/(4*q1); (R(3,1)-R(1,3))/(4*q1); (R(1,2)-R(2,1))/(4*q1)];
end

function q_{AC} = quatMult(q_{AB},q_{BC})
% Input: two quaternions to be multiplied
% Output: output of the multiplication
end
q = q_{AB};

p = q_{BC};

% JPL
q_{AC} = \[\begin{array}{c}
-q(2) \cdot p(2) - q(3) \cdot p(3) - q(4) \cdot p(4) + q(1) \cdot p(1); \\
q(1) \cdot p(2) + q(4) \cdot p(3) - q(3) \cdot p(4) + q(2) \cdot p(1); \\
-q(4) \cdot p(2) + q(1) \cdot p(3) + q(2) \cdot p(4) + q(3) \cdot p(1); \\
q(3) \cdot p(2) - q(2) \cdot p(3) + q(1) \cdot p(4) + q(4) \cdot p(1); 
\end{array}\];

end

function B_{r} = rotVecWithQuat(q_{BA}, A_{r})
% Input: the orientation quaternion and the coordinate of the vector to be mapped
% Output: the coordinates of the vector in the target frame
R_{BA} = quatToRotMat(q_{BA});
B_{r} = R_{BA} \cdot A_{r};
end

Exercise 2.8

Find the analytical expression of the end-effector rotational velocity $\vec{\omega}_{IE} = I\vec{\omega}_{IE}(\dot{\theta})$.

Solution 2.8

The end-effector rotational velocity $\vec{\omega}_{IE}$ is obtained by summing the single joint velocity contributions:

$$I\vec{\omega}_{IE} = I\vec{\omega}_{01} + I\vec{\omega}_{12} + \cdots + I\vec{\omega}_{56} + I\vec{\omega}_{6E}$$  \hspace{1cm} (22)
\section{Inverse Kinematics}

\textbf{Exercise 3.1}

Derive the Jacobians of the end-effector position and orientation based on the linear and rotational velocity derived above in exercise 2. The Jacobians should depend on the minimal coordinates only. Remember that:

\begin{equation}
\mathbf{v}_{IE} = \mathbf{J}_P(\vec{\theta}) \dot{\vec{\theta}} 
\end{equation}

\begin{equation}
\mathbf{\omega}_{IE} = \mathbf{J}_R(\vec{\theta}) \dot{\vec{\theta}}
\end{equation}

Please implement the following two functions:

\begin{verbatim}
function J_P = jointToPosJac(x)
% Input: joint angles
% Output: Jacobian of the end-effector position
end

function J_R = jointToRotJac(x)
% Input: joint angles
% Output: Jacobian of the end-effector orientation
end
\end{verbatim}

\textbf{Solution 3.1}

The translation and rotation Jacobians can be evaluated starting from the results that were obtained in the previous exercises. By combining the analytical expressions of the linear and angular end-effector velocities, one has:

\begin{equation}
\mathbf{v}_{IE} = \mathbf{v}_{01} + \mathbf{v}_{12} + \cdots + \mathbf{v}_{56} + \mathbf{v}_{6E}
\end{equation}

\begin{equation}
= \mathbf{i} \omega_1 \times \mathbf{r}_{12} + \mathbf{i} \omega_2 \times \mathbf{r}_{23} + \cdots + \mathbf{i} \omega_5 \times \mathbf{r}_{56} + \mathbf{i} \omega_E \times \mathbf{r}_{6E}
\end{equation}

\begin{equation}
= \mathbf{i} \omega_1 \times (\mathbf{r}_{12} - \mathbf{r}_{11}) + \mathbf{i} \omega_2 \times (\mathbf{r}_{13} - \mathbf{r}_{12}) + \cdots + \mathbf{i} \omega_5 \times (\mathbf{r}_{16} - \mathbf{r}_{15})
\end{equation}

\begin{equation}
= \mathbf{i} \dot{\omega}_0 \times (\mathbf{r}_{12} - \mathbf{r}_{11})
\end{equation}

\begin{equation}
+ (\mathbf{i} \dot{\omega}_1 + \mathbf{i} \dot{\omega}_2) \times (\mathbf{r}_{13} - \mathbf{r}_{12})
\end{equation}

\begin{equation}
+ \cdots
\end{equation}

\begin{equation}
+ (\mathbf{i} \dot{\omega}_5 + \mathbf{i} \dot{\omega}_6) \times (\mathbf{r}_{16} - \mathbf{r}_{15})
\end{equation}

