Lehrstuhl für
Technische Dynamik

Computational Multibody Dynamics

Exercises

effective: February 6, 2023

Friedrich-Alexander-Universität
Technische Fakultät

Dr.-Ing. Giuseppe Capobianco
Lehrstuhl für Technische Dynamik
Universität Erlangen-Nürnberg
Immerwahrstraße 1
D-91058 Erlangen
Tel 09131 8561008
Fax 09131 8561011
Mail giuseppe.capobianco@fau.de
http://www.ltd.tf.uni-erlangen.de

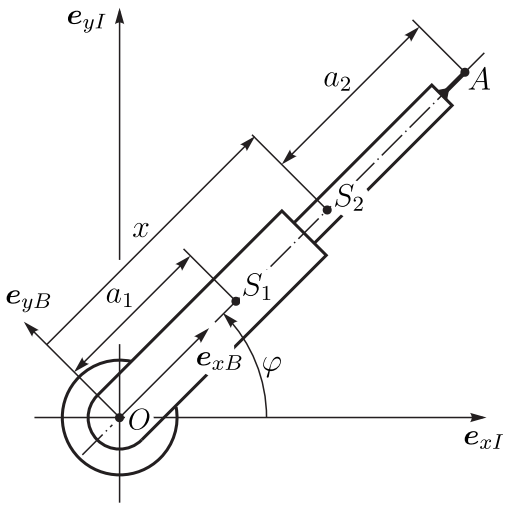
Editor: G. Capobianco
Tel 09131 8561008
Mail giuseppe.capobianco@fau.de

Contents

1 Kinematics

1.1 Robot arm

Consider a planar robot that consists of a fixed basis and a pivoting arm that is fixed at the origin O and can rotate around the axis $\boldsymbol{e}_{zI} = \boldsymbol{e}_{zB}$. This rotation of the arm fixed basis B w.r.t. the inertial basis I is described by the angle φ. The arm is composed of two bodies with respective center of mass S_1 and S_2. The distance between O and S_1 is a_1 and the distance between the end effector A and S_2 is a_2. The two bodies are connected via a linear guidance and the elongation of the arm is described by the coordinate x such that $\boldsymbol{r}_{OS_2} = x\,\boldsymbol{e}_{xB}$. The generalized coordinates of the robot are $\boldsymbol{q}^T = (\varphi, x)$ and the generalized velocities are chosen to be $\boldsymbol{u} = \dot{\boldsymbol{q}}$.



(a) Compute the representations of the position vectors $\boldsymbol{r}_{OS_1}$, $\boldsymbol{r}_{OS_2}$ and $\boldsymbol{r}_{OA}$ with respect to the bases I and B, respectively.

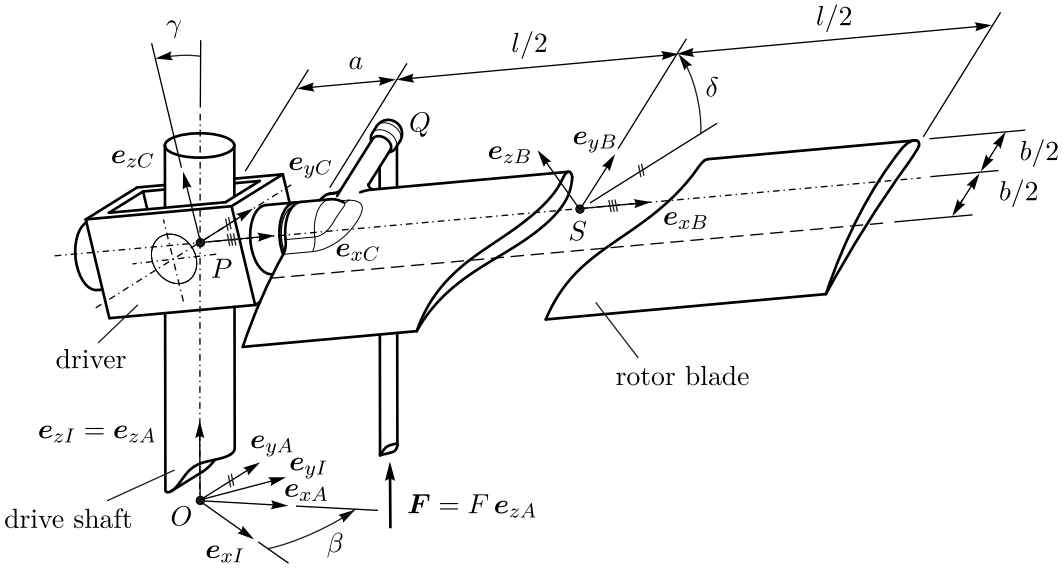
(b) Calculate the representation ${}^B\boldsymbol{\omega}_{IB}$ of the angular velocity of the basis B w.r.t. I. Moreover, compute the representations of the angular velocities $\boldsymbol{\Omega}_1$ and $\boldsymbol{\Omega}_2$ w.r.t. the basis B, respectively.

(c) Compute the representation of the velocity $\boldsymbol{v}_A$ with respect to B in two ways. First, by differentiation of the representations in the inertial system, second, by means of Euler's rule. Which is the most efficient way?

(d) Determine the representation of the acceleration $\boldsymbol{a}_A$ w.r.t. the basis I.

1.2 Helicopter rotor blade

In the following, the kinematics of a helicopter rotor blade is analyzed. The rotor blade is driven by a shaft, whose axis passes through the origin O of the inertial system I and is oriented in $\boldsymbol{e}_{zI}$ direction.



The basis A, which is fixed to the shaft, is rotated w.r.t. the inertial basis I by the angle β around the $\boldsymbol{e}_{zI}$. At a fixed height, the flapping hinge connects the driver to the shaft at the point P. The driver fixed basis C is rotated with respect to the basis A by the angle γ around the negative $\boldsymbol{e}_{yA}$ axis. The inclination of the rotor blade with respect to the driver is described by the angle δ, i.e., the blade fixed basis B is rotated w.r.t. the basis C by the angle δ around the $\boldsymbol{e}_{xC}$ axis.

