University of Ljubljana
Faculty of electrical engineering
Laboratory of robotics

Subject

Robot kinematics and dynamics

Laboratory exercises

as. dr. Jure Rejc
jure.rejc@robo.fe.uni-lj.si
tel: (01) 4768 379

prof. dr. Matjaž Mihelj
matjaz.mihelj@robo.fe.uni-lj.si
tel: (01) 4768 373

Contents

ROBOT GEOMETRICAL MODEL

Each robot mechanism consists of individual robot segments connected with joints. The serial mechanisms (antropomorfe robot) have serial segments in regard to each other, but also parallel mechanisms exist. The subject of these exercises is only about serial mechanisms.

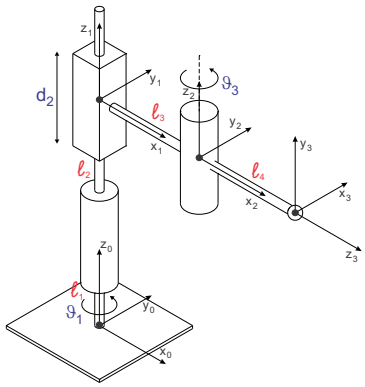

Figure 1.1: Simple RTR robot mechanism

Robot mechanisms can have cylindrical or spherical joints, but the most spread are rotational and translational joints, as shows Figure 1.1. Both most spread robot joints enable movement of neighbor robot joints in or around one axis of joint coordinate system. When the joint is translational type the movement of the joint is translation (marked as d) in one coordinate system axis direction. When the joint is rotational type the movement of the joint is rotation (marked as ϑ) around one

coordinate system axis. The joints can have a distance in regard to each other and this distance is marked with ℓ.

It was mentioned and showed (Figure 1.1) that each robot joint is represented with coordinate system. At the end of the serial mechanism is also attached a coordinate system. The pose of this coordinate system in regard to the base coordinate system (x_0, y_0, z_0) is dependent of joint variables (ϑ and d) and distances between joints marked with ℓ.

Usually the segments length are kept constant, but joint variables changes in translational or rotational manner. With joint movements the pose of the last coordinate system changes in regard to the base coordinate system. When we are talking about the movement of the robot joint without the causes of movement we are talking about **robot kinematics**.

To describe the pose of the joint coordinate system in regard to the previous coordinate system the homogenous matrices (dimension of 4×4) are used, usually marked with a letter **A**. In combination with the robot mechanism sketch it is possible to define **direct geometrical model** of the robot with an use of homogenous transformation matrices $\mathbf{A}_1 \dots \mathbf{A}_n$.

To define robot geometrical model several methods were developed, but the most used in the field of robotics is Denavit-Hartenberg method. This method defines the rules how to set the joint coordinate systems. Each coordinate system pose is described with four parameters (ϑ, d, a in α). The basic approach is to wite these parameters in a table in regard to the figure with properly set joint coordinate frames. The table is a base for writing homogenous transformation matrices **A**. To define individual homogenous transformation matrix **A** the user needs to follow transformation order shown by the equation 1.1.

$$A_i = Rot(\vartheta_i) \cdot Trans(d_i) \cdot Trans(a_i) \cdot Rot(\alpha_i) \tag{1.1}$$

The pose of the last coordinate system in regard to the base coordinate system is also described by homogenous transformation matrix, marked with a letter $\mathbf{T}_n$, where n represents the number of joints of the robot mechanism. The calculation of the matrix $\mathbf{T}_n$ is shown by the equation 1.2.

$$T_n = A_1 \cdot A_2 \cdot ... \cdot A_n \tag{1.2}$$

1 Direct kinematic model for the robot Stäubli RX90 and the robot KUKA youBot

Figure 1.2: Left: robot Stäubli RX90; right: robot KUKA youBot

The robot Stäubli RX90 (Figure 1.2, left) is an industrial robot with serial mechanism and 6 degrees of freedom (DOF). The robot mechanism can be driven by two controllers. The original industrial controller enables only the position control. The other, dedicated controller, developed in our laboratory enables also the force control of the robot mechansim. During the laboratory practice this robot will be used only once, when the exercise of transforming the forces and torques is scheduled. The robot is attached with force/torque sensor and enables attaching heavy weights.

The robot KUKA youBot (Figure 1.2) is used mainly in research and education institutions. It enables the open source architecture with low level hardware interfering. The robot can be split in two parts. The omni-directional mobile platform consisting of metal housing, mecanum wheels, four motors, personal computer and dedicated electronics. On the mobile platform it is attached a 5 segment robot arm with two finger gripper. During the laboratory practice both robot parts will be used.

2 The student tasks

- For both introduced robots with coordinate systems attached (Figures 1.3 and 1.4) fill in Table 1.1 and Table 1.2. Please, rewrite the table on the sheet of paper.
- When tables are properly filled, fill also the Matlab templates of the following functions.

  `function A = hdh(`ϑ`, d, a,` α`)` is using Denavit-Hartenberg method parameters ϑ, d, a and α to calculate homogenous transformation matrix `A`.

  `function A = dirkinA(q) and function A = dirkinA_staubli(q)` is using the joint variables (angle, position) written in vector **q** and calculates the array of matrices `A`. For calculation of a single matrix `A` the `hdh(`ϑ`, d, a,` α`)` function must be used.

  `function T6 = dirkinT6(A) and function T5 = dirkinT5(A)` is using the array of matrices `A` and calculates the pose of the robot top as a matrix `T6` and `T5`.

  In the further text the templates of the upper functions are written. Your task is to properly fill the function lines marked with `%%% STUDENT %%%`.

3 Necessary knowledge and goal of the exercise

The goal of the exercise is the use of Denavit-Hartenberg method to define kinematic model of the robot Stäubli RX90 and the robot KUKA youBot. To accomplish the goal it is necessary to master the use of homogenous transformation matrices and to be familiar with the Denavit-Hartenberg method. The results in a form of homogenous transformation matrix A and T will be used in all further laboratory exercises.

Figures 1.3 and 1.4 show robots Stäubli RX90 and KUKA youBot positioned in a reference configuration with coordinate systems attached in each joint. The orientation of the coordinate systems is based on Denavit-Hartenberg method. The figures are a startup point for developing direct geometrical or kinematic model with homogenous transformation matrices $\mathbf{A}_1 \dots \mathbf{A}_5$ ($\mathbf{A}_6$). These matrices are used to define the homogenous transformation matrices $\mathbf{T}_5$ and $\mathbf{T}_6$.

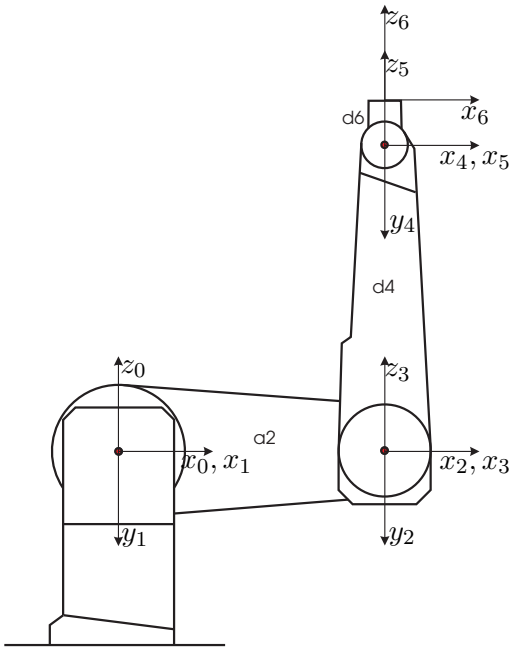

Figure 1.3: Robot Stäubli in reference configuration with Denavit-Hartenberg method based setting of the joint coordinate systems

i	ϑ_i	d_i	a_i	α_i
1				
2				
3				
4				
5				
6				

Table 1.1: Template for writing the scalar parameters following Denavit-Hartenberg method for the robot Stäubli RX90

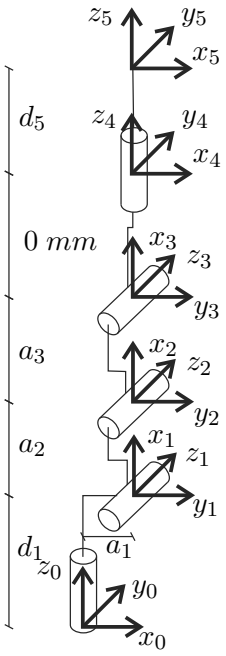

Figure 1.4: Robot KUKA youBot in reference configuration with Denavit-Hartenberg method based setting of the joint coordinate systems

i	ϑ_i	d_i	a_i	α_i
1				
2				
3				
4				
5				

Table 1.2: Template for writing the scalar parameters following Denavit-Hartenberg method for the robot KUKA youBot

4 Matlab templates

Template for function: **function A = hdh(ϑ, d, a, α)**

```
% th, d, a, al - Denavit-Hartenberg parameters of the robot (input):
% Theta(rotz), d(transz), a(transx), Alpha(rotx).
% A - homogenous transformation matrix (output).

% Cosine of the angle THETA.
    ct = cos(th);
% Sine of the angle THETA.
    st = sin(th);

% Rotational matrix around z axis.
    rz = ;                                              %%% STUDENT %%%

% Translational matrix along z and x axis.
    trans = ;                                           %%% STUDENT %%%

% Cosine of the angle ALPHA.
    cal = cos(al);
% Sine of the angle ALPHA.
    sal = sin(al);

% Rotational matrix around x axis.
    rx = ;                                              %%% STUDENT %%%

% Calculation of the homogenous transformation matrix.
    A = ;                                               %%% STUDENT %%%
```

Test results:

$$hdh(1,2,3,4) = \begin{bmatrix} 0.5403 & 0.5500 & -0.6368 & 1.6209 \\ 0.8415 & -0.3532 & 0.4089 & 2.5244 \\ 0 & -0.7568 & -0.6536 & 2.0000 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

Template for function: **function A = dirkinAstaubli(q)**

```
% q - Vector of joint variables for 6 joints (input).
% A(:,:,1) ... A(:,:,6) - Array of A matrices (output).

% Robot parameters.
    a2 = 0.45;
    d4 = 0.45;
    d6 = 0.085;

% Definition of A matrices
    A = zeros(4,4,6);
    A(:,:,1) = ;                                        %%% STUDENT %%%
    A(:,:,2) = ;                                        %%% STUDENT %%%
    A(:,:,3) = ;                                        %%% STUDENT %%%
    A(:,:,4) = ;                                        %%% STUDENT %%%
    A(:,:,5) = ;                                        %%% STUDENT %%%
    A(:,:,6) = ;                                        %%% STUDENT %%%
```

Template for function: **function A = dirkinA(q)**

```
% q - Vector of joint variables for 5 joints (input).
% A(:,:,1) ... A(:,:,5) - Array of A matrices (output).

% Robot parameters.
    d1=0.147;
    d5=0.117;
    a1=0.033;
    a2=0.155;
    a3=0.135;

% Definition of A matrices
    A = zeros(4,4,5);
    A(:,:,1) = ;                                    %%% STUDENT %%%
    A(:,:,2) = ;                                    %%% STUDENT %%%
    A(:,:,3) = ;                                    %%% STUDENT %%%
    A(:,:,4) = ;                                    %%% STUDENT %%%
    A(:,:,5) = ;                                    %%% STUDENT %%%
```

Template for function: **function T6 = dirkinT6(A)**

```
% A(:,:,1) ... A(:,:,5) - Array of A matrices (input).
% T5 - The matrix represents the pose of the robot top (output).

T6 = ;                                              %%% STUDENT %%%
```

Test results:

$$dirkinT6(dirkinAstaubli([1,2,3,4,5,6])) = \begin{bmatrix} -0.5170 & 0.5432 & -0.6616 & -0.3906 \\ -0.8486 & -0.4266 & 0.3128 & -0.4941 \\ -0.1123 & 0.7231 & 0.6815 & -0.2236 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

Template for function: **function T5 = dirkinT5(A)**

```
% A(:,:,1) ... A(:,:,5) - Array of A matrices (input).
% T5 - The matrix represents the pose of the robot top (output).

T5 = ;                                              %%% STUDENT %%%
```

Test results:

$$dirkinT5(dirkinA([1,2,3,4,5])) = \begin{bmatrix} 0.6673 & -0.7108 & 0.2227 & 0.0501 \\ -0.7356 & -0.5819 & 0.3468 & 0.0780 \\ -0.1169 & -0.3952 & -0.9111 & 0.0142 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

GEOMETRICAL JACOBIAN MATRIX

In previous exercise the kinematic model for two robots was introduced. The result of the model is a matrix $\mathbf{T}$, defining the pose of the last coordinate system in regard to the base coordinate system axis. At this moment we can ask ourselves how the differential change in a joint q_j result in the change of the pose represented by the matrix $\mathbf{T}$ (the change of position $\mathbf{p}$ and orientation $\boldsymbol{\omega}$).