At this point it is worth noticing that it is \( \mathbf{i} \omega_{k-1,k} = \mathbf{i} \dot{\omega}_k \dot{\theta}_k \), where \( \dot{\omega}_k = \mathbf{R}_{1k} \cdot \vec{k} \dot{\omega}_k \) and \( \dot{\theta}_k \) represents the axis around which joint \( k \) can rotate. Recalling that:

\begin{equation}
\mathbf{i} \dot{\omega}_k = \mathbf{i} \omega_{k-1} + \mathbf{i} \dot{\omega}_{k-1,k},
\end{equation}

one has:

\begin{equation}
\mathbf{v}_{IE} = (\mathbf{i} \dot{\omega}_0 + \mathbf{i} \dot{\omega}_0) \times (\mathbf{r}_{12} - \mathbf{r}_{11})
\end{equation}

\begin{equation}
+ (\mathbf{i} \dot{\omega}_1 + \mathbf{i} \dot{\omega}_2) \times (\mathbf{r}_{13} - \mathbf{r}_{12})
\end{equation}

\begin{equation}
+ \cdots
\end{equation}

\begin{equation}
+ (\mathbf{i} \dot{\omega}_5 + \mathbf{i} \dot{\omega}_6) \times (\mathbf{r}_{16} - \mathbf{r}_{15})
\end{equation}

\begin{equation}
= \mathbf{i} \dot{\omega}_1 \dot{\theta}_1 \times (\mathbf{r}_{12} - \mathbf{r}_{11})
\end{equation}

\begin{equation}
+ (\mathbf{i} \dot{\omega}_1 \dot{\theta}_1 + \mathbf{i} \dot{\omega}_2 \dot{\theta}_2) \times (\mathbf{r}_{13} - \mathbf{r}_{12})
\end{equation}

\begin{equation}
+ \cdots
\end{equation}

\begin{equation}
+ (\mathbf{i} \dot{\omega}_1 \dot{\theta}_1 + \cdots + \mathbf{i} \dot{\omega}_6 \dot{\theta}_6) \times (\mathbf{r}_{16} - \mathbf{r}_{15})
\end{equation}
Expanding and reordering the terms in the last equation, one has:
\[
\vec{v}_{IE} = I_0 \hat{\omega}_1 \times (I_{rIE} - I_{rI1}) \\
+ I_0 \hat{\omega}_2 \times (I_{rIE} - I_{rI2}) \\
+ \ldots \\
+ I_0 \hat{\omega}_6 \times (I_{rIE} - I_{rI6})
\]
which, rewritten in matrix form, gives:
\[
\vec{v}_{IE} = \begin{bmatrix}
I_0 \hat{\omega}_1 \times (I_{rIE} - I_{rI1}) \\
I_0 \hat{\omega}_2 \times (I_{rIE} - I_{rI2}) \\
\vdots \\
I_0 \hat{\omega}_6 \times (I_{rIE} - I_{rI6})
\end{bmatrix}
\begin{bmatrix}
\dot{\theta}_1 \\
\dot{\theta}_2 \\
\vdots \\
\dot{\theta}_6
\end{bmatrix}
\]
where \( J_P(\vec{\theta}) \) is the translation Jacobian matrix that projects a vector from the joint velocity space to the cartesian linear velocity space.