(a) Provide the transformation matrices ${}^{CA}\boldsymbol{T}$ and ${}^{BC}\boldsymbol{T}$.

(b) Calculate the representation ${}^{C}\boldsymbol{\omega}_{IC}$ angular velocity of the basis C w.r.t. the basis I.

(c) Compute the position vector ${}^{C}\boldsymbol{r}_{PS}$, the velocity ${}^{C}\boldsymbol{v}_{S}$ and the acceleration ${}^{C}\boldsymbol{a}_{S}$.

(d) Determine the angular velocity ${}^{B}\boldsymbol{\Omega}$ of the rotor blade.

1.3 Tait–Bryan angles

The rotation of a free rigid body can be described by Tait–Bryan angels $\boldsymbol{p}^T = (\alpha, \beta, \gamma)$. These parametrize the body fixed system B by three successive elementary rotations $I \to A \to C \to B$, starting from the inertial system I. The corresponding transformation matrices are

$$
{}^{AI}\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{pmatrix}
{}^{CA}\boldsymbol{T} = \begin{pmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{pmatrix}
{}^{BC}\boldsymbol{T} = \begin{pmatrix} \cos(\gamma) & \sin(\gamma) & 0 \\ -\sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix}
$$

(a) Sketch the relative rotation of the coordinate systems I, A, C, B. Which axis remains invariant under which rotation?

(b) Calculate the components ${}^{I}\boldsymbol{\omega}_{IB}$ of the angular velocity of the basis B w.r.t. the basis I represented in the basis I.

(c) Compute the angular velocity ${}^{B}\boldsymbol{\omega}_{IB}$ w.r.t. the basis B.

(d) Compare the norm of the representations of the angular velocities calculated in the previous two exercises.

(e) Write the relation between the time derivative of the Tait–Bryan angles and the angular velocities ${}^{B}\boldsymbol{\omega}_{IB}$ in the form ${}^{B}\boldsymbol{\omega}_{IB} = \boldsymbol{F}(\boldsymbol{p})\dot{\boldsymbol{p}}$ with $\boldsymbol{F}(\boldsymbol{p}) \in \mathbb{R}^{3,3}$. For which angles is this relation invertible? Are Tait–Bryan angles suitable for the description of small rotations?

(f) For the case where $\boldsymbol{F}(\boldsymbol{p})$ is invertible, derive the kinematic equation of the form $\dot{\boldsymbol{p}} = \boldsymbol{H}(\boldsymbol{p})\,{}^{B}\boldsymbol{\omega}_{IB}$.

(g) Linearize the transformation matrix ${}^{BI}\boldsymbol{T}$ by neglecting terms of second and higher order in the angles.

(h) Compute the representations of the angular velocity $\boldsymbol{\omega}_{IB}$ w.r.t. the bases I and B for small angles, respectively.

1.4 Euler angles

The rotation of a free rigid body can be described by Euler angels $\boldsymbol{p}^T = (\psi, \vartheta, \varphi)$. These parametrize the body fixed system B by three successive elementary rotations $I \to A \to C \to B$, starting from the inertial system I. The corresponding transformation matrices are

$$ {}^{AI}\boldsymbol{T} = \begin{pmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} {}^{CA}\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\vartheta) & \sin(\vartheta) \\ 0 & -\sin(\vartheta) & \cos(\vartheta) \end{pmatrix} {}^{BC}\boldsymbol{T} = \begin{pmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix} $$

(a) Sketch the relative rotation of the coordinate systems I, A, C, B. Which axes remain invariant under which rotation?

(b) Compute the angular velocity ${}^{B}\boldsymbol{\omega}_{IB}$ w.r.t. the basis B.

(c) Write the relation between the time derivative of the Euler angles and the angular velocities ${}^{B}\boldsymbol{\omega}_{IB}$ in the form ${}^{B}\boldsymbol{\omega}_{IB} = \boldsymbol{F}(\boldsymbol{p})\dot{\boldsymbol{p}}$ with $\boldsymbol{F}(\boldsymbol{p}) \in \mathbb{R}^{3,3}$. For which angles is this relation invertible? Are Euler angles suitable for the description of small rotations?

(d) For the case where $\boldsymbol{F}(\boldsymbol{p})$ is invertible, derive the kinematic equation of the form $\dot{\boldsymbol{p}} = \boldsymbol{H}(\boldsymbol{p})\,{}^{B}\boldsymbol{\omega}_{IB}$.

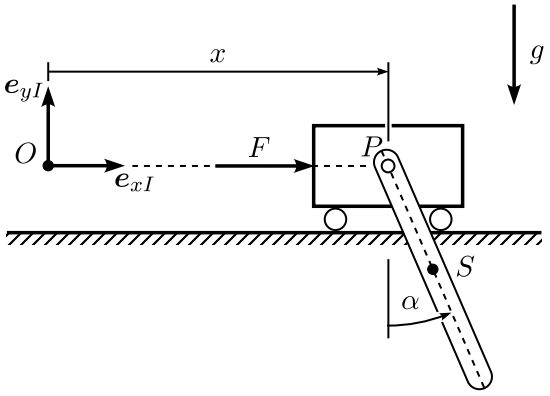
(e) Linearize the transformation matrix ${}^{BI}\boldsymbol{T}$ by neglecting terms of second and higher order in the angles.

(f) Compute the representation of the angular velocity $\boldsymbol{\omega}_{IB}$ w.r.t. the basis B for small angles.

2 Projected Newton–Euler equations

2.1 Pendulum on a cart

Consider a cart (body 1) with mass m_1 and center of mass P, which can move along the $\boldsymbol{e}_{xI}$ axis only. The position of P is measured by the horizontal coordinate x. Moreover, a force F having a horizontal line of action going through P acts on the cart. A rod (body 2) of length L is mounted to the cart via a revolute joint in P. The distance of the center of mass S of the rod w.r.t. P is $L/2$. The rotation of the rod w.r.t. the vertical axis is described by the angle α. The rod's mass is m_2 and its rotational inertia is θ_S. The system is subjected to gravity with gravitational acceleration $\boldsymbol{g} = -g\,\boldsymbol{e}_{yI}$. The pendulum on a cart is described by natural coordinates using $\boldsymbol{q}^T = (x, \alpha)$.