If the change is very short in time, than the change of joint variable can be represented as a joint velocity ($\dot{q}_j$) resulting as a velocity of the last coordinate system (translational velocity $\dot{p}$ or rotational velocity ω), both expressed in regard to the base coordinate system axis. The relation (Equation 2.3) between the velocity of the top of the robot $\dot{\mathbf{p}}$ ($\mathbf{v}$) and joint velocities $\dot{\mathbf{q}}$ represents a **Jacobian matrix**, marked with a letter $\mathbf{J}$.

$$\mathbf{v} = \dot{\mathbf{p}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{2.3}$$

1 The impact of the joint velocities on the robot end movement

The impact of the joint velocity $\dot{q}_j$ on the robot end velocity can be analyzed according to the figure 2.1 showing the robot mechanism with 6 DOF. The rotational velocity of the joint j regarding to the base

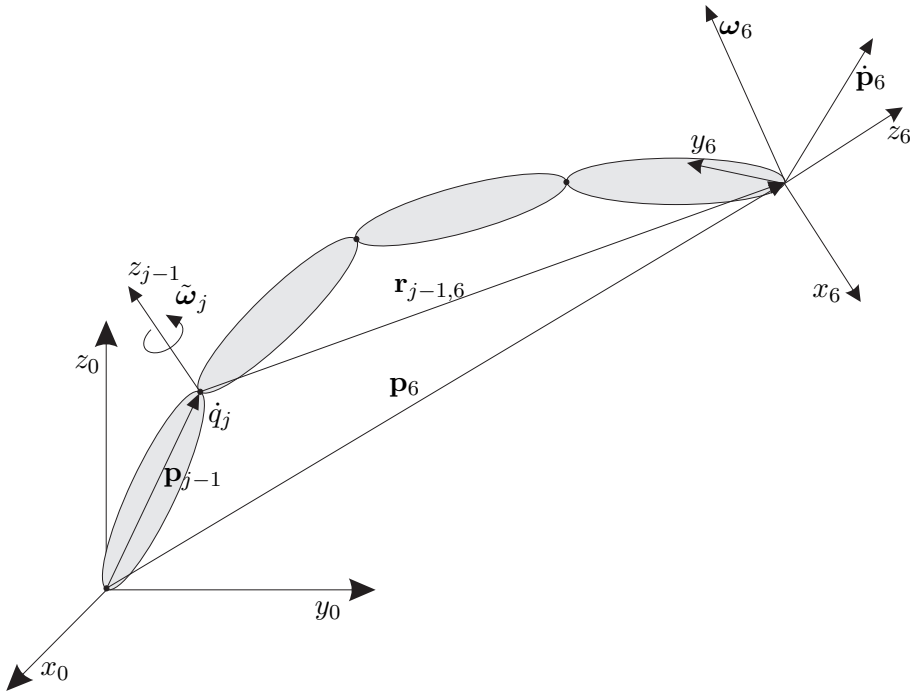

Figure 2.1: The influence of the joint velocity in joint j on the robot top movement. The figure shows conditions where $j = 2$.

coordinate frame can be written as

$$\tilde{\boldsymbol{\omega}}_j = \mathbf{z}_{j-1}\dot{q}_j, \tag{2.4}$$

where vector $\mathbf{z}_{j-1}$ sets the joint axis regarding to the base coordinate frame and $\dot{q}_j$ represents the scalar velocity of the joint rotation. The velocity of the top of the robot as a result of the scalar velocity $\dot{q}_j$ can be determined by vector product

$$\begin{aligned}\dot{\mathbf{p}}_6(\dot{q}_j) = \tilde{\boldsymbol{\omega}}_j \times \mathbf{r}_{j-1,6} = \mathbf{z}_{j-1}\dot{q}_j \times (\mathbf{p}_6 - \mathbf{p}_{j-1}) = \\ = \mathbf{z}_{j-1} \times (\mathbf{p}_6 - \mathbf{p}_{j-1})\dot{q}_j = \mathbf{j}_{P_j}\dot{q}_j,\end{aligned} \tag{2.5}$$

where the vector $\mathbf{p}_6$ connects the base coordinate system with the top of the robot and the vector $\mathbf{p}_{j-1}$ connects base coordinate system with a joint j; $\mathbf{j}_{P_j} = \mathbf{z}_{j-1} \times (\mathbf{p}_6 - \mathbf{p}_{j-1})$.

If the joint is translational then the contribution to the entire robot top velocity is equal to the translational velocity of the joint

$$\dot{\mathbf{p}}_6(\dot{q}_j) = \mathbf{z}_{j-1}\dot{q}_j = \mathbf{j}_{P_j}\dot{q}_j, \tag{2.6}$$

where $\mathbf{j}_{P_j} = \mathbf{z}_{j-1}$.

The rotational velocity of the robot top as a result of the joint velocity q_j is equal to the angular velocity in the joint, expressed to the base coordinate system. Therefore

$$\boldsymbol{\omega}_6(\dot{q}_j) = \tilde{\boldsymbol{\omega}}_j = \mathbf{z}_{j-1}\dot{q}_j = \mathbf{j}_{O_j}\dot{q}_j, \tag{2.7}$$

where $\mathbf{j}_{O_j} = \mathbf{z}_{j-1}$.

The translational joint does not change the robot top orientation. and therefore the contribution to the rotational velocity of the robot top is equal to zero

$$\boldsymbol{\omega}_6(\dot{q}_j) = \mathbf{0} = \mathbf{j}_{O_j}\dot{q}_j, \tag{2.8}$$

where $\mathbf{j}_{O_j} = \mathbf{0}$.

The entire speed of the robot top is specified with the joint movement. That is why it is necessary to sum all joint contributions. Let's look first conditions for the translational velocity. Because the partial velocities $\dot{\mathbf{p}}_6(\dot{q}_j)$ are expressed in the base robot frame, can be summed up

$$\begin{aligned}\dot{\mathbf{p}}_6(\dot{\mathbf{q}}) &= \dot{\mathbf{p}}_6(\dot{q}_1) + \ldots + \dot{\mathbf{p}}_6(\dot{q}_j) + \ldots + \dot{\mathbf{p}}_6(\dot{q}_6) = \\ &= \mathbf{j}_{P_1}\dot{q}_1 + \ldots + \mathbf{j}_{P_j}\dot{q}_j + \ldots + \mathbf{j}_{P_6}\dot{q}_6 = \\ &= \begin{bmatrix}\mathbf{j}_{P_1} & \cdots & \mathbf{j}_{P_6}\end{bmatrix}\begin{bmatrix}\dot{q}_1 \\ \vdots \\ \dot{q}_6\end{bmatrix} = \begin{bmatrix}\mathbf{j}_{P_1} & \cdots & \mathbf{j}_{P_6}\end{bmatrix}\dot{\mathbf{q}}.\end{aligned} \tag{2.9}$$

It can be seen that the translational velocity is determined by a product of the matrix with columns $\mathbf{j}_{P_j}$ and joint velocities vector $\dot{\mathbf{q}}$.

Similar the rotational contributions can be summed up

$$\begin{aligned}\boldsymbol{\omega}_6(\dot{\mathbf{q}}) &= \boldsymbol{\omega}_6(\dot{q}_1) + \ldots + \boldsymbol{\omega}_6(\dot{q}_j) + \ldots + \boldsymbol{\omega}_6(\dot{q}_6) = \\ &= \mathbf{j}_{O_1}\dot{q}_1 + \ldots + \mathbf{j}_{O_j}\dot{q}_j + \ldots + \mathbf{j}_{O_6}\dot{q}_6 = \\ &= \begin{bmatrix}\mathbf{j}_{O_1} & \cdots & \mathbf{j}_{O_6}\end{bmatrix}\begin{bmatrix}\dot{q}_1 \\ \vdots \\ \dot{q}_6\end{bmatrix} = \begin{bmatrix}\mathbf{j}_{O_1} & \cdots & \mathbf{j}_{O_6}\end{bmatrix}\dot{\mathbf{q}}.\end{aligned} \tag{2.10}$$

The rotational velocity of the robot top is determined by a product of the matrix with columns $\mathbf{j}_{O_j}$ and joint velocities vector $\dot{\mathbf{q}}$.

2 Jacobian matrix

The Jacobian matrix establish the relationship between the robot top velocities $\mathbf{v} = \begin{bmatrix}\dot{\mathbf{p}}_6 & \boldsymbol{\omega}_6\end{bmatrix}^T$ and joint velocities $\dot{\mathbf{q}}$

$$\dot{\mathbf{v}} = \mathbf{J}\dot{\mathbf{q}}, \tag{2.11}$$

or

$$\begin{bmatrix}\dot{\mathbf{p}}_6 \\ \boldsymbol{\omega}_6\end{bmatrix} = \begin{bmatrix}\mathbf{J}_P \\ \mathbf{J}_O\end{bmatrix} \dot{\mathbf{q}}. \tag{2.12}$$

Jacobian matrix of the robot can be split into the translational and rotational part

$$\mathbf{J}^{6\times 6} = \begin{bmatrix}\mathbf{J}_P^{3\times 6} \\ \mathbf{J}_O^{3\times 6}\end{bmatrix}. \tag{2.13}$$

The expressions (2.9) and (2.10) are used for calculation of the geometrical Jacobian matrix of the robot. Each column in the Jacobian matrix defines how each of the joint velocities influence on the robot top velocity.

3 Translational Jacobian submatrix

From the expression (2.9) the Jacobian submatrix for calculation of the translational velocities can be extracted

$$\mathbf{J}_P = \begin{bmatrix}\mathbf{j}_{P_1} & \dots & \mathbf{j}_{P_j} & \dots & \mathbf{j}_{P_6}\end{bmatrix}^{3\times 6}. \tag{2.14}$$

The column of the Jacobian matrix $\mathbf{j}_{P_j}$ shows how the velocity of the joint j influence on the translational velocity of the robot top. Therefore we can write

$$\mathbf{j}_{P_j} = \begin{cases}\mathbf{z}_{j-1} & \text{translational joint} \\ \\ \mathbf{z}_{j-1} \times (\mathbf{p}_6 - \mathbf{p}_{j-1}) & \text{rotational joint}\end{cases}, \tag{2.15}$$

which follows from equations (2.5) and (2.6). Vector $\mathbf{z}_{j-1}$ is defined as the third column of the rotational matrix $\mathbf{R}_{j-1}$ (submatrix of the matrix $\mathbf{A}_{j-1}$), therefore

$$\mathbf{z}_{j-1} = \mathbf{R}_1(q_1) \dots \mathbf{R}_{j-1}(q_{j-1})\mathbf{z}_0\,, \tag{2.16}$$

where $\mathbf{z}_0 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$. Vector $\mathbf{p}_6$ defines the robot top position in robot base frame and is defined by first three elements of the forth column of the matrix

$$\mathbf{T}_6 = \mathbf{A}_1(q_1) \dots \mathbf{A}_6(q_6). \tag{2.17}$$

Vector $\mathbf{p}_{j-1}$ is defined by the first three elements of the forth column of the transformation matrix $\mathbf{T}_{j-1} = \mathbf{A}_1(q_1) \dots \mathbf{A}_{j-1}(q_{j-1})$; $\mathbf{p}_0 = \mathbf{0}^{3\times 1}$.

4 Rotational Jacobian submatrix

$\mathbf{J}_O$ is the Jacobian submatrix for rotational robot top velocities in the base frame calculation. The expression (2.10) defines

$$\mathbf{J}_O = \begin{bmatrix} \mathbf{j}_{O_1} & \dots & \mathbf{j}_{O_j} & \dots & \mathbf{j}_{O_6} \end{bmatrix}^{3\times 6}, \tag{2.18}$$

where

$$\mathbf{j}_{O_j} = \begin{cases} \mathbf{0} & \text{translational joint} \\ \\ \mathbf{z}_{j-1} & \text{rotational joint.} \end{cases} \tag{2.19}$$

It is included that the translational joint does not change the robot top orientation and is rotational velocity of the robot top equal to zero. The rotational joint causes the rotation of the robot top as is stated by the equation (2.7). Vector $\mathbf{z}_{j-1}$ is defined in the equation (2.16).

5 The student tasks

- In Matlab fill the function template `function J = jacobi0(q)` for calculation of the geometrical Jacobian matrix for the robot KUKA youBot. Use the functions from the first exercise. The input parameter is a vector of joint variables **q** and the output is a geometrical Jacobian matrix of the robot **J**. Your task is to properly fill in the functions in a template were it is marked with `%%% STUDENT %%%`.
- In simulation environment Matlab Simulink build a control scheme where a geometrical Jacobian matrix is used. The scheme and details about it is written at the end of the chapter.

6 Necessary knowledge and goal of the exercise

The goal of the exercise is to get familiar with the use of geometrical Jacobian matrix in robotics. Filled Matlab function will be used in Matlab Simulink environment to test the control scheme on the real robot.

The necessary knowledge to accomplish a goal is writing Matlab code and basics of using Matlab Simulink.

7 Matlab template

Template for function: **function J = jacobi0(q)**

```
% q - joint variables (input).
% J - geometrical Jacobian matrix (output).

% Calculation of the transformation matrices A.
    A = ;                                              %%% STUDENT %%%

% Calculation of the transformation matrix T.
    T5 = ;                                             %%% STUDENT %%%

% Initialization of an empty Jacobian matrix.
    Jp = zeros(3,5);    % Position submatrix.
    Jo = zeros(3,5);    % Orientation submatrix.

% Initialization of the variables.
    z_0 = [0,0,1]';     % Vector in the z_0 axis direction.
    p_0 = [0,0,0]';     % Position vector of the first joint towards the base frame
    Tn = eye(4);        % The initial value of the matrix Tn; n = 1,2...

% 1. COLUMN OF THE JACOBIAN SUBMATRIX
    % Calculation of the 1st column of the positional Jacobian submatrix.
    Jp(:,1) =                                            %%% STUDENT %%%
    % Calculation of the 1st column of the rotational Jacobian submatrix.
    Jo(:,1) =                                            %%% STUDENT %%%

% 2. COLUMN OF THE JACOBIAN SUBMATRIX
    % Calculation of the Tn matrix that contains Zj-1 and Pj-1 information.
    Tn =                                                 %%% STUDENT %%%
    % Calculation of the 2nd column of the positional Jacobian submatrix.
    Jp(:,2) =                                            %%% STUDENT %%%
    % Calculation of the 2nd column of the rotational Jacobian submatrix.
    Jo(:,2) =                                            %%% STUDENT %%%

% 3. COLUMN OF THE JACOBIAN SUBMATRIX
    % Calculation of the Tn matrix that contains Zj-1 and Pj-1 information.
    Tn =                                                 %%% STUDENT %%%
    % Calculation of the 3rd column of the positional Jacobian submatrix.
    Jp(:,3) =                                                %%% STUDENT %%%
    % Calculation of the 3rd column of the rotational Jacobian submatrix.
    Jo(:,3) =                                            %%% STUDENT %%%

% 4. COLUMN OF THE JACOBIAN SUBMATRIX
    % Calculation of the Tn matrix that contains Zj-1 and Pj-1 information.
    Tn =                                                 %%% STUDENT %%%
    % Calculation of the 4th column of the positional Jacobian submatrix.
    Jp(:,4) =                                            %%% STUDENT %%%
    % Calculation of the 4th column of the rotational Jacobian submatrix.
    Jo(:,4) =                                            %%% STUDENT %%%

% 5. COLUMN OF THE JACOBIAN SUBMATRIX
    % Calculation of the Tn matrix that contains Zj-1 and Pj-1 information.
    Tn =                                                 %%% STUDENT %%%
    % Calculation of the 5th column of the positional Jacobian submatrix.
    Jp(:,5) =                                            %%% STUDENT %%%
    % Calculation of the 5th column of the rotational Jacobian submatrix.
    Jo(:,5) =                                            %%% STUDENT %%%

% The whole Jacobian matrix is combined from both submatrices.
    J = [Jp; Jo];
```

Test result:

$$jacobi0([1,2,3,4,5]) = \begin{bmatrix} -0.0780 & -0.0718 & -0.0369 & -0.0576 & 0 \\ 0.0501 & -0.1118 & -0.0575 & -0.0897 & 0 \\ 0 & -0.0597 & 0.0812 & -0.0482 & 0 \\ 0 & -0.8415 & -0.8415 & -0.8415 & 0.2227 \\ 0 & 0.5403 & 0.5403 & 0.5403 & 0.3468 \\ 1 & 0 & 0 & 0 & -0.9111 \end{bmatrix}$$

8 A simulation scheme in Matlab Simulink

In simulation environment Matlab Simulink build a simulation scheme from Figure 2.2. A block *Constant* represents the reference velocity of the robot end $\dot{\mathbf{x}} = \begin{bmatrix} -0.01 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$. The result of the block is a geometrical Jacobian matrix $\mathbf{J}(\mathbf{q})$ *MATLAB Function - Jacobi0* that depends on the current value of joint variables ($\mathbf{q}$). Inside this block it is necessary to copy the whole *jacobi0* function and all other dependant functions (*hdh, dirkinA, dirkinT5*). Further block *Pseudoinverse* invert geometrical Jacobian matrix that is further multiplied with the reference velocity of the robot end ($\dot{\mathbf{v}}$); the units are m/s. The simulation scheme is recalculated with a sample time of dT = 5 ms. The whole simulation scheme can be represented by the equation 2.20.

$$\dot{\mathbf{q}} = \mathbf{J}^{-1}(\mathbf{q})\dot{\mathbf{v}} \tag{2.20}$$

To open the Simulink template use the Matlab command line and run program *initVR.m*!