Using the results obtained by solving Exercise 8, and taking into account that \( I_0 \hat{\omega}_0 \) and \( I_0 \hat{\omega}_6 \) are both equal to zero, one has:
\[
I_0 \hat{\omega}_{IE} = I_0 \hat{\omega}_1 + I_0 \hat{\omega}_2 + \ldots + I_0 \hat{\omega}_6
\]
\[
= I_0 \hat{\omega}_1 \dot{\theta}_1 + I_0 \hat{\omega}_2 \dot{\theta}_2 + \ldots + I_0 \hat{\omega}_6 \dot{\theta}_6
\]
\[
= \begin{bmatrix}
I_0 \hat{\omega}_1 \\
I_0 \hat{\omega}_2 \\
\vdots \\
I_0 \hat{\omega}_6
\end{bmatrix}
\dot{\theta}
\]
where \( J_R(\vec{\theta}) \) is the rotation Jacobian matrix that projects a vector in the joint velocity space to the cartesian angular velocity space.

We now inspect a second method to derive the rotation Jacobian. It can be shown that the time derivative of the rotation matrix \( R_{IE}(\vec{\theta}) \) is given by:
\[
\dot{R}_{IE}(\vec{\theta}) = S(I_0 \hat{\omega}_{IE}) \cdot R_{IE}(\vec{\theta})
\]
where \( S(I_0 \hat{\omega}_{IE}) \) is the skew matrix operator defined as:
\[
S(\vec{\alpha}) = \begin{bmatrix}
0 & -\alpha_z & \alpha_y \\
\alpha_z & 0 & -\alpha_x \\
-\alpha_y & \alpha_x & 0
\end{bmatrix}
\]
Using the chain rule, one can also write:
\[
\dot{R}(\vec{\theta}) = \frac{\partial R(\vec{\theta})}{\partial \vec{\theta}} \cdot \dot{\vec{\theta}}
\]
Combining the last two equations and using the orthonormality property of rotation matrices, one has:
\[
S(I_0 \hat{\omega}_{IE}) = \dot{R}_{IE}(\vec{\theta}) \cdot R_{IE}(\vec{\theta})^T.
\]
The end-effector velocity \( I_0 \hat{\omega}_{IE} \) can be easily extracted from \( S(I_0 \hat{\omega}_{IE}) \). The rotational Jacobian can now be evaluated as:
\[
J_R(\vec{\theta}) = \frac{\partial I_0 \hat{\omega}_{IE}}{\partial \vec{\theta}}
\]
function J_P = jointToPosJac(x)

% Input: joint angles
% Output: Jacobian of the end-effector position
T_I0 = getTransformI0();
T_01 = jointToTransform01(x(1));
T_12 = jointToTransform12(x(2));
T_23 = jointToTransform23(x(3));
T_34 = jointToTransform34(x(4));
T_45 = jointToTransform45(x(5));
T_56 = jointToTransform56(x(6));
T_I1 = T_I0*T_01;
T_I2 = T_I1*T_12;
T_I3 = T_I2*T_23;
T_I4 = T_I3*T_34;
T_I5 = T_I4*T_45;
T_I6 = T_I5*T_56;

R_I1 = T_I1(1:3,1:3);
R_I2 = T_I2(1:3,1:3);
R_I3 = T_I3(1:3,1:3);
R_I4 = T_I4(1:3,1:3);
R_I5 = T_I5(1:3,1:3);
R_I6 = T_I6(1:3,1:3);

r_I1 = T_I1(1:3,4);
r_I2 = T_I2(1:3,4);
r_I3 = T_I3(1:3,4);
r_I4 = T_I4(1:3,4);
r_I5 = T_I5(1:3,4);
r_I6 = T_I6(1:3,4);

omega_hat_1 = [0 0 1]';
omega_hat_2 = [0 1 0]';
omega_hat_3 = [0 1 0]';
omega_hat_4 = [1 0 0]';
omega_hat_5 = [0 1 0]';
omega_hat_6 = [1 0 0]';
r_I_IE = jointToPosition(x);