(a) Compute the representations w.r.t. the basis I of the accelerations of P and S, respectively.

(b) Calculate the representation w.r.t. the basis I of the angular accelerations $\boldsymbol{\Psi}_1$ and $\boldsymbol{\Psi}_2$ of both bodies.

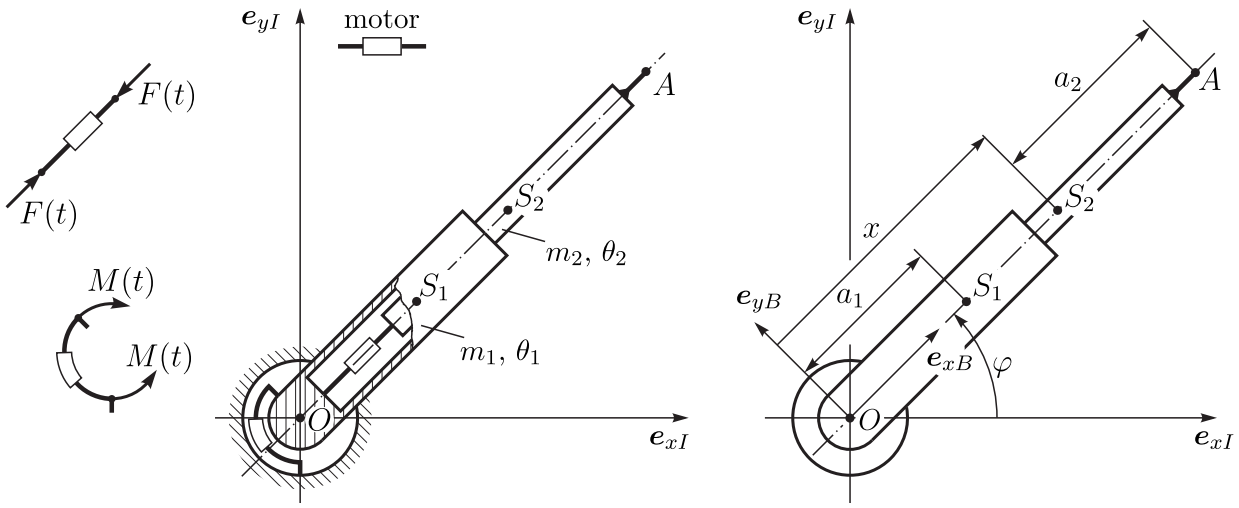
(c) With the help of a free-body diagram and the Newton–Euler equations, compute the equations of motion of the system.

(d) Compute the representations w.r.t. the basis I of the Jacobians of the points P and S, as well as of the Jacobians of the rotation of the two bodies.

(e) Use the projected Newton–Euler equations to compute the equations of motion of the system.

2.2 Robot arm

Let us consider the dynamics of the robot arm from exercise 1.1. The first arm (mass m_1, rotational inertia θ_1) is actuated by a motor that exerts a torque $M(t)$ relative to the basis. The sliding joint that connects the second arm (mass m_2, moment of inertia θ_2) is actuated with a linear motor that exerts a force $F(t)$ between the first and second arm.



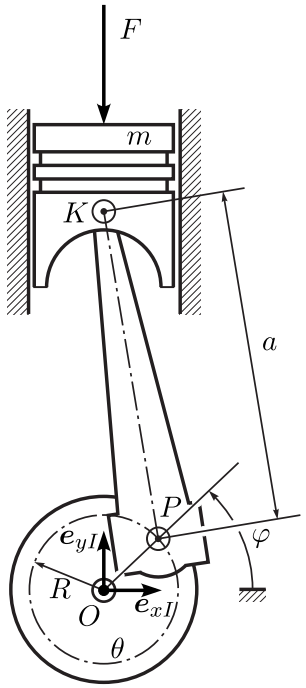
(a) Compute the representations ${}^{B}\mathbf{J}_{S_1}$ and ${}^{B}\mathbf{J}_{S_2}$ of the Jacobians of S_1 and S_2 w.r.t. the basis B, respectively.

(b) Calculate the representations ${}^{B}\mathbf{J}_{R_1}$ and ${}^{B}\mathbf{J}_{R_2}$ of the rotational Jacobians of both bodies w.r.t. the basis B.

(c) Derive the equations of motion using the projected Newton–Euler equations.

2.3 Slider crank mechanism

The slider crank mechanism in an engine is studied. The system is composed by a crank shaft with rotational inertia θ, which is fixed at the origin and can rotate around the $\boldsymbol{e}_{zI}$ axis by the angle φ. At a distance R from O, a massless connecting rod is connected to the crank shaft in the point P via a revolute joint. The connecting rod of length a is finally connected to the piston in the point K via a revolute joint. The piston has mass m and is linearly guided such that it can only move along the $\boldsymbol{e}_{yI}$ axis, i.e., $\boldsymbol{r}_{OK} \parallel \boldsymbol{e}_{yI}$. Due to the combustion, a force $\boldsymbol{F} = -F\boldsymbol{e}_{yI}$ is acting on the piston.



(a) Compute the representations ${}^{I}\boldsymbol{\Omega}_c$ and ${}^{I}\boldsymbol{v}_P$ of the angular velocity of the crank shaft and of the velocity of the point P.

(b) Calculate the representations ${}^{I}\boldsymbol{\Omega}_r$ and ${}^{I}\boldsymbol{v}_K$ of the angular velocity of the connection rod and of the velocity of the point K.

(c) Derive the representation of the Jacobian of the rotation of the crank shaft as well as the representation of the Jacobian of the point K w.r.t. the basis I. Moreover, compute the representations of the time derivative of the angular momentum $\boldsymbol{L}_O$ of the crank shaft and of the linear momentum $\boldsymbol{p}_p$ of the piston.

(d) Derive the equation of motion of the system using the projected Newton–Euler equations.