To preserve the template, please save your changes in a new file!

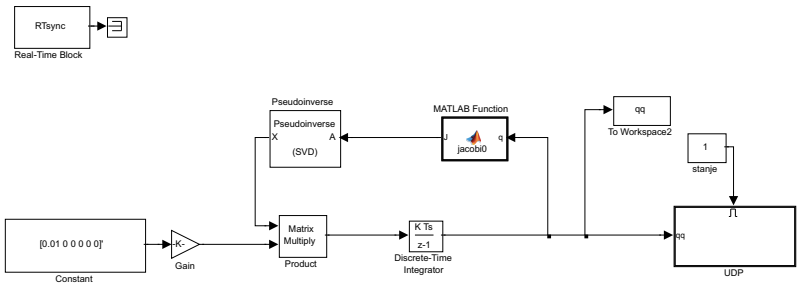

Figure 2.2: Simulacijska shema v Simulinku.

Where Simulink objects can be found:
Descrete-Time Integrator ⇒ *Simulink* → *Discrete*
Gain value: 1.0; Initial condition: q0; Sample time: dT
Matrix Multiply ⇒ *Simulink* → *Math Operations*
Embadded MATLAB Function ⇒ *Simulink* → *User-Defined Func.*
Pseudoinverse ⇒ *Signal Processing Blockset* → *Math Functions* → *Matrices and Linear Algebra* → *Matrix Inverse*

THE RELATION BETWEEN GEOMETRICAL AND ANALYTICAL JACOBIAN MATRIX AND INVERSE KINEMATICS WITH THE USE OF ANALYTICAL JACOBIAN MATRIX

Rotation matrices provide redundant description of the orientation. Orientation of the object in space is defined by only 3 parameters, while the rotational matrices contains 9.

The minimal use of three parameters can be performed with three angles $\boldsymbol{\phi} = \begin{bmatrix} \varphi & \vartheta & \psi \end{bmatrix}$.

Assume the rotational matrix that describe rotation around one axis as a function of one angle. General rotation around the three axes we can get with a combination of three consecutive rotations, where we

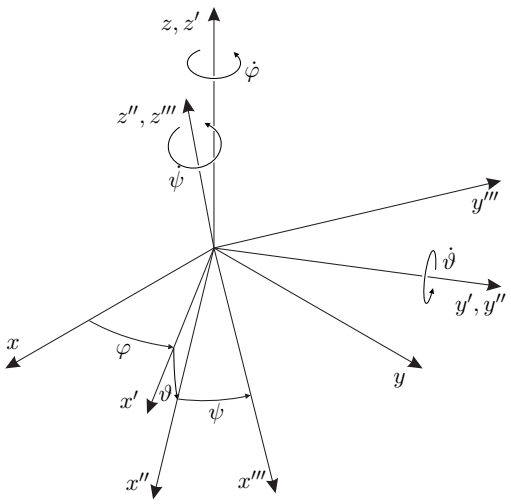

Figure 3.1: Presentation of the orientation of the ZYZ Euler angles

must be careful that two successive rotations are not performed around two parallel axis. To describe the orientation of an object in free space 12 different combinations of three basic rotations around individual axes can be used. As an example the combination ZYZ means, that first the rotation around z axis is performed, then around y axis of the new frame and at the end another rotation around z axis of the newest frame. The situation is shown in figure 3.1. Each sequence of rotations represents a triplet of the Euler angles.

1 *ZYX* Euler angles

The rotation ZYX is defined as a sequence of the following basic rotations:

- Rotation of a basic coordinate frame for an angle φ around z axis. This rotation is described by rotational matrix $\mathbf{R}_z(\varphi)$.
- Rotation of the current coordinate frame for an angle ϑ around y' axis. This rotation is described by rotational matrix $\mathbf{R}_{y'}(\vartheta)$.
- Rotation of the current coordinate frame for an angle ψ around x'' axis. This rotation is described by rotational matrix $\mathbf{R}_{x''}(\psi)$.

The overall rotation is obtained by a sequence of rotations around the current coordinate system, therefore the postmultiplication is used in the multiplication of the matrices (Equation 3.21).

$$\begin{aligned}\mathbf{R}(\boldsymbol{\phi}) &= \mathbf{R}_z(\varphi)\mathbf{R}_{y'}(\vartheta)\mathbf{R}_{x''}(\psi) \\ &= \begin{bmatrix} c_\varphi c_\vartheta & c_\varphi s_\vartheta s_\psi - s_\varphi c_\psi & c_\varphi s_\vartheta c_\psi + s_\varphi s_\psi \\ s_\varphi c_\vartheta & s_\varphi s_\vartheta s_\psi + c_\varphi c_\psi & s_\varphi s_\vartheta c_\psi - c_\varphi s_\psi \\ -s_\vartheta & c_\vartheta s_\psi & c_\vartheta c_\psi \end{bmatrix} \\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}\end{aligned} \tag{3.21}$$

If the elements of the $\mathbf{R}(\boldsymbol{\phi})$ matrix are known, then it is possible to calculate the Euler angles. With a comparison of the elements of the matrix in equation (3.21) it is possible to calculate three Euler angles. Assuming that $r_{11} \neq 0$ and $r_{21} \neq 0$ following line can be written

$$\varphi = atan2(r_{21}, r_{11}), \tag{3.22}$$

where $atan2$ is a four-quadrant arcus tangent function. With squaring and aggregation of the elements r_{32} and r_{33} and including r_{31} following

can be derived

$$\vartheta = atan2(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}). \tag{3.23}$$

Choosing the positive sign of the term $\sqrt{r_{32}^2 + r_{33}^2}$ limits the value of an angle ϑ to $(0, \pi)$. The angle ψ can be defined on the basic of the following equation

$$\psi = atan2(r_{32}, r_{33}). \tag{3.24}$$

In summary, the triplet ZYX of the Euler angles can be determined on the basis of the following equations

$$\begin{aligned} \varphi &= atan2(r_{21}, r_{11}) \\ \vartheta &= atan2(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}) \\ \psi &= atan2(r_{32}, r_{33}). \end{aligned} \tag{3.25}$$

2 Transformation matrix

Recall that the location of the object in space with a minimum number of parameters can be written with six variables, where the first three define the position and form a vector $\mathbf{p}$, while the other three determine the orientation and form a vector $\boldsymbol{\phi}$

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \boldsymbol{\phi} \end{bmatrix}. \tag{3.26}$$

The geometrical Jacobian matrix provides the geometrical relationship between velocities in the joints and the robot top velocities relative to the base coordinate system. When describing the location of the top of the robot with a minimal number of parameters $\mathbf{x}$, there is an option of calculating the Jacobian matrix elements with a differentiating the elements of the vector $\mathbf{x}$ by the joint variables

$$\dot{\mathbf{x}} = \frac{\partial \mathbf{x}}{\partial \mathbf{q}} \dot{\mathbf{q}} = \mathbf{J}_A(\mathbf{q}) \dot{\mathbf{q}}, \tag{3.27}$$

therefore we can calculate the translational velocity of the top of the robot as

$$\dot{\mathbf{p}} = \frac{\partial \mathbf{p}}{\partial \mathbf{q}} \dot{\mathbf{q}} = \mathbf{J}_p(\mathbf{q}) \dot{\mathbf{q}} \tag{3.28}$$

and rotational velocity of the robot top as

$$\dot{\boldsymbol{\phi}} = \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{q}} \dot{\mathbf{q}} = \mathbf{J}_\phi(\mathbf{q}) \dot{\mathbf{q}}, \tag{3.29}$$

where vector $\boldsymbol{\phi}$ defines the triplet of the Euler angles. When translation and rotation are taken into account we can write

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{p}} \\ \dot{\boldsymbol{\phi}} \end{bmatrix} = \begin{bmatrix} \mathbf{J}_p(\mathbf{q}) \\ \mathbf{J}_\phi(\mathbf{q}) \end{bmatrix} \dot{\mathbf{q}} = \mathbf{J}_A(\mathbf{q})\dot{\mathbf{q}}. \tag{3.30}$$

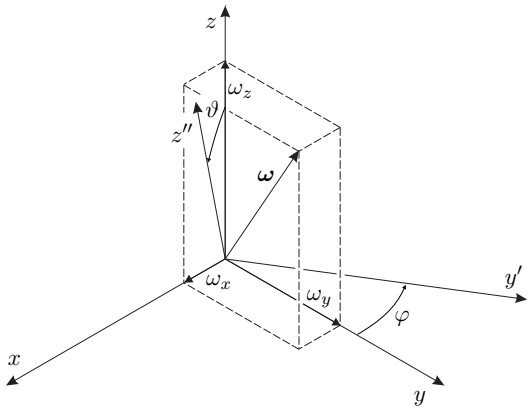

Figure 3.2: Decomposition of velocity vector into rotational velocities in the reference coordinate system.

Follows the search of transformation matrix that connects geometrical and analytical Jacobian matrix defined on the basis of ZYX Euler angles. For this purpose it is necessary to find the relationship between rotational velocities $\boldsymbol{\omega} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$ and ZYX Euler angles velocities $\dot{\boldsymbol{\phi}}$ (Figure 3.2). Figure 3.3 shows an example of ZYZ Euler angles, where the velocities $\dot{\varphi}$, $\dot{\vartheta}$ in $\dot{\psi}$ are presented regarding to the exes of the current coordinate frame. This figure also shows the calculation process of the contribution of each individual velocity to the velocities defined in the base coordinate frame. For the ZYX Euler angles:

- as a result of velocity $\dot{\varphi}$: $\begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T = \dot{\varphi} \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$; we considered that at the beginning the z axis of the current coordinate frame is aligned with the z axis of the base coordinate frame,

- as a result of velocity $\dot{\vartheta}$: $\begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T = \dot{\vartheta} \begin{bmatrix} -s_\varphi & c_\varphi & 0 \end{bmatrix}^T$; we considered that a rotation φ around z axis of the base frame is present between the base and current frame,

- as a result of velocity $\dot{\psi}$: $\begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T = \dot{\psi} \begin{bmatrix} c_\varphi c_\vartheta & s_\varphi c_\vartheta & -s_\vartheta \end{bmatrix}^T$; we considered that a rotation φ around z axis of the base frame is

present between the base and current frame and a rotation ϑ around y axis of the current frame.

The procedure of velocitiy calculation regarding to the reference frame axes:

$$\begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}0\\ 0\\ \dot{\varphi}\end{bmatrix} + \begin{bmatrix}c_\varphi & -s_\varphi & 0\\ s_\varphi & c_\varphi & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}0\\ \dot{\vartheta}\\ 0\end{bmatrix} + \begin{bmatrix}c_\varphi & -s_\varphi & 0\\ s_\varphi & c_\varphi & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}c_\vartheta & 0 & s_\vartheta\\ 0 & 1 & 0\\ -s_\vartheta & 0 & c_\vartheta\end{bmatrix}\begin{bmatrix}\dot{\psi}\\ 0\\ 0\end{bmatrix}, \quad (3.31)$$

The effects of all three Euler angles velocities on the angular velocities around base coordinate frame is obtained if the above terms are put in a matrix form

$$\begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\end{bmatrix} = \begin{bmatrix}0 & -s_\varphi & c_\varphi c_\vartheta\\ 0 & c_\varphi & s_\varphi c_\vartheta\\ 1 & 0 & -s_\vartheta\end{bmatrix}\begin{bmatrix}\dot{\varphi}\\ \dot{\vartheta}\\ \dot{\psi}\end{bmatrix}, \quad (3.32)$$

that can be written as

$$\boldsymbol{\omega} = \mathbf{T}(\boldsymbol{\phi})\dot{\boldsymbol{\phi}}. \quad (3.33)$$

It remains a determination of the transformation matrix for transformation between the geometrical and analytical Jacobian matrix. We will take into account that the translational velocities of the robot top do not change if different formats for robot top orientation are used. This means that the submatrix of the Jacobian matrix, which refers to the translational velocities, is equal to $\mathbf{J}_p(\mathbf{q})$, no matter which analytical or geometrical Jacobian matrix is used. The transformation of the rotational velocities is defined by a matrix $\mathbf{T}(\boldsymbol{\phi})$ from which we can derive transformation between $\mathbf{J}_o(\mathbf{q})$ and $\mathbf{J}_\phi(\mathbf{q})$.