J_P = [ cross(R_I1*omega_hat_1, r_I_IE - r_I1) ...
        cross(R_I2*omega_hat_2, r_I_IE - r_I2) ...
        cross(R_I3*omega_hat_3, r_I_IE - r_I3) ...
        cross(R_I4*omega_hat_4, r_I_IE - r_I4) ...
        cross(R_I5*omega_hat_5, r_I_IE - r_I5) ...
        cross(R_I6*omega_hat_6, r_I_IE - r_I6) ...
    ];
end

function J_R = jointToRotJac(x)

% Input: joint angles
% Output: Jacobian of the end-effector orientation
T_I0 = getTransformI0();
T_01 = jointToTransform01(x(1));
T_12 = jointToTransform12(x(2));
T_23 = jointToTransform23(x(3));
T_34 = jointToTransform34(x(4));
T_45 = jointToTransform45(x(5));
T_56 = jointToTransform56(x(6));
T_I1 = T_I0*T_01;
T_I2 = T_I1*T_12;
T_I3 = T_I2*T_23;
T_I4 = T_I3*T_34;
T_I5 = T_I4*T_45;
T_I6 = T_I5*T_56;
R_{I1} = T_{I1}(1:3,1:3);
R_{I2} = T_{I2}(1:3,1:3);
R_{I3} = T_{I3}(1:3,1:3);
R_{I4} = T_{I4}(1:3,1:3);
R_{I5} = T_{I5}(1:3,1:3);
R_{I6} = T_{I6}(1:3,1:3);

\omega^\hat{1} = [0 0 1]';
\omega^\hat{2} = [0 1 0]';
\omega^\hat{3} = [0 1 0]';
\omega^\hat{4} = [1 0 0]';
\omega^\hat{5} = [0 1 0]';
\omega^\hat{6} = [1 0 0]';

J_{R} = [ R_{I1}\cdot\omega^\hat{1} ... 
R_{I2}\cdot\omega^\hat{2} ... 
R_{I3}\cdot\omega^\hat{3} ... 
R_{I4}\cdot\omega^\hat{4} ... 
R_{I5}\cdot\omega^\hat{5} ... 
R_{I6}\cdot\omega^\hat{6} ... ];

Exercise 3.2

Find a set of minimal coordinates such that the end-effector position is at \( \vec{r}^* = [0.5649 \ 0 \ 0.5509]^T \) while its orientation is kept as identity. To this end you will have to implement an iterative inverse kinematics algorithm in Matlab. The algorithm can be summarized as follows:

1. Initialize the iterations to \( i = 0 \) and start with some initial guess \( \vec{\theta}_0 \)

2. Compute difference between desired values and actual values \( \Delta \vec{r}_i = \vec{r}^* - \vec{r}(\vec{\theta}_i) \), and \( \Delta q_i = q^* - q(\vec{\theta}_i) \), with \( q^* = (1, 0, 0, 0) \).

3. Construct the following system of linear equations and solve for \( \Delta \vec{\theta}_i \).

\[
\begin{pmatrix}
\Delta \vec{r}_i \\
\Delta q_i
\end{pmatrix} =
\begin{bmatrix}
J_P(\vec{\theta}_i) & \xi(q(\vec{\theta}_i))J_R(\vec{\theta}_i)
\end{bmatrix}
\Delta \vec{\theta}_i
\] (36)

4. Apply the correction \( \Delta \vec{\theta}_i \) onto \( \vec{\theta}_i \) with \( \vec{\theta}_{i+1} = \vec{\theta}_i + \Delta \vec{\theta}_i \)

5. Check if the correction is smaller then a specific threshold \( \epsilon \), if not set \( i := i + 1 \) and continue with item 2.