3 Python and VS Code

We use Python as a programming language as it is widespread, open-source and relatively easy to learn. We suggest to use Visual Studio Code (VS Code) as text editor for coding, however, you are free to use any editor of your choice.

3.1 Installation

(a) Download and install Python from `https://www.python.org/downloads/`. It has been tested with Python 3.10, but newer versions should work too. Alternatively you can use a package manager to install Python (recommended on Mac/Linux). On Windows, make sure that Python is on your system's path.

(b) Check that Python is properly installed by typing `python` in a terminal, which should start the Python application. *Note*: depending on the installation, e.g. on Mac, the Python command is `python3`.

(c) Download and install VS Code `https://code.visualstudio.com/`.

(d) Start VS Code and install the Python extension called "Python" (provided by Microsoft). For help, consult `https://code.visualstudio.com/docs/languages/python`.

3.2 Setup and workflow of programming exercises

src: `Ex3_2`

For each programming exercise, a folder containing the needed files is provided on StudOn. The name of the folder is indicated as *src* in the exercise, see for example "src: `Ex3_2`" above. The present exercise walks you through the download and setup process, which will be the same for all programming exercises.

(a) Download the folder `Ex3_2` from StudOn.

(b) Open the downloaded folder in VS Code.

(c) Start a new terminal in VS Code (Terminal → New Terminal). The path should point to the downloaded folder now. Otherwise, navigate to the downloaded folder using the `cd` command.

(d) Every exercise will be solved within a virtual environment such that installed packages do not interfere between different exercises. To create the virtual environment named `myvenv`, execute

```
python -m venv myvenv
```

You might have to replace the `python` command by `python3`, see exercise 3.1 (b).

(e) By executing the `activate` script in the `bin` folder of your virtual environment, you can activate it. E.g. on Mac/Linux

```
source ./myvenv/bin/activate
```

and on Windows just `./myvenv/bin/activate`.

(f) Within your virtual environment, you now have pure python without any packages installed. For every exercise, you will find a `setup.py` file in the source folder. This file specifies all required packages for the exercise. To install the packages, simply run

```
pip install -e .
```

which will install the source folder in editable mode.

(g) Use `pip list` to see all installed packages. Check if `PyCMD 3.2` is installed. If this is the case, you are set up to solve exercise 3.2.

(h) Use VS Code to run `hello_world.py` in the Terminal.

Remark: For later exercises, a folder with a solution to the exercise is also provided on StudOn. If you want to run the code inside a solution folder, you need to install that folder too, i.e., you need to do all the steps of this exercise also for the solution folder.

3.3 Introduction to Python

src: `Ex3_3`

This exercises gives a brief introduction to the usage of Python as well as of the packages NumPy (linear algebra) and Matplotlib (plotting).

(a) Download and install the `Ex3_3` folder. (This step will not be mentioned in later exercises!)

(b) Create a new file called `intro_to_python.py`. Solve the remainder of this exercise in that file.

(c) Consider the function

$$f(n) = \sum_{i=1}^{n} i\,,$$

which sums the numbers from 1 to n. Compute $f(100)$ using a "for loop".

(d) Implement the function f and print the value $f(75)$.

(e) Import NumPy as `np`. Use the functions `np.arange` and `np.sum` to implement f using both the normal and the lambda function syntax.

(f) From Matplotlib, import Pyplot as `plt` and plot f as a function of n for $1 \leq n \leq 20$ using the function `plt.plot`.

(g) Define

$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 3 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix}$$

as NumPy arrays. Compute and print

$$\boldsymbol{Ab}, \quad \boldsymbol{b}^T\boldsymbol{A}, \quad \boldsymbol{AB}, \quad \boldsymbol{A}^T\boldsymbol{Bb}, \quad \boldsymbol{A}^{-1}\boldsymbol{b}.$$

4 Solving ordinary differential equations

4.1 Explicit Euler method

src: `Ex4_1`

The pendulum on a cart system, see exercise 2.1, is described in natural coordinates using $\boldsymbol{q} = (x, \alpha)^T$, i.e., $\boldsymbol{u} = \dot{\boldsymbol{q}}$. The mass matrix and the total generalized forces of the system are

$$\boldsymbol{M}(\boldsymbol{q}) = \begin{pmatrix} m_1 + m_2 & \frac{1}{2} m_2 L \cos\alpha \\ \frac{1}{2} m_2 L \cos\alpha & \theta_s + \frac{1}{4} m_2 L^2 \end{pmatrix}, \qquad \boldsymbol{h}(t, \boldsymbol{q}, \boldsymbol{u}) = \begin{pmatrix} \frac{1}{2} m_2 L \dot{\alpha}^2 \sin\alpha + F(t) \\ -\frac{1}{2} m_2 g L \cos\alpha \end{pmatrix}.$$

It is the goal of this exercise to simulate the pendulum on a cart system using the explicit Euler method. Do all the subsequent programming steps in the file `pendulum_on_cart.py` by filling the gaps.

(a) Implement the mass matrix and the generalized forces as python functions.

(b) Familiarize yourself with the initialization of the solution `q` used to store the simulated generalized coordinates, i.e., `q` is the list $(q_0, q_1, \dots)$ such that `q[k]` $= q_k$. Initialize the solution `u` for the storage of the simulated generalized velocities in the analogous way. Moreover, save the initial conditions q_0 and u_0 in the solution lists.

(c) Show that the explicit Euler method for a mechanical system described in natural coordinates takes the form

$$\begin{aligned} \boldsymbol{q}_{k+1} &= \boldsymbol{q}_k + \Delta t\, \boldsymbol{u}_k \\ \boldsymbol{u}_{k+1} &= \boldsymbol{u}_k + \Delta t\, \boldsymbol{M}^{-1}(t_k, \boldsymbol{q}_k)\, \boldsymbol{h}(t_k, \boldsymbol{q}_k, \boldsymbol{u}_k). \end{aligned}$$

(d) Use a for-loop to implement the above Euler method.

(e) Plot the generalized coordinates over time.

(f) Remove the `exit()` command to show the animation of the system.