As a start we can take next three relations

$$\begin{aligned}\boldsymbol{\omega} &= \mathbf{J}_o(\mathbf{q})\dot{\mathbf{q}},\\ \dot{\boldsymbol{\phi}} &= \mathbf{J}_\phi(\mathbf{q})\dot{\mathbf{q}},\\ \boldsymbol{\omega} &= \mathbf{T}(\boldsymbol{\phi})\dot{\boldsymbol{\phi}}.\end{aligned} \quad (3.34)$$

From these we can write

$$\mathbf{T}(\boldsymbol{\phi})\mathbf{J}_\phi(\mathbf{q})\dot{\mathbf{q}} = \mathbf{J}_o(\mathbf{q})\dot{\mathbf{q}}, \quad (3.35)$$

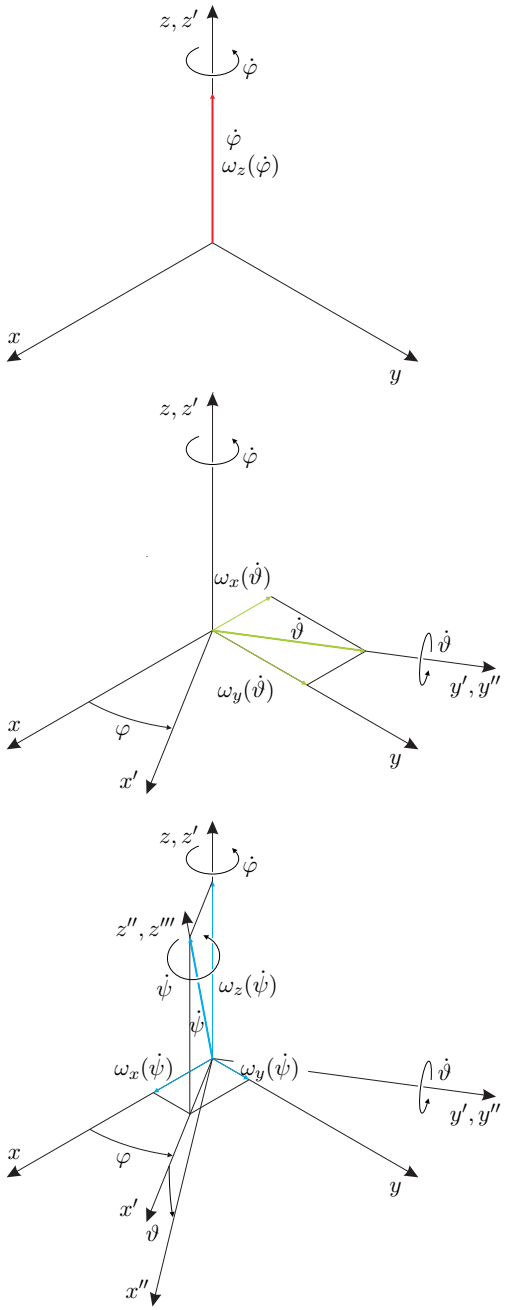

Figure 3.3: Decomposition of ZYZ Euler velocities in components of reference coordinate system.

and this means

$$\mathbf{T}(\boldsymbol{\phi})\mathbf{J}_{\phi}(\mathbf{q}) = \mathbf{J}_{o}(\mathbf{q}). \tag{3.36}$$

The whole transformation can be written as

$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} \mathbf{J}_{p}(\mathbf{q}) \\ \mathbf{J}_{o}(\mathbf{q}) \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}(\boldsymbol{\phi}) \end{bmatrix} \begin{bmatrix} \mathbf{J}_{p}(\mathbf{q}) \\ \mathbf{J}_{\phi}(\mathbf{q}) \end{bmatrix} = \mathbf{T}_{A}(\boldsymbol{\phi})\mathbf{J}_{A}(\mathbf{q}), \tag{3.37}$$

where $\mathbf{I}$ is the unit matrix with dimensions 3×3 and $\mathbf{0}$ is a matrix of zeros with dimensions 3×3. The opposite also applies

$$\mathbf{J}_{A}(\mathbf{q}) = \mathbf{T}_{A}^{-1}(\boldsymbol{\phi})\mathbf{J}(\mathbf{q}). \tag{3.38}$$

3 Calculation of the inverse kinematics

Analytical Jacobian matrix of the robot $\mathbf{J}_{A}(\mathbf{q})$ can be used for calculation of the robot top velocities $\dot{\mathbf{x}}$ (the coordinates of the robot top are presented with minimal number of parameters – the position and orientation described with Euler angles) if the joint velocities $\dot{\mathbf{q}}$ are known

$$\dot{\mathbf{x}} = \mathbf{J}_{A}(\mathbf{q})\dot{\mathbf{q}}. \tag{3.39}$$

Also the reversed relation can be used, if the Jacobian matrix is not singular

$$\dot{\mathbf{q}} = \mathbf{J}_{A}^{-1}(\mathbf{q})\dot{\mathbf{x}}. \tag{3.40}$$

For calculation of the inverse kinematics of the robot based on the inverse analytical Jacobian matrix the most simple Euler integration method will be used. The estimated value of the joint angles $\hat{\mathbf{q}}(t_{k+1})$ at the time t_{k+1} can be determined on the basis of the estimated value of joint angles $\hat{\mathbf{q}}(t_k)$ at the time t_k and the robot velocity $\dot{\mathbf{q}}(t_k)$

$$\hat{\mathbf{q}}(t_{k+1}) = \hat{\mathbf{q}}(t_k) + \dot{\mathbf{q}}(t_k)\Delta t, \tag{3.41}$$

where a Δt is a calculation step of the robot controller. If we take into account the equation (3.40) the upper equation can be rewritten as

$$\hat{\mathbf{q}}(t_{k+1}) = \hat{\mathbf{q}}(t_k) + \mathbf{J}_{A}^{-1}(\hat{\mathbf{q}}(t_k))\dot{\mathbf{x}}(t_k)\Delta t. \tag{3.42}$$

On the basis of equation (3.42) it could be possible to calculate the inverse kinematics of the robot where the initial position and the robot top velocities are known. Since the numerical integration brings drift is

this result useless. Solution to the problem of a drift can be found in the introduction of the feedback loop, which eliminates drift by comparing the true robot top pose $\mathbf{x}$ with the estimated robot top pose $\hat{\mathbf{x}}$, calculated on the angles defined in equation (3.42). First we define the error of the calculated robot top pose

$$\tilde{\mathbf{x}} = \mathbf{x} - \hat{\mathbf{x}} = \mathbf{x} - k(\hat{\mathbf{q}}), \tag{3.43}$$

where $k(\hat{\mathbf{q}})$ indicates the robot direct geometrical model. The equation (3.43) differentiate over time and the result is

$$\dot{\tilde{\mathbf{x}}} = \dot{\mathbf{x}} - \dot{\hat{\mathbf{x}}} = \dot{\mathbf{x}} - \mathbf{J}_A(\hat{\mathbf{q}})\dot{\hat{\mathbf{q}}}. \tag{3.44}$$

The error $\tilde{\mathbf{x}}$ needs to be included into the estimation of the joint velocities $\dot{\hat{\mathbf{q}}}$ in a way that error converges to zero. One of the options is following control algorithm

$$\dot{\hat{\mathbf{q}}} = \mathbf{J}_A^{-1}(\hat{\mathbf{q}})(\dot{\mathbf{x}} + \mathbf{K}\tilde{\mathbf{x}}) = \mathbf{J}_A^{-1}(\hat{\mathbf{q}})\dot{\mathbf{x}} + \mathbf{J}_A^{-1}(\hat{\mathbf{q}})\mathbf{K}\tilde{\mathbf{x}}, \tag{3.45}$$

where $\mathbf{K}$ is a matrix of the proportional gains of the closed-loop system. With equation (3.45) integration into the equation (3.44) we get

$$\dot{\tilde{\mathbf{x}}} + \mathbf{K}\tilde{\mathbf{x}} = \mathbf{0}. \tag{3.46}$$

If the gain matrix $\mathbf{K}$ is positive-definite (diagonal) then the system (3.46) is asymptotically stable. Implementation of calculating inverse kinematics of the robot based on inverse analytical Jacobian matrix is shown in Figure 3.4.

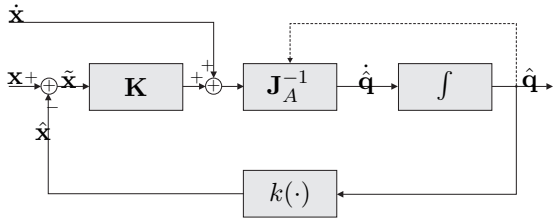

Figure 3.4: The inverse kinematics with an use of the inverse analytical Jacobian matrix. $\mathbf{K}$ represents the diagonal gain matrix, $\mathbf{J}_A$ analytical Jacobian matrix, $\int$ is an integrator, $k(\cdot)$ a direct kinematics, $\mathbf{x}$ is the robot top pose that needs to be transformed into the joint coordinates, $\dot{\mathbf{x}}$ are the reference speed of the top of the robot and $\hat{\mathbf{x}}$ is estimated pose of the robot based on the calculated joint angles. All robot poses are expressed with minimal number of parameters (position and orientation with Euler angles).

4 The student tasks

- In Matlab fill in the template of the functions `function X = q2ZYXeul(q)` and `function invJ = ijacZYXeul(q)`.

  `function X = q2ZYXeul(q)` takes the joint angles as an input parameter **q** and calculates the pose of the robot top as a position XYZ and orientation represented with ZYX Euler angles. The position and Euler angles makes the vector of the robot top pose. Use the function for robot direct kinematics calculation.

  `function invJ = ijacZYXeul(q)` takes the joint angles as an input parameter **q** and calculates the inverse analytical Jacobian matrix. Use the function for calculation of the geometrical Jacobian matrix and the poses of the robot represented with position and Euler angles.

  Your task is to properly fill the function lines marked with `%%% STUDENT %%%`.

- Regarding to the Figure 3.4 build a Simulink scheme for inverse kinematics calculation based on the inverse Jacobian matrix as is shown on Figure 3.5. Test system performance with an use of the test data, which enable movement of the robot platform along the curve in XZ plane, as shown in Figure 3.6. The test data is stored in a file named `xTrajec.mat`.

5 Necessary knowledge and goal of the exercise

The additional knowledge when working on Matlab and Matlab Simulink is not necessary. The goal of the exercise is to build simulation scheme of inverse kinematics with an use of analytical Jacobian matrix.

6 Matlab templates

Template for function: **x = q2ZYXeul(q)**

```
% q - Vector of joint variables for 6 joints (input)
% x - The pose of the robot top as XYZ and ZYZ Euler angles (output)

% Calculation of the transformation matrix T
    T5 = ;                                              %%% STUDENT %%%

% Decomposition of the rotational matrix elements
    r21 = ;                                             %%% STUDENT %%%
    r31 = ;                                             %%% STUDENT %%%
    r32 = ;                                             %%% STUDENT %%%
    r33 = ;                                             %%% STUDENT %%%
    r11 = ;                                             %%% STUDENT %%%

% Calculation of Euler angles
    fi = ;                                              %%% STUDENT %%%
    theta = ;                                           %%% STUDENT %%%
    psi = ;                                             %%% STUDENT %%%

% The robot top orientation represented with a ZYZ Euler angles vector
    ea = [fi, theta, psi];

% The pose of the robot top as XYZ and ZYZ Euler angles
    X = ;                                               %%% STUDENT %%%
```

Test results:
$q2ZYXeul([1,2,3,4,5]) = \begin{bmatrix}0.0501 & 0.0780 & 0.0142 & -0.8341 & 0.1172 & -2.7323\end{bmatrix}^T$

Template for function: **function invJ = ijacZYXeul(q)**

```
% q - Vector of joint variables for 6 joints (input)
% invJ - Inverse analytical Jacobian matrix (output).

% Calculation of the geometrical Jacobian matrix
    Jg = ;                                              %%% STUDENT %%%

% Calculation of the pose of the robot top as XYZ and ZYZ Euler angles
    X = ;                                               %%% STUDENT %%%

% From vector X the right Euler angles components are extracted
    fi      = ;                                         %%% STUDENT %%%
    theta   = ;                                         %%% STUDENT %%%
    psi     = ;                                         %%% STUDENT %%%

% Calculation of the transformation matrix for conversion between the geometrical and
% analytical Jacobian matrix
    Tr = ;                                              %%% STUDENT %%%

% Calculation of the transformation matrix for conversion between the geometrical and
% analytical Jacobian matrix
% Inverse Jacobian matrix [6x5]-->psevdo inverz: dq=[J'*J]^-1*J'
    invJ = ;                                            %%% STUDENT %%%
```

Testni rezultat:

$$ijacZYXeul([1,...]) = \begin{bmatrix} -0.4473 & 0.2872 & 0 & 0.9507 & 2.0295 & -0.6544 \\ -23.6551 & -36.9097 & 12.9682 & 0.0035 & 1.0608 & 3.8794 \\ -2.1018 & -3.2731 & 8.8754 & 0 & -0.0035 & -0.0127 \\ 25.7491 & 40.1877 & -21.8436 & -0.0043 & -1.3194 & -4.8250 \\ -0.4029 & 0.2587 & 0 & -0.0444 & 2.2481 & -0.5966 \end{bmatrix}$$

For identity (I) use function eye(N,M) nad for zero matrix use zeros(N,M)!

7 Simulation scheme in Matlab Simulink environment

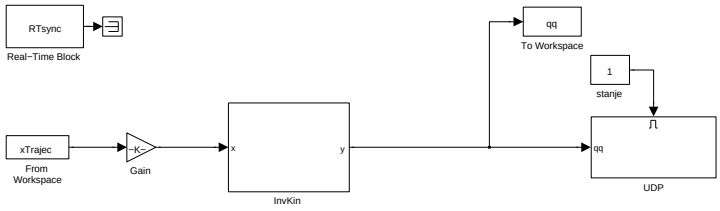

Figure 3.5: Simulation scheme in the Simulink.