Please implement your solution in the following function. You will also need to implement the function \( \xi(q(\vec{\theta})) \) which maps the rotational velocity to the quaternion derivative.

```
function X = xiMatrix(q)
% Input: quaternion
% Output: 4x3 matrix mapping local rotational velocities to ... quaternion derivatives
end
```

```
function x = inverseKinematics(r_des,q_des,x_init,epsilon)
% Input: desired end-effector position, desired end-effector ... orientation (quaternion), initial guess for joint angles, ... threshold for stopping-criterion
% Output: joint angles with match desired end-effector position ... and orientation
end
```
Solution 3.2
Recall that the mapping between the time derivative of the quaternion $\dot{q}_{EI}$ and the local angular velocity $E\tilde{\omega}_{IE}$ is given by:

$$\dot{q}_{EI} = \frac{1}{2}\Xi(q_{EI}(\hat{\theta})) \cdot E\tilde{\omega}_{IE}, \tag{37}$$

where

$$\Xi(q_{EI}(\hat{\theta})) = \begin{bmatrix} -\dot{q}_{EI} \\ q_{EI,0}1_{3x3} + \dot{q}_{BI} \end{bmatrix} \tag{38}$$

Since the rotation Jacobian that we derived earlier maps joint velocities to $I\tilde{\omega}_{IE}$, we will need multiply by the appropriate rotation matrix to obtain $E\tilde{\omega}_{IE}$.

```matlab
function X = xiMatrix(q)
% Input: quaternion
% Output: 4x3 matrix mapping local rotational velocities to ... quaternion derivatives
% Make sure that q is a column
q = q(:);
X = 0.5*[−q(2:end)'; q(1)*eye(3)+skewMatrix(q(2:end))];
end

function x = inverseKinematics(r_des,q_des_IE,x_init,epsilon)
% Input: desired end-effector position, desired end-effector ... orientation (quaternion), initial guess for joint angles, ... threshold for stopping-criterion
% Output: joint angles which match desired end-effector position ... and orientation
%% Setup
iterations = 0;
dth = inf(length(x_init),1);
max_iterations = 1000;
q_des_IE = q_des_IE(:);
r_des = r_des(:);
x_init = x_init(:);
q_des_EI = invertQuat(q_des_IE);
q_EI = invertQuat(jointToQuat(theta));
dr = r_des - jointToPosition(theta);
dq = q_des_EI-q_EI;
R_IE = jointToRotMat(theta);
A = [jointToPosJac(theta);
    xiMatrix(q_EI)*R_IE'*jointToRotJac(theta)];
dth = A\[dr; dq];
% Update
theta = theta + dth;
iterations = iterations+1;
```
Exercise 3.3
So far we always represented the orientation of the end-effector by means of the full rotation matrix or quaternions. This has the advantage that subsequent relative orientations can just be concatenated by multiplying the matrices or quaternions together. Now, in order to get a more compact and human-readable representation of the end-effector orientation we will express it by means of Euler-angles $\vec{\alpha} = (\alpha, \beta, \gamma)$ with the sequence z-y-x. To this end, please reimplement the inverse kinematics algorithm by using a new representation transformation $\Xi_{zyx}(\vec{\theta})$ for the rotation Jacobian. You should implement the following functions:

1. function a = jointToEulAng(x)
   % Input: joint angles
   % Output: corresponding euler angles with z-y-x sequence
2. end

3. function X = xiMatrixEuler(a)
   % Input: z-y-x Euler angles
   % Output: 3x3 matrix mapping global rotational velocities to ...
   % Euler angle derivatives
4. end

5. function x = inverseKinematicsEuler(r_des, a_des, x_init, epsilon)
   % Input: desired end-effector position, desired end-effector ...
   % orientation (Euler-angles), initial guess for joint angles, ...
   % threshold for stopping-criterion
   % Output: joint angles with match desired end-effector position ...
   and orientation
6. end

7. end
8. x = theta;
9. pos_error = r_des - jointToPosition(x);
10. quat_error = 1 - norm(quatMult(q_des_EI, ...
    quatInverse(jointToQuat(x))));
11. fprintf('Inverse kinematics terminated after %d ...
    iterations.\n',iterations);
12. fprintf('Position error: %e.\n',norm(pos_error));
13. fprintf('Attitude error: %e.\n',quat_error);
14. end