4.2 PyCMD - modular simulation structure

src: `Ex4_2`

The goal of this exercise is to implement the simulation done in exercise 4.1 in a modular way. This means that the pendulum on a cart system is implemented as a "system"-class having a standardized structure. Moreover, the explicit Euler method is implemented as a "solver"-class, which can simulate any mechanical system defined as a "system"-class.

(a) Familiarize yourself with the main file for the simulation of the pendulum on a cart system, i.e., with the file `simulations/pendulum_on_cart.py`.

(b) Complete the implementation of the pendulum on a cart system class, which can be found in `PyCMD/system/PendulumOnCart.py`

(c) Complete the implementation of the explicit Euler method, which can be found in `PyCMD/solver/EulerExplicit.py`

(d) Import the implicit Euler method from `PyCMD/solver/EulerImplicit.py` and simulate the pendulum on a cart system with that.

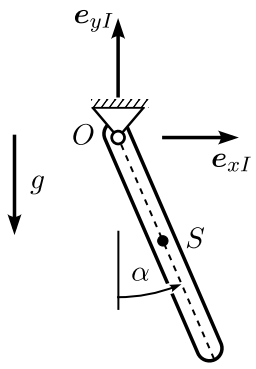
(e) Create a new file called `pendulum.py` in the `simulations` folder. In that file, simulate the pendulum from exercise 4.3 using the implicit Euler scheme. The pendulum system class is already implemented. Use $m = 1$ and $L = 1$.

4.3 Symplectic Euler method

src: `Ex4_3`

In this exercise, a planar pendulum of length L and mass m is simulated using different Euler methods. The equation of motion of the pendulum is

$$\frac{mL^2}{3}\ddot{\alpha} + \frac{mgL}{2}\sin\alpha = 0$$



Even though all Euler method have convergence order one, they qualitatively perform very differently. This is analyzed by studying the energy

$$E(\alpha, \dot{\alpha}) = \frac{mL^2}{6}\dot{\alpha}^2 + \frac{mgL}{2}(1 - \cos\alpha)$$

of the pendulum.

(a) Run `simulation/pendulum.py` to simulate the pendulum using the explicit Euler method. What energy behavior do you observe? Is this behavior physical?

(b) Additionally simulate the pendulum using the implicit Euler method. Does this method have a better energy behavior?

The symplectic Euler method is a combination of the implicit and explicit Euler method, i.e., it uses the implicit Euler method for the kinematic equations and the explicit Euler method for the equations of motion. Specifically,

$$\begin{aligned}\boldsymbol{q}_{k+1} &= \boldsymbol{q}_k + \Delta t\, \boldsymbol{u}_{k+1}\\ \boldsymbol{u}_{k+1} &= \boldsymbol{u}_k + \Delta t\, \boldsymbol{M}^{-1}(t_k, \boldsymbol{q}_k)\, \boldsymbol{h}(t_k, \boldsymbol{q}_k, \boldsymbol{u}_k)\,.\end{aligned}$$

(c) Save a copy of `PyCMD/solvers/EulerExplicit.py` as `PyCMD/solvers/EulerSymplectic.py`. Modify the just created file to obtain an implementation of the symplectic Euler method.

(d) Simulate the pendulum using the symplectic Euler method and compare its energy behavior to the other Euler methods.

4.4 Implicit trapezoidal rule and Newton–Raphson method

src: `Ex4_4`

Starting point of this exercise is the simulation of the pendulum from exercise 4.3 using the implicit Euler method, see `simulation/pendulum.py`.

(a) Implement the implicit trapezoidal rule

$$\begin{aligned}\boldsymbol{q}_{k+1} &= \boldsymbol{q}_k + \frac{\Delta t}{2}\left(\boldsymbol{u}_k + \boldsymbol{u}_{k+1}\right)\\ \boldsymbol{u}_{k+1} &= \boldsymbol{u}_k + \frac{\Delta t}{2}\Big(\boldsymbol{M}^{-1}(t_k, \boldsymbol{q}_k)\, \boldsymbol{h}(t_k, \boldsymbol{q}_k, \boldsymbol{u}_k) + \boldsymbol{M}^{-1}(t_{k+1}, \boldsymbol{q}_{k+1})\, \boldsymbol{h}(t_{k+1}, \boldsymbol{q}_{k+1}, \boldsymbol{u}_{k+1})\Big)\,.\end{aligned}$$

and use it to simulate the pendulum. Compare the results to the simulation with the implicit Euler method.

Hint: The code structure of implicit methods is always the same. Moreover, it can be useful to introduce an auxiliary variable

$$\bar{\boldsymbol{u}}_k = \boldsymbol{u}_k + \frac{\Delta t}{2} \boldsymbol{M}^{-1}(t_k, \boldsymbol{q}_k)\, \boldsymbol{h}(t_k, \boldsymbol{q}_k, \boldsymbol{u}_k)$$

such that the discretization of the equations of motion in residual form is

$$\boldsymbol{M}(t_{k+1}, \boldsymbol{q}_{k+1})\big(\boldsymbol{u}_{k+1} - \bar{\boldsymbol{u}}_k\big) - \frac{\Delta t}{2} \boldsymbol{h}(t_{k+1}, \boldsymbol{q}_{k+1}, \boldsymbol{u}_{k+1}) = 0\,.$$

(b) Fill the gaps of `PyCMD/solvers/NewtonRaphson.py` to implement the Newton–Raphson method.

(c) Use your Newton–Raphson method instead of the `root` function to solve the time step in both solvers used above.

5 Model assembly and code structure

5.1 System class and model assembly

src: `Ex5_1`

The `System`-class is the core object for the modular simulation of mechanical systems. Using the command `System.add(block)`, a building `block` can be added to the system. These building blocks are themselves instances of classes representing rigid bodies, joints, actuators, etc. Every block provides its contribution to the system's quantities, e.g., provides the local mass matrix `block.M` or the local force contribution `block.h`. From all the local contributions of its building blocks, the system calculates the global quantities, e.g., the global mass matrix `System.M` or the global force vector `System.h`.

(a) Open the system-class `PyCMD/system/System.py` and familiarize yourself with the `assemble` method.