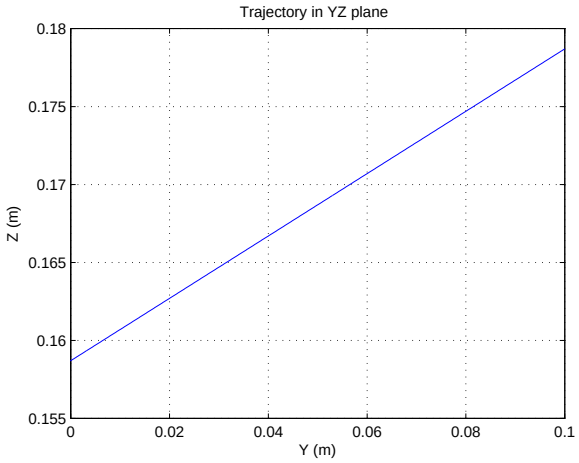

Figure 3.6: Test trajectory calculated to test the robot inverse kinematics.

To open the Simulink template use the Matlab command line and run program *initVR_eul.m*!

Where Simulink objects can be found:
Descrete Derivative ⇒ *Simulink* → *Discrete*
Gain value: 1.0; Inital condition ...: 0.0
Saturation ⇒ *Simulink* → *Discontinuities*
Upper limit: 0.5; Lower limit: -0.5
Gain ⇒ *Simulink* → *Math Operations*
Gain: fbGain
MATLAB Function ⇒ *Simulink* → *User-Defined Func.*
Matrix Multiply ⇒ *Simulink* → *Math Operations*
Descrete-Time Integrator ⇒ *Simulink* → *Discrete*
Gain value: 1.0; Initial condition: q0; Sample time: dT

FORCE AND TORQUE TRANSFORMATION

The robot mechanism can be influenced by outside forces that act on the robot mechanism. Usually the industrial robots are not in direct contact with the environment but in research area the robots are. This concept is believed to be transferred into the industrial area in several years. To be able to sense the outside forces it is necessary to equip the robot end with force and torque sensor. This sensor can measure the forces and torques acting on the robot end and can be used in special control schemes to move the robot mechanism. For this kind of approach it is necessary to be familiar with transformation of forces and torques between coordinate systems.

For the transformation of forces and torques we use the concept of kineto-static duality, which is easily derived in the case of velocity.

1 Velocity transformation

Figure 4.1 shows rigid body with two coordinate frames attached ($O_1 - x_1y_1z_1$ and $O_2 - x_2y_2z_2$). The outer coordinate frame is marked as $O_0 - x_0y_0z_0$. The rigid body is moving regarding to the reference or base coordinate frame. The connection between translational and rotational velocities of both coordinate frames is defined by

$$\begin{aligned} \dot{\mathbf{p}}_2^0 &= \dot{\mathbf{p}}_1^0 + \boldsymbol{\omega}_1^0 \times \mathbf{r}_{12}^0 \\ \boldsymbol{\omega}_2^0 &= \boldsymbol{\omega}_1^0, \end{aligned} \tag{4.47}$$

where all vectors are expressed in the reference frame O_0.

The vector product that is present in the equation (4.47) can be written also as a product of oblique-symmetric matrix $\mathbf{S}(\mathbf{r}_{12}^0)$ and a vector $\boldsymbol{\omega}_1^0$, torej

$$\boldsymbol{\omega}_1^0 \times \mathbf{r}_{12}^0 = -\mathbf{r}_{12}^0 \times \boldsymbol{\omega}_1^0 = -\mathbf{S}(\mathbf{r}_{12}^0)\boldsymbol{\omega}_1^0, \tag{4.48}$$

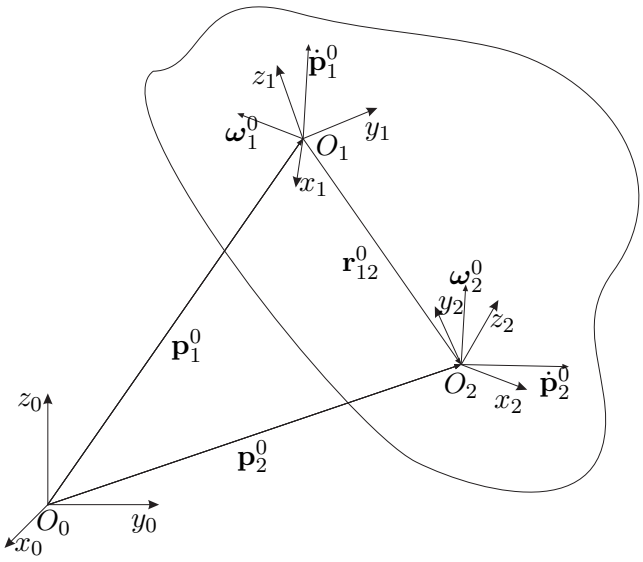

Figure 4.1: Introduction of linear and angular velocity in different coordinate systems on the same rigid body.

where oblique-symmetric matrix is defined as

$$\mathbf{S}(\mathbf{r}_{12}^0) = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}. \tag{4.49}$$

This matrix includes the components of the vector $\mathbf{r}$ that connects frames O_1 and O_2. For oblique-symmetric matrix following specifications can be written

$$\mathbf{S}(\mathbf{Rr}) = \mathbf{RS}(\mathbf{r})\mathbf{R}^T \tag{4.50}$$

and

$$\mathbf{S} + \mathbf{S}^T = \mathbf{0}. \tag{4.51}$$

With an use of oblique-symmetric matrix the equations (4.47) are defined as

$$\begin{aligned} \dot{\mathbf{p}}_2^0 &= \dot{\mathbf{p}}_1^0 - \mathbf{S}(\mathbf{r}_{12}^0)\boldsymbol{\omega}_1^0 \\ \boldsymbol{\omega}_2^0 &= \boldsymbol{\omega}_1^0, \end{aligned} \tag{4.52}$$

or in a matrix form

$$\begin{bmatrix} \dot{\mathbf{p}}_2^0 \\ \boldsymbol{\omega}_2^0 \end{bmatrix} = \begin{bmatrix} \mathbf{I} & -\mathbf{S}(\mathbf{r}_{12}^0) \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{p}}_1^0 \\ \boldsymbol{\omega}_1^0 \end{bmatrix}, \tag{4.53}$$

where $\mathbf{I}$ is an unit matrix with the proper dimensions.

All vectors in equation (4.53) are defined regarding to the reference coordinate frame $O_0 - x_0y_0z_0$. If the vectors are defined regarding to the local coordinate frame, then the right transformations must be taken into account. The vector $\mathbf{r}_{12}^1$ that is defined in coordinate frame $O_1 - x_1y_1z_1$ can be also expressed in base or reference frame $O_0 - x_0y_0z_0$ as

$$\mathbf{r}_{12}^0 = \mathbf{R}_1^0\mathbf{r}_{12}^1, \tag{4.54}$$

where the matrix $\mathbf{R}_1^0$ defines the orientation of the coordinate frame $O_1 - x_1y_1z_1$ regarding to $O_0 - x_0y_0z_0$. Similarly, we can transform the vectors $\dot{\mathbf{p}}_1^1$ and $\boldsymbol{\omega}_1^1$ that are both expressed in coordinate frame $O_1 - x_1y_1z_1$

$$\begin{aligned} \dot{\mathbf{p}}_1^0 &= \mathbf{R}_1^0\dot{\mathbf{p}}_1^1 \\ \boldsymbol{\omega}_1^0 &= \mathbf{R}_1^0\boldsymbol{\omega}_1^1. \end{aligned} \tag{4.55}$$

The vectors $\dot{\mathbf{p}}_2^2$ and $\boldsymbol{\omega}_2^2$ are both expressed in coordinate frame $O_2 - x_2y_2z_2$ and that is why the transformation into the reference frame is as follows

$$\begin{aligned} \dot{\mathbf{p}}_2^0 &= \mathbf{R}_2^0\dot{\mathbf{p}}_2^2 = \mathbf{R}_1^0\mathbf{R}_2^1\dot{\mathbf{p}}_2^2 \\ \boldsymbol{\omega}_2^0 &= \mathbf{R}_2^0\boldsymbol{\omega}_2^2 = \mathbf{R}_1^0\mathbf{R}_2^1\boldsymbol{\omega}_2^2, \end{aligned} \tag{4.56}$$

where the matrix $\mathbf{R}_2^0$ defines orientation of the coordinate frame $O_2 - x_2y_2z_2$ regarding to $O_0 - x_0y_0z_0$ and the matrix $\mathbf{R}_2^1$ defines the orientation of the coordinate frame $O_2 - x_2y_2z_2$ regarding to $O_1 - x_1y_1z_1$.

Considering the equations (4.54), (4.55) and (4.56) and also the characteristic of the oblique-symmetric matrix (4.50) in the equations (4.52) following can be expressed

$$\begin{aligned} \mathbf{R}_1^0\mathbf{R}_2^1\dot{\mathbf{p}}_2^2 &= \mathbf{R}_1^0\dot{\mathbf{p}}_1^1 - \mathbf{R}_1^0\mathbf{S}(\mathbf{r}_{12}^1){\mathbf{R}_1^0}^T\mathbf{R}_1^0\boldsymbol{\omega}_1^1 \\ \mathbf{R}_1^0\mathbf{R}_2^1\boldsymbol{\omega}_2^2 &= \mathbf{R}_1^0\boldsymbol{\omega}_1^1. \end{aligned} \tag{4.57}$$

These equations can be multiplied from both ends with $\mathbf{R}_1^{-1}$ and $\mathbf{R}_2^{1^{-1}} = \mathbf{R}_1^2$ and the result can be written as matrix form

$$\begin{bmatrix} \dot{\mathbf{p}}_2^2 \\ \boldsymbol{\omega}_2^2 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_1^2 & -\mathbf{R}_1^2\mathbf{S}(\mathbf{r}_{12}^1) \\ \mathbf{0} & \mathbf{R}_1^2 \end{bmatrix} \begin{bmatrix} \dot{\mathbf{p}}_1^1 \\ \boldsymbol{\omega}_1^1 \end{bmatrix}. \tag{4.58}$$

The equation (4.58) represents basic velocity transformation from the initial into the new coordinate frame. The transformation matrix in

the equation (4.58) has a role of the Jacobian matrix. It represents the velocity transformation and this equation can be also written as

$$\mathbf{v}_2^2 = \mathbf{J}_1^2 \mathbf{v}_1^1. \tag{4.59}$$

The equation 4.59 illustrates the relationship between conversion of the velocities between two coordinate frames where $\mathbf{v}_1^1 = \begin{bmatrix}\dot{\mathbf{p}}_1^1 & \boldsymbol{\omega}_1^1\end{bmatrix}^T$, $\mathbf{v}_2^2 = \begin{bmatrix}\dot{\mathbf{p}}_2^2 & \boldsymbol{\omega}_2^2\end{bmatrix}^T$ and

$$\mathbf{J}_1^2 = \begin{bmatrix}\mathbf{R}_1^2 & -\mathbf{R}_1^2\mathbf{S}(\mathbf{r}_{12}^1) \\ \mathbf{0} & \mathbf{R}_1^2\end{bmatrix}. \tag{4.60}$$

2 Force and torque transformation

At the begging of the chapter we introduced the concept of the kineto-static duality.

The equations 4.61 and 4.62 represent kineto-static duality for transformation of velocities and forces between two coordinate systems.

$$\dot{\mathbf{x}} = \mathbf{J}\dot{\mathbf{q}}, \tag{4.61}$$

$$\boldsymbol{\tau} = \mathbf{J}^T\mathbf{F}. \tag{4.62}$$

The equation (4.61) defines the transformation between the velocities expressed in joint coordinate frame into the velocities at the top of the robot. On the other hand the equation (4.62) defines the transformation of the forces on the top of the robot into the joint torques. The relation that makes the transformation is Jacobian and transposed Jacobian matrix. The duality presented with equations (4.61) and (4.62) will be used to define force and torque transformations.

For this reason it is necessary to find the transformation that is dual to the transformation (4.59). Regarding to (4.62) we can write

$$\boldsymbol{\gamma}_1^1 = \mathbf{J}_1^{2^T}\boldsymbol{\gamma}_2^2, \tag{4.63}$$

where vector $\boldsymbol{\gamma}_1^1 = \begin{bmatrix}\mathbf{f}_1^1 & \boldsymbol{\mu}_1^1\end{bmatrix}^T$ represents forces and torques expressed in coordinate frame $O_1 - x_1y_1z_1$ and vector $\boldsymbol{\gamma}_2^2 = \begin{bmatrix}\mathbf{f}_2^2 & \boldsymbol{\mu}_2^2\end{bmatrix}^T$ represents forces and torques expressed in coordinate frame $O_2 - x_2y_2z_2$. The matrix $\mathbf{J}_1^{2^T}$ is equal to

$$\mathbf{J}_1^{2^T} = \begin{bmatrix}\mathbf{R}_1^{2^T} & \mathbf{0} \\ (-\mathbf{R}_1^2\mathbf{S}(\mathbf{r}_{12}^1))^T & \mathbf{R}_1^{2^T}\end{bmatrix}. \tag{4.64}$$

Considering the characteristic of the oblique-symmetric matrix (4.51) the transformation $\left(-\mathbf{R}_1^2\mathbf{S}(\mathbf{r}_{12}^1)\right)^T$, with an use of certain rules

$$(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T, \quad \mathbf{R}_1^{2T} = \mathbf{R}_1^{2^{-1}} = \mathbf{R}_2^1, \quad \mathbf{S} = -\mathbf{S}^T, \tag{4.65}$$

can be written as

$$\left(-\mathbf{R}_1^2\mathbf{S}(\mathbf{r}_{12}^1)\right)^T = -\mathbf{S}^T(\mathbf{r}_{12}^1)\mathbf{R}_2^1 = \mathbf{S}(\mathbf{r}_{12}^1)\mathbf{R}_2^1. \tag{4.66}$$

This gives us the final equation, which allows the transformation of forces and torques between the two coordinate frames

$$\begin{bmatrix} \mathbf{f}_1^1 \\ \boldsymbol{\mu}_1^1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_2^1 & \mathbf{0} \\ \mathbf{S}(\mathbf{r}_{12}^1)\mathbf{R}_2^1 & \mathbf{R}_2^1 \end{bmatrix} \begin{bmatrix} \mathbf{f}_2^2 \\ \boldsymbol{\mu}_2^2 \end{bmatrix}. \tag{4.67}$$

3 The student tasks

At the top of the robot Stäubli a weight is attaches as is shown on Figure 4.2. A force that this weight produces is measured with a force and torque sensor attached at the top of the robot. This sensor measures the forces and torques in it's own coordinate frame. You need to calculate what is the actual force and torque at the end of the handle and in the base coordinate frame of the robot.