15. function q_inv = invertQuat(q)
16. % Input: a unit quaternion
17. % Output: the inverse of the input quaternion
18. q = q(:);
19. q_inv = [q(1); -q(2:end)];
20. q_inv = q_inv/norm(q_inv);
21. end

22. function S = skewMatrix(x)
23. % Input: a vector in R3
24. % Output: the skew matrix S(x)
25. S = [0 -x(3) x(2);
26.     x(3) 0 -x(1);
27.     -x(2) x(1) 0];
28. end

29. function r = jointToPosition(x)
30. % Input: joint angles
31. % Output: corresponding end-effector position
32. r = r_des + quatInverse(jointToQuat(x));
33. end

34. function q = jointToQuat(x)
35. % Input: joint angles
36. % Output: corresponding unit quaternion
37. q = quatMult(quatInverse(jointToQuat(x)), r_des_EI);
38. q = q/norm(q);
39. end

40. function a = jointToEulAng(x)
41. % Input: joint angles
42. % Output: corresponding euler angles with z-y-x sequence
43. end

44. function X = xiMatrixEuler(a)
45. % Input: z-y-x Euler angles
46. % Output: 3x3 matrix mapping global rotational velocities to ...
47. % Euler angle derivatives
48. end

49. function x = inverseKinematicsEuler(r_des, a_des, x_init, epsilon)
50. % Input: desired end-effector position, desired end-effector ...
51. % orientation (Euler-angles), initial guess for joint angles, ...
52. % threshold for stopping-criterion
53. % Output: joint angles with match desired end-effector position ...
54. % and orientation
55. end

16
Solution 3.3

```matlab
function a = jointToEulAng(x)
% Input: joint angles
% Output: corresponding euler angles with z-y-x sequence
R_IE = jointToRotMat(x);
[yaw, pitch, roll] = dcm2angle(R_IE,'zyx');
a = [yaw;
pitch;
roll];
end

function X = xiMatrixEuler(a)
% Input: Euler angles with z-y-x sequence
% Output: 3x3 matrix mapping global rotational velocities to ... Euler angle derivatives
ps = a(1);
th = a(2);
X = [cos(ps)*sin(th)/cos(th) sin(ps)*sin(th)/cos(th) 1;
     -sin(ps) cos(ps) 0;
     cos(ps)/cos(th) sin(ps)/cos(th) 0];
end

function x = inverseKinematicsEuler(r_des,a_des,x_init,epsilon)
% Input: desired end-effector position, desired end-effector ... orientation (Euler-angles), initial guess for joint angles, ... threshold for stopping-criterion
% Output: joint angles which match desired end-effector position ... and orientation
%
% Setup
iterations = 0;
dth = inf(length(x_init),1);
max_iterations = 1000;
a_des = a_des(:);
r_des = r_des(:);
x_init = x_init(:);

% Initialize with initial guess
theta = x_init;

% Iterate until terminating condition
while (norm(dth)>epsilon && iterations < max_iterations)
    dr = r_des - jointToPosition(theta);
    a_star = jointToEulAng(theta);
da = a_des-a_star;
A = [jointToPosJac(theta);
     xiMatrixEuler(a_star)*jointToRotJac(theta)];
dth = pinv(A)*[dr; da];
    iterations = iterations+1;
end
x = theta;
```
pos_error = r.des - jointToPosition(x);
eul_error = a.des - jointToEulAng(x);

fprintf('Inverse kinematics terminated after %d ...
  iterations.
',iterations);
fprintf('Position error: %e.
',norm(pos_error));
fprintf('Attitude error: %e.
',norm(eul_error));

end

Exercise 3.4
In this exercise you are going to implement a basic pose controller. The controller will act only on the kinematic level, i.e. it will produce end-effector velocities as a function of the current and desired end-effector pose.

References