(b) Explain the code of the `assemble` method in your own words.

(c) Use an example to explain how the `assemble` method handles building blocks that have positional degrees of freedom.

The contributions of the building block j to the system's quantities depend only on local coordinates $\boldsymbol{q}_j$ and $\boldsymbol{u}_j$, which are a subset of the global coordinates $\boldsymbol{q}$ and $\boldsymbol{u}$. The local and global quantities are linked by introducing respective connectivity matrices $\boldsymbol{C}_j^q$ and $\boldsymbol{C}_j^u$.

(d) Derive the relation between $\dot{\boldsymbol{q}}_j$ and $\dot{\boldsymbol{q}}$. Moreover, show that

$$\frac{\partial \boldsymbol{q}_j}{\partial \boldsymbol{q}} = \frac{\partial \dot{\boldsymbol{q}}_j}{\partial \dot{\boldsymbol{q}}} = \boldsymbol{C}_j^q \,.$$

(e) Implement the method `System.q_dot(t,q,u)` that assembles the different contributions $\dot{\boldsymbol{q}}_j(t, \boldsymbol{q}_j, \boldsymbol{u}_j)$ into the global vector $\dot{\boldsymbol{q}}(t, \boldsymbol{q}, \boldsymbol{u})$, where $\dot{\boldsymbol{q}}_j$ is represented by `block.q_dot(t,q_j,u_j)`.

The computation of the global mass matrix and global force vectors from local contributions is given by

$$\boldsymbol{M}(t,\boldsymbol{q}) = \sum_j (\boldsymbol{C}_j^u)^T \boldsymbol{M}_j^{\mathrm{loc}}(t,\boldsymbol{q}_j)\boldsymbol{C}_j^u \quad \text{and} \quad \boldsymbol{f}(t,\boldsymbol{q}) = \sum_j (\boldsymbol{C}_j^u)^T \boldsymbol{f}_j^{\mathrm{loc}}(t,\boldsymbol{q}_j,\boldsymbol{u}_j)$$

(f) Given that the local mass matrix and the local gyroscopic force contribution of a building block are `block.M(t,q_j)` and `block.f_gyr(t,q_j,u_j)`, implement the global mass matrix `System.M(t,q)` and the global gyroscopic force `System.f_gyr(t,q,u)`.

(g) Run the simulation of the pendulum on a cart (`simulations/pendulum_on_cart.py`) to validate your implementation.

(h) For the implementation of implicit integration methods, such as the implicit Euler method, the quantity $\boldsymbol{A} = \frac{\partial}{\partial \boldsymbol{q}}\big(\boldsymbol{M}(t,\boldsymbol{q})\boldsymbol{u}\big)$ implemented as `System.Mu_q` is needed. The local contribution to $\boldsymbol{A}$ of each building block j is

$$\boldsymbol{A}_j^{\mathrm{loc}} = \frac{\partial}{\partial \boldsymbol{q_j}}\big(\boldsymbol{M}_j^{\mathrm{loc}}(t,\boldsymbol{q}_j)\boldsymbol{u}_j\big)\,,$$

which is implemented as `block.Mu_q`. Explain the implementation of $\boldsymbol{A}$ by deriving the relation between the global quantity $\boldsymbol{A}$ and the local contributions $\boldsymbol{A}_j^{\mathrm{loc}}$ using connectivity matrices.

6 Relative Kinematics

6.1 Joints

When working with relative coordinates, the joints "provide" coordinates $(\boldsymbol{q}_j, \boldsymbol{u}_j)$ to the system. The kinematics of the joint is described by the relative displacement ${}^{P_1}\boldsymbol{r}_{P_1P_2}(t, \boldsymbol{q}_j)$ and the relative rotation described by ${}^{P_1P_2}\boldsymbol{T}(t, \boldsymbol{q}_j)$, as well as the respective velocities, accelerations and Jacobians.

Do the following steps for the joints specified below.

(a) Sketch the kinematics of the joint.

(b) Compute all kinematic quantities describing the joint.

(c) State the kinematic equation of the body.

Linear guidance A linear guidance displaces the point P_2 w.r.t. P_1 by a distance x along the $\boldsymbol{e}_{xP_1}$-axis. The linear guidance does not allow relative rotation between the predecessor and successor body. The linear guidance is described by natural coordinates with $\boldsymbol{q}_j = (x)$.

Revolute joint about z-axis This joint does not displace the point P_2 w.r.t. P_1. The relative rotation between the P_2 and P_1 frame is an elementary rotation with angle φ around the $\boldsymbol{e}_{zP_1}$-axis. The revolute joint is described by natural coordinates with $\boldsymbol{q}_j = (\varphi)$.

Spherical joint This joint does not displace the point P_2 w.r.t. P_1. The relative rotation between the P_2 and P_1 frame is an arbitrary rotation described by the rotation parametrization $\boldsymbol{p}$, i.e., ${}^{P_1P_2}\boldsymbol{T}(t, \boldsymbol{p})$. Choices for such a rotation parametrization could be Euler angles or Tait–Bryan angles, see Exercises 1.3 and 1.4. In general, the angular velocity can be related to the time derivative of the rotation parametrization by

$$ {}^{P_1}\boldsymbol{\omega}_{P_1P_2} = \boldsymbol{F}(\boldsymbol{p})\dot{\boldsymbol{p}} \quad \text{and} \quad \dot{\boldsymbol{p}} = \boldsymbol{H}(\boldsymbol{p})\, {}^{P_1}\boldsymbol{\omega}_{P_1P_2} $$

The spherical joint is described by the coordinates $\boldsymbol{q}_j = \boldsymbol{p}$ and $\boldsymbol{u}_j = {}^{P_1}\boldsymbol{\omega}_{P_1P_2}$.

6.2 Implementation of joints

src: `Ex6_2`

In this exercise, the revolute joint and a spherical joint are implemented in `PyCMD`. Both require the implementation of elementary rotations `IB_T_elem`, which you can import from `PyCMD.math`.

(a) Familiarize yourself with the implementation of the linear guidance.