In the first exercise you developed a direct kinematics for robot Stäubli. The vector of measured joint angles can be used in forward kinematics function. The measured forces on the top of the robot can read with a commands `[w, ft, q] = read_staubli`, where `w` defines the pose of the robot end coordinate system, `ft` represents the current measured forces and torques and vector `q` holds current joint angles.

Two Matlab templates can be found further in a text to transform the forces and torques from the sensor coordinate system into other coordinate systems. You need to fill the lines marked with `%%% STUDENT %%%`.

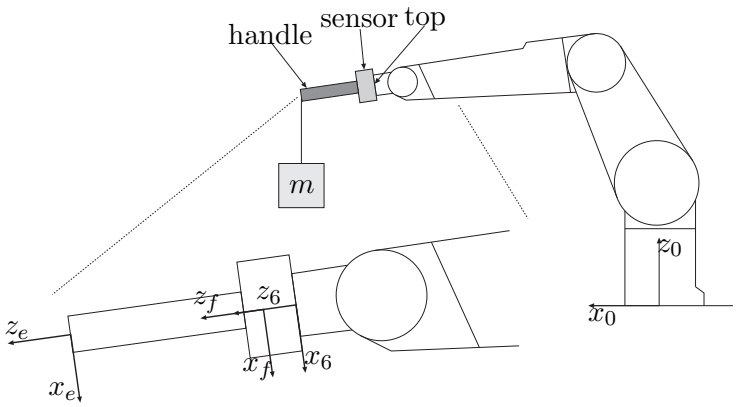

Figure 4.2: Setting of the base, force sensor and handle coordinate frames.

The transformation 4.68 between robot top coordinate frame and force sensor coordinate frame is defined by

$$T_{6f} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.035 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.68}$$

and the transformation 4.69 between force sensor coordinate frame and handle coordinate frame by

$$T_{fe} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.14 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.69}$$

4 Necessary knowledge and goal of the exercise

The goal of the exercise is to test the concept of kineto-static duality in real situation to transform forces and torques among different coordinate systems. It is necessary to have knowledge of transformational matrices and Matlab scriping.

5 Matlab templates

Template for function: **fte = ft2endeffector(q, ft)**

```
% Function recalculates the forces and torques measured by force and torque sensor
% into the handle coordinate frame.
% Input vector q gives the information about joint angles and ft vector gives
% current force and torque measured by the sensor.
% The function output is a vector fte of the forces and torques expressed in the
% handle coordinate frame.

% Transformation from robot top frame to the sensor frame
    t6F = [eye(3),[0; 0; 0.035]; [0 0 0 1]];

% Transformation from sensor frame to the handle frame
    tFE = [eye(3),[0; 0; 0.14]; [0 0 0 1]];

% Calculate transformation T for force and torque transformation
% between sensor and handle frame
    T = ;                                                  %%% STUDENT %%%

% The output force and torque vector
    fte = ;                                                %%% STUDENT %%%
```

Test result:

$ft2endeffector([1,2,3,4,5,6]', [1,2,3,4,5,6]') = \begin{bmatrix} 1 & 2 & 3 & 4.28 & 4.86 & 6 \end{bmatrix}^T$

Template for function: **ftb = ft2base(q, ft)**

```
% Function recalculates the forces and torques measured by force and torque sensor
% into the base coordinate frame.
% Input vector q gives the information about joint angles and ft vector gives
% current force and torque measured by the sensor.
% The function output is a vector ftb of the forces and torques expressed in the
% base coordinate frame.

% Transformation from robot top frame to the sensor frame
    t6F = [eye(3),[0; 0; 0.035]; [0 0 0 1]];

% Transformation from sensor frame to the handle frame
    tFE = [eye(3),[0; 0; 0.14]; [0 0 0 1]];

% Array of the matrices A
    A = ;                                                  %%% STUDENT %%%

% Transformation matrix of the robot top pose
    T6 = ;                                                 %%% STUDENT %%%

% Calculate transformation T for force and torque transformation
% between sensor and base frame
    T = ;                                                  %%% STUDENT %%%

% The output force and torque vector
    ftb = ;                                                %%% STUDENT %%%
```

For identity (I) use function eye(N,M) nad for zero matrix use zeros(N,M)!

Test result:

$$ft2base([1,2,3,4,5,6]', [1,2,3,4,5,6]') =$$
$$\begin{bmatrix} -1.4153 & -0.7634 & 3.3785 & -5.1061 & -1.9701 & 6.8875 \end{bmatrix}^T$$

TRAJECTORY PLANNING

During previous exercises we were generating different mathematical models of robot mechanisms, especially kinematics. But before this knowledge can be used for robot control we need to look into the field of trajectory planning. The goal of trajectory planning is to generate reference inputs to the control system, which will ensure that the top of the robot moves along the desired trajectory. The movement of the robot manipulator is usually defined in external coordinates. In the simplest tasks only the start and end points of the top of the robot is defined and this approach will also be used in our exercise.

In the field of robotics several methods for robot trajectory planing can be found. For point-to-point interpolation linear interpolation and 3th and 5th degree polynomial approach is mainly used. Besides these also other interpolation approaches were developed as on-line path planning with an use of A* algorithm and B-splines approach.

To demonstrate trajectory interpolation during laboratory practice we will use the simplest approach, where user defines start and end point and the trajectory between these to points is linear.

1 Trajectory interpolation between two points

Many times the trajectory of the top of the robot is not important and only joint interpolation is sufficient. But usually the user needs a movement of the top of the robot between two points in a line, where several joints need to move. The point in space of the robot top is

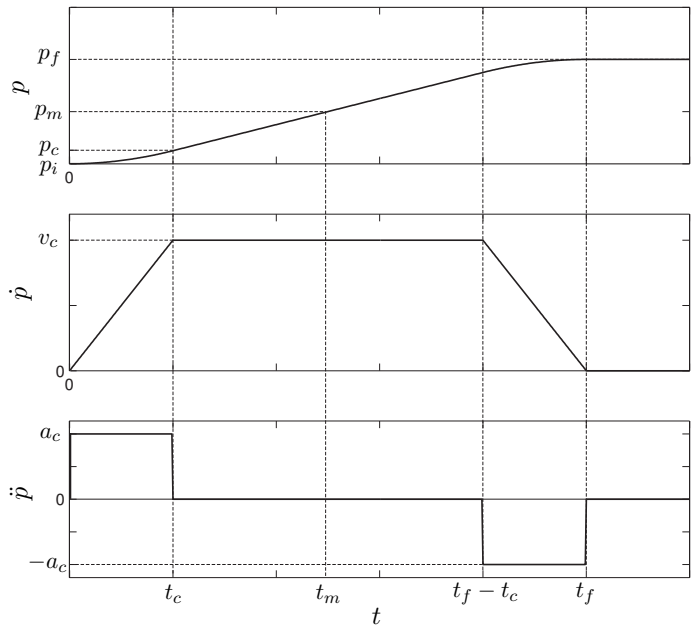

Figure 5.1: The time chart of variables at the trapezoidal velocity profile.

usually marked with a letter p. When the robot is moving from start to end position, the start position is marked with a letter p_i and the end point with a letter p_f. The move is completed in time marked with a letter t_f.

When using industrial robots most frequently *trapezoidal velocity profile* is used. Start of movement is at the time $t = 0$ with the constant acceleration phase. Follows the phase where the velocity is constant. The movement ends with the phase where the robot top decelerate at the constant rate (Figure 5.1). The resultant time progress of the movement of the top of the robot is composed of a central linear part, which begins and ends with a parabolic segment. The start and end velocity are equal to zero. The phase of the constant acceleration lasts the same as the phase of constant deceleration. In both cases the acceleration rate is the same a_c. This gives the symmetrical trajectory where following can be written

$$p_m = \frac{p_f + p_i}{2} \quad ; \quad t_m = \frac{t_f}{2}. \tag{5.70}$$

However, to achieve linear motion of the robot top it is necessary to take care, that desired velocity and the desired acceleration and deceleration apply only for the longest path according to base coordinate frame axes. For all other moves given velocities and acceleration and deceler-

ation do not apply, but must be calculated according to the parameters of the longest path.

The trajectory $p(t)$ has to fulfill some restrictions to move the robot top from the start position p_i into final position p_f in time t_f. The velocity at the end of the parabolic phase has to be equal to the constant velocity of the linear phase. This equal velocity can be calculated by the use of the equation

$$v = a_c t, \tag{5.71}$$

that describes the movement where the acceleration is constant. At the end of the first phase is

$$v_c = a_c t_c. \tag{5.72}$$

The velocity in the second phase can be obtained out the Figure 5.1

$$v_c = \frac{p_m - p_c}{t_m - t_c}, \tag{5.73}$$

where p_c describes the rate of movement at the end of the parabolic phase in time t_c. During this time the movement is performed with the constant acceleration a_c and the velocity can be written as (5.71). The position can be calculated with integration of the equation (5.71)

$$p = \int v dt = a_c \int t dt = a_c \frac{t^2}{2} + p_i, \tag{5.74}$$

where the integration constant is the start position p_i. At the end of the first phase we can write

$$p_c = p_i + \frac{1}{2} a_c t_c^2. \tag{5.75}$$

The velocity at the end of the first phase (5.72) can be equalized with the constant velocity of the second phase

$$a_c t_c = \frac{p_m - p_c}{t_m - t_c}. \tag{5.76}$$

If the equation (5.75) is put into the equation (5.76) and the term (5.70) is taken into account than following quadratic equation can be written

$$a_c t_c^2 - a_c t_f t_c + p_f - p_i = 0. \tag{5.77}$$

The t_c can be expressed as

$$t_c = \frac{t_f}{2} - \frac{1}{2} \sqrt{\frac{t_f^2 a_c - 4(p_f - p_i)}{a_c}}. \tag{5.78}$$

To generate the movement from initial pose p_i and final pose p_f the following polynomial is generated for the first phase of the movement

$$p_1(t) = p_i + \frac{1}{2}a_c t^2 \qquad 0 \leq t \leq t_c. \tag{5.79}$$

In the second phase of the movement a line is generated with start point $[t_c, p_c]$ and has the slope v_c:

$$(p - p_c) = v_c(t - t_c). \tag{5.80}$$

After some "housekeeping" next equation can be produced

$$p_2(t) = p_i + a_c t_c(t - \frac{t_c}{2}) \qquad t_c < t \leq (t_f - t_c). \tag{5.81}$$

In the last phase again the parabola is generated with the points $[t_f, p_f]$ and flipped downwards

$$p_3 = p_f - \frac{1}{2}a_c(t - t_f)^2 \qquad (t_f - t_c) < t \leq t_f \tag{5.82}$$

This procedure gives the analytic course of the robot top movement between two points in robot work space.

2 The student tasks

In Matlab write functions to carry out the following tasks:

- `function [time, p] = interpolate(t_c, t_f, a_c, dT, p_i, p_f)` generated the interpolated path between two specified coordinates (p_i, p_f) and specified input parameters (t_c, t_f, a_c, dT).
- `function trajectory()` calculates the input parameters for the function *interpolate* and draw the interpolated path for each frame axes and combined movement.
- Open the simulation model *interpolationKUKA.mdl* and test the interpolated trajectory by sending the platform velocity profile to the robot KUKA youBot.

You need to properly fill the lines marked with `%%% STUDENT %%%`.

3 Necessary knowledge and goal of the exercise

To complete the exercise it is necessary to understand basics of physics about velocities and accelerations and basics of Matlab environment. The exercise goal is to generate trajectory (Figure 3.6) for robot KUKA youBot platform with an use of trapezoidal profile.

4 Matlab templates

Template for function: **[time, p] = interpolate(t_c, t_f, a_c, dT, p_i, p_f)**

```
% t_c - Time of acceleration and deceleration
% t_f - The whole movement time
% a_c - Acceleration and deceleration
% dT  - Interpolation sample time
% p_i - Initial position
% p_f - Final position

% The time vector for interpolation phases
    t1 = ;                                          %%% STUDENT %%%
    t2 = ;                                          %%% STUDENT %%%
    t3 = ;                                          %%% STUDENT %%%

% Interpolated path for acceleration, constant velocity
% and deceleration phase
    p1 = ;                                          %%% STUDENT %%%
    p2 = ;                                          %%% STUDENT %%%
    p3 = ;                                          %%% STUDENT %%%

% All vectors combined into one vector
    time = [t1,t2,t3];
    p = [p1,p2,p3];
```

Template for function: **trajectory.m**

```
% Erase old data, clear screen and close windows
    clear all; close all; clc;

% Sample time
    dT = 0.005;

% Robot initial configuration
    p0 = [0 0 0];

% The pose of the robot top at the joint initial values q0.
    pX0 = p0(1);
    pY0 = p0(2);

% The change of the robot platform position in meters
    dpX = 1.0;
    dpY = 1.0;

% Desired velocity in m/s
    v_c = 0.25;

% Acceleration and deceleration (m/s2)
    a_c = 0.1;

% Which move (Y or Z direction) is longer?
% Lp_i ... Long initial
% Lp_f ... Long final
% Sp_i ... Short initial
% Sp_f ... Short final
    if(dpX >= dpY)
        Lp_i = pX0;
        Lp_f = pX0 + dpX;
        Sp_i = pY0;
        Sp_f = pY0 + dpY;
    else
        Lp_i = pY0;
        Lp_f = pY0 + dpY;
        Sp_i = pX0;
        Sp_f = pX0 + dpX;
    end

% *** CALCULATION FOR LONGEST MOVEMENT ***
 % Time of the acceleration
    t_c = ;                                          %%% STUDENTI %%%

 % The position at the end of acceleration phase
    p_c = ;                                          %%% STUDENTI %%%

 %  The movement length when the velocity is constant
    d_l = ;                                          %%% STUDENTI %%%

 % If the length of the linear phase is negative then the acceleration is to high
    if(d_l < 0 )
        sprintf('Acceleration too high!')
        break
    end

 % Duration of the linear phase
    t_l = ;                                          %%% STUDENTI %%%

 % The total time of the movement
    t_f = ;                                          %%% STUDENTI %%%

 % Calculation of the interpolated path for the longest move
    [time1,p1] = ;                                   %%% STUDENTI %%%

% *** CALCULATION FOR SHORTEST MOVEMENT ***
 % Acceleration in the shortest path
    a_c = ;                                          %%% STUDENTI %%%

 % Calculation of the interpolated path for the shortest move
    [time2,p2] = ;                                   %%% STUDENTI %%%

 % Charts display ...
```

NEWTON-EULER DYNAMICS

All laboratory examples (except transformation of forces and torques) were about kinematics; positions, velocities and accelerations. On all robot mechanisms forces and torques influence all the time and those cause the movement of the robot. In the dynamics of the robots we are interested in motion of the robot mechanism as a result of the forces and torques in individual robot joint. The inverse problem is also very interesting, when we are asked what forces and torques in joints are necessary to produce the wished movement of the robot mechanism.