(b) Implement the revolute joint from Exercise 6.1. You can test the revolute joint by running the simulation of the pendulum on cart `simulations/pendulum_on_cart.py`.

(c) Use the Tait–Bryan angles from Exercise 1.3 to implement the spherical joint from Exercise 6.1.

(d) Model the pendulum on a cart system using a spherical joint instead of a revolute joint.

(e) Which part of the implementation of the spherical joint would change, if instead of the Tait–Bryan angles a different rotation parametrization would be used?

6.3 Implementation of the rigid body

src: `Ex6_3`

This exercise concerns the implementation of the rigid body described using relative coordinates, i.e., the implementation of the class `RigidBodyRel`.

(a) Explain the main idea behind the description of a rigid body in relative coordinates. Why is this idea limited to mechanical systems with tree-structure?

(b) Implement the transformation matrix ${}^{IB}\boldsymbol{T}(t, \boldsymbol{q})$ as `RigidBodyRel.IB_T(t, q)`. Note, that $\boldsymbol{q}$ and `q` correspond to the local generalized coordinates of the body, i.e., they correspond to $\boldsymbol{q}_2$ of the lecture notes.

(c) Complete the implementation `RigidBodyRel.B_Omega(t, q, u)` of the representation ${}^{B}\boldsymbol{\Omega}(t, \boldsymbol{q}, \boldsymbol{u})$ of the angular velocity of the body w.r.t. the body fixed frame B.

(d) Implement the representation ${}^{I}\boldsymbol{r}_{OP}(t, \boldsymbol{q})$ of the position vector of the point P on the body as `RigidBodyRel.r_OP(t, q, B_r_SP)`. Herein, `B_r_SP` is the position vector ${}^{B}\boldsymbol{r}_{SP}$ of P w.r.t. the center of mass S of the body represented in the body fixed frame B.

(e) Explain why the additional argument `B_r_SP` is needed in the actual implementation. What changes to the code would be necessary if we would implement ${}^{I}\boldsymbol{r}_{OS}(t, \boldsymbol{q})$ instead of the position vector of an arbitrary point P?

7 One-dimensional force interactions

7.1 Force with constant direction

src: `Ex7_1`

Consider a time dependent force with a constant direction described by the unit vector $\boldsymbol{n}$, i.e., a force $\boldsymbol{F}(t) = F(t)\,\boldsymbol{n}$. Hereby, the strength of the force $F(t)$ is a given function of time. It is assumed, that the force acts on a point P of some body of the system.

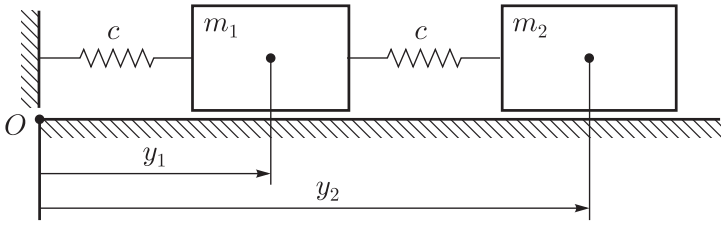
(a) Derive the generalized force contribution $\boldsymbol{f}$ as well as the generalized force direction $\boldsymbol{w}$ of $\boldsymbol{F}$.

(b) Show that the force is a potential force with potential $V(t, \boldsymbol{q}) = -\boldsymbol{r}_{OP}^T(t, \boldsymbol{q})\boldsymbol{F}(t)$.

(c) Complete the implementation of the force as the class `Force`.

7.2 Two-point interaction

src: `Ex7_2`



In this exercise, we use the two-point interaction to simulate the shown two-mass oscillator. The oscillator is composed of two rigid bodies with respective masses m_1 and m_2. The rigid bodies are described by the coordinates y_1 and y_2 describing the horizontal position of the respective center of mass of the bodies. The first body is attached to the origin by a spring with stiffness c and undeformed length l_0. An identical spring is used to connect the second body to the first. It can be assumed, that the springs are connected to the center of mass of each body.

(a) Familiarize yourself with the implementation of the `TwoPointInteraction` and the force law `LinearSpring`.

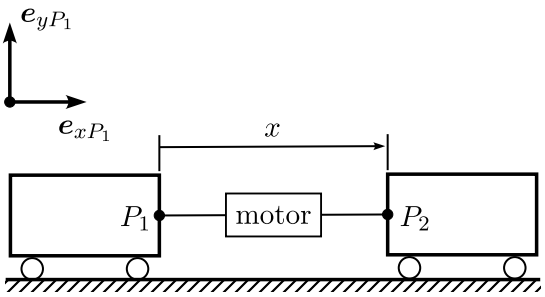
(b) Complete the simulation file `two_mass_oscillator.py` and simulate the two-mass oscillator.

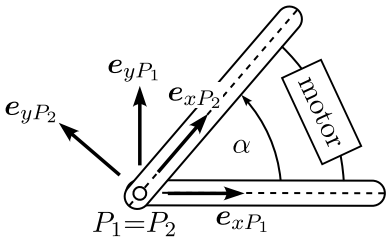
(c) Replace the linear springs by force elements composed of a linear spring and a linear damper acting in parallel, respectively.

7.3 Actuated joints

src: `Ex7_3`

Consider a linear guidance and a revolute joint which are each actuated by a motor. In both cases, the motor can be treated as one-dimensional force interaction with force law $\lambda = \tau$, where τ is the given scalar force and torque, respectively.





(a) Draw a free-body diagram for both cases.

(b) Show that in both cases the force contribution of the motor is $\boldsymbol{f}^\tau = (\boldsymbol{C}_j^u)^T \tau$, where $\boldsymbol{C}_j^u$ is the connectivity matrix of the respective joint. Moreover, derive the generalized force direction $\boldsymbol{w}_\tau$.

(c) Explain the changes that have been made to the `System` class in order to assemble the global matrix of generalized force directions $\boldsymbol{W}_\tau$ of the motors present in the system.

(d) Familiarize yourself with the class `LinearGuidanceRelActuated`, which implements the actuated linear guidance.