We can talk about robot dynamics, when considering the forces and moments in it and the movement of the mechanism caused by these forces and torques.Tto mathematical describe the robot dynamics we normally use two methods: Langrange and Newton-Euler method . The Newton-Euler will be used as an example during the laboratory practice.

1 The equations for calculation of the robot dynamics model

From the base frame of the robot at first we define the angular velocities and accelerations in each robot joint, translational accelerations of the bases of each segment frames and translational acceleration of the center of masses of the individual segments.

Based on the assumption that the angular velocity and angular acceleration of the initial segment (robot base) are equal to zero ($\boldsymbol{\omega}_0^0 = 0, \dot{\boldsymbol{\omega}}_0^0 = 0$). The joint variables are marked with the letter q and $q \equiv \vartheta$

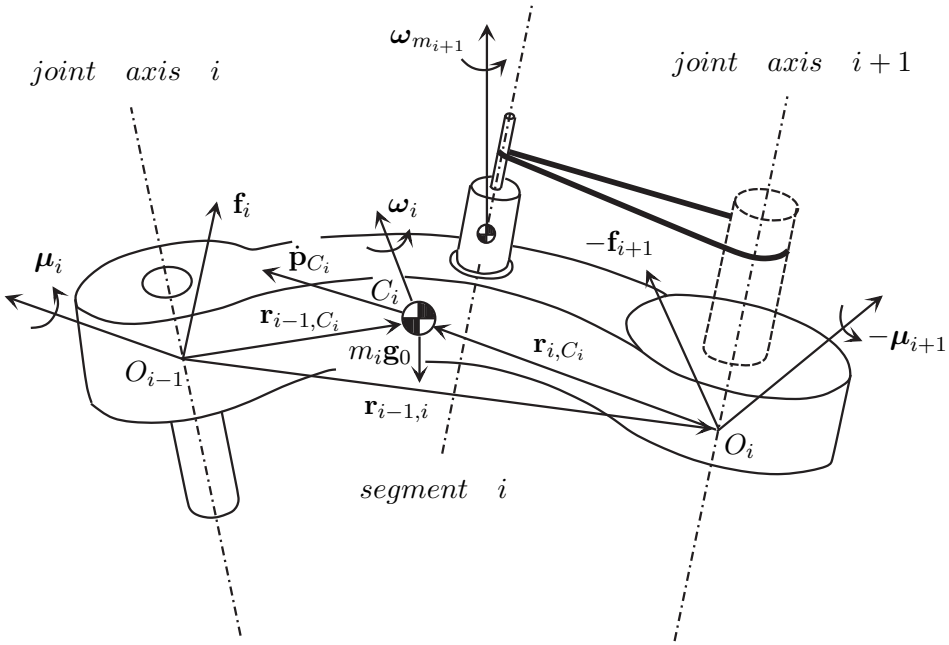

Figure 6.1: Free segment

for rotational joint and $q \equiv d$ for translational joint. The Figure 6.1 shows forces and torques that act on the free segment.

As we know that the sum of all forces acting on a segment is equal of the change in momentum of center of mass of the segment we can write the following relationship

$$\mathbf{f}_i - \mathbf{f}_{i+1} + m_i\mathbf{g}_0 = m_i\ddot{\mathbf{p}}_{C_i}\,, \tag{6.83}$$

and also applies that the sum of all torques that act on the segment is equal to the change in angular momentum about the center of mass of the segment. Following can be written

$$\boldsymbol{\mu}_i + \mathbf{f}_i \times \mathbf{r}_{i-1,C_i} - \boldsymbol{\mu}_{i+1} - \mathbf{f}_{i+1} \times \mathbf{r}_{i,C_i} = \frac{d}{dt}\left(\bar{\mathbf{I}}_i\boldsymbol{\omega}_i\right)\,. \tag{6.84}$$

In further text we will define all kinematic variables that we need for calculation of forces and torques. All quantities are calculated regarding to the local coordinate frame i that makes the calculation simpler and more transparent.

2 The calculation of the inverse dynamic model

The angular velocity of the segment i is defined as

$$\textcircled{1} \quad \boldsymbol{\omega}_i^i = \begin{cases} \mathbf{R}_i^{i-1^T} \boldsymbol{\omega}_{i-1}^{i-1} & \text{translational joint} \\ \mathbf{R}_i^{i-1^T} \left(\boldsymbol{\omega}_{i-1}^{i-1} + \dot{q}_i \mathbf{z}_0 \right) & \text{rotational joint} \end{cases} \tag{6.85}$$

The angular acceleration of the segment i is defined as

$$\textcircled{2} \quad \dot{\boldsymbol{\omega}}_i^i = \begin{cases} \mathbf{R}_i^{i-1^T} \dot{\boldsymbol{\omega}}_{i-1}^{i-1} & \text{translational joint} \\ \mathbf{R}_i^{i-1^T} \left(\dot{\boldsymbol{\omega}}_{i-1}^{i-1} + \ddot{q}_i \mathbf{z}_0 + \dot{q}_i \boldsymbol{\omega}_{i-1}^{i-1} \times \mathbf{z}_0 \right) & \text{rotational joint} \end{cases} \tag{6.86}$$

The translational velocity of the coordinate frame i base is defined as

$$\dot{\mathbf{p}}_i^i = \begin{cases} \mathbf{R}_i^{i-1^T} \left(\dot{\mathbf{p}}_{i-1}^{i-1} + \dot{q}_i \mathbf{z}_0 \right) + \boldsymbol{\omega}_i^i \times \mathbf{r}_{i-1,i}^i & \text{translational joint} \\ \mathbf{R}_i^{i-1^T} \dot{\mathbf{p}}_{i-1}^{i-1} + \boldsymbol{\omega}_i^i \times \mathbf{r}_{i-1,i}^i & \text{rotational joint} \end{cases} \tag{6.87}$$

The translational acceleration of the coordinate frame i base is defined

$$\textcircled{3} \quad \ddot{\mathbf{p}}_i^i = \begin{cases} \mathbf{R}_i^{i-1^T} \left(\ddot{\mathbf{p}}_{i-1}^{i-1} + \ddot{q}_i \mathbf{z}_0 \right) + 2\dot{q}_i \boldsymbol{\omega}_i^i \times \mathbf{R}_i^{i-1^T} \mathbf{z}_0 \\ + \dot{\boldsymbol{\omega}}_i^i \times \mathbf{r}_{i-1,i}^i + \boldsymbol{\omega}_i^i \times \left(\boldsymbol{\omega}_i^i \times \mathbf{r}_{i-1,i}^i \right) & \text{translational joint} \\ \mathbf{R}_i^{i-1^T} \ddot{\mathbf{p}}_{i-1}^{i-1} + \dot{\boldsymbol{\omega}}_i^i \times \mathbf{r}_{i-1,i}^i + \boldsymbol{\omega}_i^i \times \left(\boldsymbol{\omega}_i^i \times \mathbf{r}_{i-1,i}^i \right) & \text{rotational joint} \end{cases} \tag{6.88}$$

The translational acceleration of the center of mass of the segment is

$$\textcircled{4} \quad \ddot{\mathbf{p}}_{C_i}^i = \ddot{\mathbf{p}}_i^i + \dot{\boldsymbol{\omega}}_i^i \times \mathbf{r}_{i,C_i}^i + \boldsymbol{\omega}_i^i \times \left(\boldsymbol{\omega}_i^i \times \mathbf{r}_{i,C_i}^i \right) \tag{6.89}$$

All the calculations of the forces and torques in each joint is started at the top of the robot. It is essential that we know force and torque that is acting on the robot top. If the robot is not in contact with the any object than we can assume that $\mathbf{f}_{n+1}^{n+1} = 0$ and $\mu_{n+1}^{n+1} = 0$, where n is the number of robot joints. The force that a segment $i-1$ is producing on the segment i is equal to

$$\textcircled{5} \quad \mathbf{f}_i^i = \mathbf{R}_{i+1}^i \mathbf{f}_{i+1}^{i+1} + m_i \ddot{\mathbf{p}}_{C_i}^i \tag{6.90}$$

The torque that a segment $i-1$ is producing on the segment i is

$$\textcircled{6} \quad \mu_i^i = -\mathbf{f}_i^i \times \left(\mathbf{r}_{i-1,i}^i + \mathbf{r}_{i,C_i}^i\right) + \mathbf{R}_{i+1}^i \mu_{i+1}^{i+1} + \mathbf{R}_{i+1}^i \mathbf{f}_{i+1}^{i+1} \times \mathbf{r}_{i,C_i}^i + \mathbf{I}_i^i \dot{\boldsymbol{\omega}}_i^i + \boldsymbol{\omega}_i^i \times \left(\mathbf{I}_i^i \boldsymbol{\omega}_i^i\right) \tag{6.91}$$

Generalized forced acting on the individual joint are defined as

$$\textcircled{7} \quad \tau_i = \begin{cases} \mathbf{f}_i^{i\,T} \mathbf{R}_i^{i-1\,T} \mathbf{z}_0 + F_{vi}\dot{q}_i + F_{si} sgn(\dot{q}_i) & \text{translational joint} \\ \\ \mu_i^{i\,T} \mathbf{R}_i^{i-1\,T} \mathbf{z}_0 + F_{vi}\dot{q}_i + F_{si} sgn(\dot{q}_i) & \text{rotational joint} \end{cases} \tag{6.92}$$

3 The calculation of the direct dynamic model

By using the equations (6.92) it is possible to calculate the joint torques.

$$\boldsymbol{\tau} = \mathbf{B}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{F}_v \dot{\mathbf{q}} + \mathbf{F}_s \mathrm{sgn}(\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}). \tag{6.93}$$

For robot dynamics simulation the direct dynamic model is needed. This model uses the torques generated by joint motors as calculated the robot movement.

$$\ddot{\mathbf{q}} = \mathbf{B}^{-1}(\mathbf{q}) \left(\boldsymbol{\tau} - \left(\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{F}_v \dot{\mathbf{q}} + \mathbf{F}_s \mathrm{sgn}(\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})\right)\right). \tag{6.94}$$

With an use of Newton-Euler approach it is very easy to get to the direct dynamic model of the robot. The base for all calculations is the equation (6.93). Column $\mathbf{b}_i$ of the inertia matrix of the robot $\mathbf{B}$ can be calculated when the $\mathbf{g}_0 = \mathbf{0}$, $\dot{\mathbf{q}} = \mathbf{0}$, $\ddot{q}_i = 1$ and $\ddot{q}_j = 0$, if $j \neq i$. This approach makes $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = 0, \mathbf{F}_v \dot{\mathbf{q}} = 0$ and $\mathbf{g}(\mathbf{q}) = 0$. The parameter of a static friction $\mathbf{F}_S$ is not included. The individual column $\mathbf{b}_i$ of the inertia matrix of the robot $\mathbf{B}$ is calculated by the next procedure.

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} \Rightarrow \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} b_{12} \\ b_{22} \\ b_{32} \end{bmatrix} \Rightarrow \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} b_{13} \\ b_{23} \\ b_{33} \end{bmatrix}$$
$$\text{pri } \ddot{q}_1 = 1 \qquad \text{pri } \ddot{q}_2 = 1 \qquad \text{pri } \ddot{q}_3 = 1 \tag{6.95}$$

The term in brackets $(\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{F}_v \dot{\mathbf{q}} + \mathbf{F}_s + \mathbf{g}(\mathbf{q}))$ is possible to calculate when we use $\ddot{\mathbf{q}} = \mathbf{0}$ in the equation (6.93). The result is:

$$\tau = \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{F}_v \dot{\mathbf{q}} + \mathbf{F}_s + \mathbf{g}(\mathbf{q}). \tag{6.96}$$

The simulation model in Simulink is shown on the Figure 6.2.

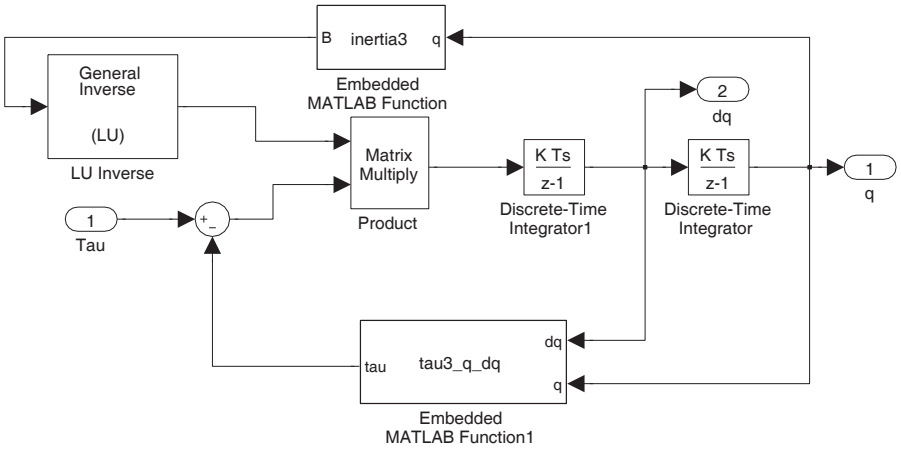

Figure 6.2: The direct dynamics Simulink model

4 The student tasks

The calculation include only the first 3 robot segments.

- In Matlab fill in the following functions `function tau = torque(ddq, dq, q, ddp0)`, `function B = inertia3(q)` and `function tau = tau3_q_dq(dq, q)`.

  The function `torque(ddq, dq, q, ddp0)` for the current robot joint angles as vector **q**, values of joint velocities as a vector **dq**, values of joint accelerations as a vector **ddqq** and acceleration vector acting on the base of the robot **ddp0**, calculates the inverse dynamic model of the robot (joint torques vector **tau**).

  The function `inertia3(q)` for robot joint angles as a vector **q**, calculates inertia matrix of the robot; use the basic function for inverse dynamic model calculation for the robot.

  The function `tau3_q_dq(dq, q)` for the joint angles as a vector **q** and joint velocities as a vector **dq**, calculates the torques in the robot joints. These torques are the result of Coriolis, gravity and friction; use the basic function for inverse dynamic model calculation for the robot.

  You need to properly fill the lines marked with `%%% STUDENT %%%`.

- In the Simulink make a simulation scheme with the following components: interpolated trajectory, inverse kinematics of the robot based on the inverse Jacobian matrix, direct dynamic model and simple PD controller (PD controller is shown on the Figure 6.3). The whole scheme is shown on the Figure 6.4. The simulation scheme needs to be tested on the real robot KUKA youBot.

5 Necessary knowledge and goal of the exercise

The exercise requires knowledge of Matlab programming and using Matlab Simulink. It is also necessary to master the theory of direct kinematics and have basics of the Newton-Euler method.

6 Matlab templates

Template for function: **tau = torque(ddq, dq, q, ddp0)**

```
% Function calculates the inverse dynamic model for the robot
% with three degrees of freedom.
% Input vectors are the joint position q, joint velocities dq,
% joint accelerations ddq and vector ddp0 that defines the robot base acceleration.
% The output is a vector of joint torques tau.

% Segment mass
    m = [0 1.318 0.821];
% Vectors from joint i-1 to i.
    r = [[0.033 0 0.147]' [0.155 0 0]' [0.135 0 0]'];
% Vectors of the center of gravity of the segments in the local coordinate frame
rC=[[0.01516 0.00359 0.03105]',[0.11397 0.0150 -0.01903]',[0.10441 0.00013 0.02022]'];
% Segments inertia matrices
    I = zeros(3,3,3);
    I(:,:,1) = [0.0029525 0 0; 0 0.0060091 0; 0 0 0.0058821];
    I(:,:,2) = [0.0031145 0 0; 0 0.0005843 0; 0 0 0.0031631];
    I(:,:,3) = [0.00041967 0 0; 0 0.00172767 0; 0 0 0.0018468];
% Viscous friction parameters for individual joints
    fv = [2 1 1];
% The data structure (mass of the 1st segment is rob.m(1))
    rob = struct('m', m, 'rC', rC, 'r', r, 'I', I, 'fv', fv);

% Transformation matrices calculation for the first three segments
    d1=0.147;        a2=0.155;
    a1=0.033;        a3=0.135;
    A = zeros(4,4,3);
    A(:,:,1) = ;                                        %%% STUDENT %%%
    A(:,:,2) = ;                                        %%% STUDENT %%%
    A(:,:,3) = ;                                        %%% STUDENT %%%

% Vector in z0 direction
    z0 = [0 0 1]';
% Torque vector initialization
    tau = zeros(3,1);

% The rotational matrices as submatrix of the A matrices
    R = zeros(3,3,3);
    R(:,:,1) = ;                                        %%% STUDENT %%%
    R(:,:,2) = ;                                        %%% STUDENT %%%
    R(:,:,3) = ;                                        %%% STUDENT %%%
```

```
% ALL FURTHER INITIALIZATIONS ARE MADE FOR THREE ROBOT JOINTS AND SEGMENTS.
% THIS MAKES ALL THE MATRICES WITH DIMENSIONS 3x3, WHERE EACH
% COLUMN REPRESENTS A VECTOR DESCRIBING ONE ROBOT SEGMENT OR JOINT.

% The segments angular velocities vectors initialization
       omega = zeros(3,3);

% The segments angular acceleration vectors initialization
       domega = zeros(3,3);

    % The segments translational acceleration of the coordinate frames vectors
    % initialization
        ddp = zeros(3,3);

    % The segments translational acceleration of the center of masses vectors
    % initialization
        ddpC = zeros(3,3);
    % The joint forces vectors initialization
        f = zeros(3,3);
    % The joint torques vectors initialization
        mi = zeros(3,3);

    % The kinematics calculation
    for ii = 1:3
        if (ii == 1) % Boundary condition for the first joint
            % Segment angular velocity
(1)             omega(:,ii) = ;                              %%% STUDENT %%%
            % Segment angular acceleration
(2)             domega(:,ii) = ;                             %%% STUDENT %%%
            % Segment translational acceleration
(3)             ddp(:,ii) = ;                                %%% STUDENT %%%
        else
            % Segment angular velocity
(1)             omega(:,ii) = ;                              %%% STUDENT %%%
            % Segment angular acceleration
(2)             domega(:,ii) = ;                             %%% STUDENT %%%
            % Segment translational acceleration
(3)             ddp(:,ii) = ;                                %%% STUDENT %%%
        end
        % Segment translational acceleration of the center of mass
(4)          ddpC(:,ii) = ;                                  %%% STUDENT %%%
    end

    % The dynamics calculation
    for ii = 3:-1:1
        if (ii==3) % Force boundary condition for the top of the robot
            % The force acting on the joint.
(5)             f(:,ii) = ;                                  %%% STUDENT %%%
            % The torque acting on the joint
(6)             mi(:,ii) = ;                                 %%% STUDENT %%%
        else
            % The force acting on the joint
(5)             f(:,ii) = ;                                  %%% STUDENT %%%
            % The torque acting on the joint
(6)             mi(:,ii) = ;                                 %%% STUDENT %%%
        end
        % The torque acting in the joint axis (strain on the motor)
(7)          tau(ii) = ;                                     %%% STUDENT %%%
    end
```

Test result:

$$torque([1,2,3]', [4,5,6]', [7,8,9]', [0,0,9.81]') = \begin{bmatrix} 5.4961 & 1.5868 & 8.3222 \end{bmatrix}^T$$

Template for function: **B = inertia3(q)**

```
% Function calculates the robot inertia matrix with three segments.
% The input is vector of joint angles q.

% Inertia matrix initialization
    B = zeros(3,3);

% Considering the condition for acceleration acting on the base of the robot
    ddp0 = ;                                          %%% STUDENT %%%

% Considering the condition for angular accelerations in joints
    dq = ;                                            %%% STUDENT %%%


% Calculation of the first column of the inertia matrix B where we use
% function for calculation of the robot inverse dynamics for torque calculation,
% that are represented as columns in inertia matrix.
    ddq = ;                                           %%% STUDENT %%%
    B(:,1) =;                                         %%% STUDENT %%%

% Calculation of the second column of the inertia matrix B
    ddq = ;                                           %%% STUDENT %%%
    B(:,2) = ;                                        %%% STUDENT %%%

% Calculation of the third column of the inertia matrix B
    ddq = ;                                           %%% STUDENT %%%
    B(:,3) = ;                                        %%% STUDENT %%%
```

Test result:

$$inertia3([1,2,3]') = \begin{bmatrix} 0.1528 & 0.0207 & -0.0093 \\ 0.0207 & 0.1071 & 0.0187 \\ -0.0093 & 0.0187 & 0.0489 \end{bmatrix}$$

Template for function: `tau = tau3_q_dq(dq, q)`

```
function tau = tau3_q_dq(dq, q)

    % The function calculates the torques in robot joints as a result of
    % Coriolis, gravity and friction components.
    % Input vectors are the joint position q, joint velocities dq.
    % The output is a vector of joint torques tau.


    % Considering the condition for acceleration acting on the base of the robot
        ddp0 = ;                                      %%% STUDENT %%%

    % Considering the condition for angular accelerations in joints
        ddq = ;                                       %%% STUDENT %%%

    % By the use of the function that calculates the inverse dynamic model
    % we calculate the joint torques
        tau = ;                                       %%% STUDENT %%%
```

Test result:

$$tau3\_q\_dq([1,2,3]',[4,5,6]') = \begin{bmatrix} 0.7680 & 8.6224 & 4.8946 \end{bmatrix}^T$$

7 Simulation scheme in the Matlab Simulink

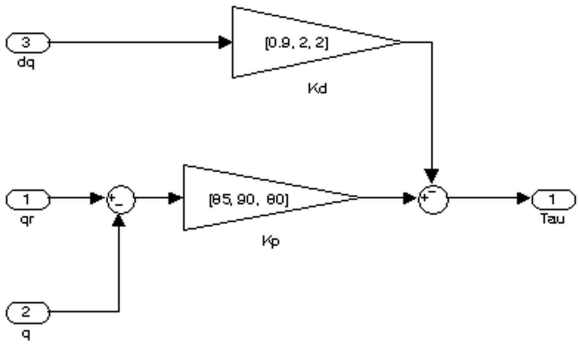

Figure 6.3: Simple PD controller. qr represents the reference position, q is current position and dq is the velocity of the robot. Kp and Kd are position and velocity gains of the controller.

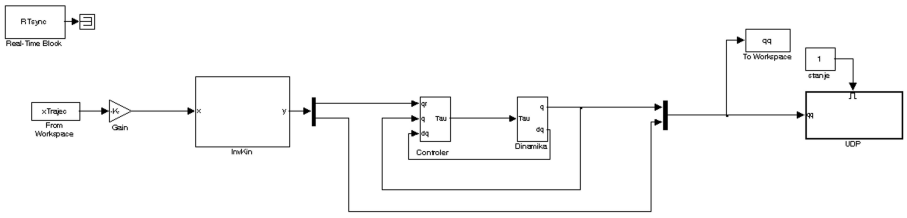

Figure 6.4: Simulation scheme that includes components: interpolated path, robot inverse kinematics on the basis of inverse Jacobian matrix, direct dynamic model and simple PD controller.

Where Simulink objects can be found:
Embadded MATLAB Func. ⇒ *Simulink* → *User-Defined Func.*
Matrix Multiply ⇒ *Simulink* → *Math Operations*
Descrete-Time Integrator ⇒ *Simulink* → *Discrete*
Gain value: 1.0
Initial condition: [0 0 0] in q0(1:3)
Sample time: dT
LU Inverse ⇒ *Signal Processing Blockset* → *Math Functions*

To open the Simulink template use the Matlab command line and run program *initVR_dynamics.m*!

HOME EXERCISES

The goal of home exercises is to prepare the student for laboratory practices. The teaching assistant will check the students home exercises randomly and can also ask a question regarding the home exercise topic. The home exercises are also marked with signs +, +o, o, o- and - and these marks are included in the final 15 % of the student mark.

If the student:

- don't have the appropriate home exercise regarding the topic of the laboratory practice,
- don't do the home exercise alone and the knowledge is not appropriate,

the student will leave the current laboratory practice with negative mark.

The home exercises can be solved with or without the use of Matlab environment. The calculation procedure and results are written in the blank space after the home exercise text.

1 Robot geometrical model

Exercise

Your task before the 1st laboratory practice is study of the kinematic model of the robot based on Denavit-Hartenberg method. In regard to the figures 1.3 and 1.4 fill in the tables 1.1 and 1.2 as a basis for successful finishing of the 1st laboratory practice named *Robot geometrical model*.

Solution

The tables can be redrawn on this page under the Solution or you can fill the tables 1.1 and 1.2 on the beginning of this manual.

2 Geometrical Jacobian matrix

Solution

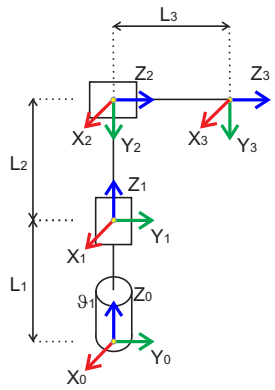

Figure 7.1: The robot in reference pose with drawn coordinate systems

For the robot in Figure 7.1 write down the matrices A_1 (the pose of the 1st coordinate system in regard to the base coordinate system), A_2 (the pose of the 2nd coordinate system in regard to the 1st coordinate system) and T_2 (the pose of the top of the robot coordinate system in regard to the base coordinate system), where following parameters needs to be included: $L_1 = 0.1$, $L_2 = 0.4$, $L_3 = 0.2$, $\vartheta_1 = 0$, $d_2 = 0$ and $d_3 = 0$. Use the derived matrices to write geometrical Jacobian matrix for the robot in depicted configuration.

Results

3 The relation between geometrical and analytical Jacobian matrix and inverse kinematics with the use of analytical Jacobian matrix

Solution

- For the robot on the figure 7.1 calculate the ZYX Euler angles $(\varphi, \vartheta, \psi)$ with the initial parameters from the previous home exercise.
- Calculate the matrix $T(\phi)$ as a submatrix of the matrix $T_A(\phi)$ with an use of the Euler angles from previous task. The matrix $T_A(\phi)$ is the transformation matrix between analitical and geomaterical Jacobian matrix.
- In this manual on the figure 3.4 we can find a sketch of the inverse kinematics. For faster work during the laboratory practice draw a real Simulink model of the inverse kinematics with right connection lines and dummy blocks. The real blocks will be picked out of the Simulink library during laboratory practice.

Solution

4 Force and torque transformation

Exercise

On the robot in figure 7.1 a force is influencing in Z direction of the robot end coordinate system. The force is 10 N. With an use of the theory in this manual, calculate the force expressed in the base coordinate system. The result should be given in force and torque vector form. Use the parameters from the 2nd home exercise!

Solution

5 Trajectory planning

Exercise

The problem involves the 1 DOF (Degree Of Freedom) active translational mechanism in a worm drive form. The mechanism is controlled by trapezoidal velocity profile. The initial worm drive position is at 0.0 m. The initial worm drive acceleration is 0.1 m/s^2 till the final velocity of 0.5 m/s is reached. The mechanism reaches the half of wished traveling distance in 10 seconds.

Solve the following tasks:

- What is the time that worm drive reaches the 1/3 of movement during the acceleration?
- What is the traveling distance after the constant velocity part of the movement is finished?
- When the worm drive stops and what is the final position?

Solution

6 Newton-Euler dynamics

Exercise

- Calculate the angular velocity of the 1st, 2nd and 3rd joint (ω_1^1, ω_2^2 and ω_3^3) where the 1st joint rotational velocity ($(\dot{q}_1)$) is 0.3. The robot parameters are the same as in 2nd home work.
- Calculate the force (f_3^3) acting on the 3nd joint if the mass of the 3nd segment is 0.3 and the translational acceleration of center of mass of the 3nd segment is [0.3 0.8 0.06]. The force on the top of the robot is 0.

Solution