(e) Implement the actuated revolute joint as `RevoluteJointRelActuated`.

(f) Use these building blocks to simulate the robot arm from exercise 2.2

8 Parametrization of rotations

8.1 Spherical joint using Euler parameters

src: `Ex8_1`

In order to have a singularity free spherical joint, Euler parameters are used to parametrize the rotation.

(a) Implement the spherical joint with Euler parameters, i.e., solve Exercise 6.2 (c) using Euler parameters instead of Tait–Bryan angles.
Hint: From `PyCMD.math` you can import `ax2skew`, which performs the 'tilde' operation. Also, its Jacobian `ax2skew_a` will be helpful.

(b) Simulate the pendulum on a cart system with this spherical joint.
Hint: You have also to adapt the explicit Euler scheme such that it can also treat systems which are not formulated in natural coordinates.

(c) Plot and analyze the norm $||\boldsymbol{p}||$ of the Euler parameters over time. What do you observe?

(d) Does the drift of $||\boldsymbol{p}||$ improve if you decrease the time step?

(e) Qualitatively explain the drift of $||\boldsymbol{p}||$?

There are several strategies to fix the drift of $||\boldsymbol{p}||$. A straightforward solution is to normalize $\boldsymbol{p}$ after every step of the numerical integration method. For that, the `step_callback` function can be used. It is called by every solver at the end of the time step. The `System`-class assembles this function, such that if a building block implements the `step_callback` function, it is called at the end of each time step.

(f) Implement the `step_callback` function for the spherical joint to normalize the Euler parameters after each time step.

(g) Redo the simulation and replot the norm $||\boldsymbol{p}||$ of the Euler parameters over time to check the implementation.

9 Inverse kinematics and inverse dynamics

9.1 Trajectory tracking: robot arm

src: `Ex9_1`

Consider the robot arm of Exercise 2.2. In this exercise, the actuator forces $\boldsymbol{\tau}(t) = (M(t), F(t))^T$ are computed such that the point A of the robot arm follows a planned (given) trajectory.

(a) Familiarize yourself with the file `robot_arm_trajectory_tracking.py`.

(b) There are two planned trajectories, a line and a circle. For the given initial conditions, sketch the planned trajectories.

(c) After initialization, the first step of the inverse kinematics computations is the computation of the initial velocity `u[0]` of the system. Explain how this is done. Why do we need that step, instead of choosing an initial velocity as usual?

(d) Implement the inverse kinematics and inverse dynamics algorithm.

(e) Run the code to check your results.

(f) In your inverse kinematics, compute the acceleration by the simpler approximation

$$\dot{\boldsymbol{u}}_{k+1} = \frac{\boldsymbol{u}_{k+1} - \boldsymbol{u}_k}{\Delta t}\,.$$

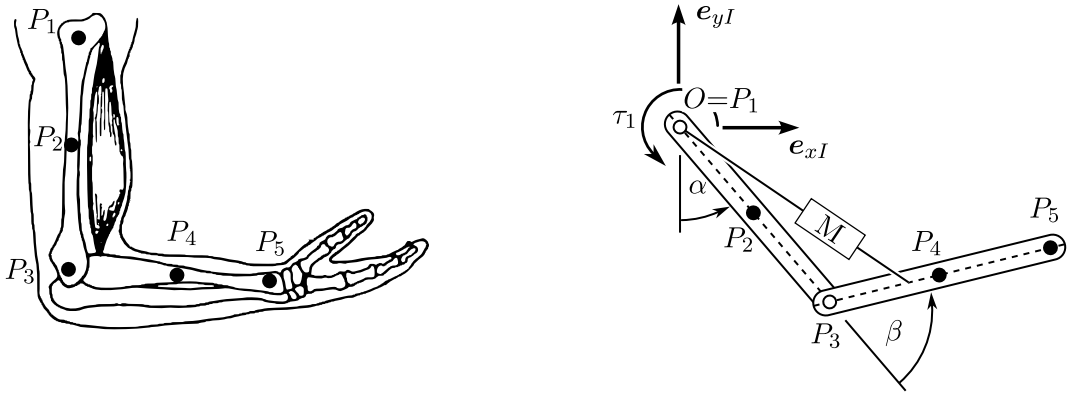
Do you observe any difference?

(g) Track the circle with an angular velocity `omega=3`. What do you observe? Can you explain this behavior? Which acceleration update works better?

9.2 Motion fitting: human arm

src: `Ex9_2`

During an experiment, the motion of the arm is measured by tracking the points P_i (i=1,...,5) using a camera system. The measurement results are a discrete data set of time instants t_k together with the values ${}^I\boldsymbol{r}^g_{OP_i}(t_k)$. The collection of time instants and marker positions are respectively saved in the files `t_g.npy` and `r_OPi_g_noisy.npy`. To "eliminate" the measurement noise, the data set has been filtered and saved as `r_OPi_g.npy`. All data for this exercise lies in the folder `simulations/motion_fitting`.



A simple mechanical arm model and an inverse kinematics and inverse dynamics analysis is used to estimate the muscle force of the biceps. The mechanical model consists of two bodies modeling the upper arm (body 1) and the lower arm (body 2). Both are assumed to be homogeneous rods of mass m_1

and m_2, and length L_1 and L_2, respectively, The upper arm is attached to the origin via a revolute joint modeling the shoulder joint. The elbow is modeled as a revolute joint connecting the upper and lower arm. Consequently, the model is described in natural coordinates with $\boldsymbol{q}^T = (\alpha, \beta)$. In the model, the markers are assumed to lie at the extreme points of the rods (P_1, P_3, P_5) and at the midpoints (center of mass) of the rods (P_2, P_4). The muscle forces moving the shoulder joint are modeled as a moment τ_1 acting on the upper arm. The biceps force τ_2 is modeled as a two-point interaction between the shoulder joint and the point on the lower arm, whose distance to the elbow is $L_2/20$.

(a) Familiarize yourself with the file `human_arm_motion_fitting.py`.

(b) Explain, how the initial configuration $\boldsymbol{q}_0$ is computed.

(c) Use a for-loop to find all configurations $\boldsymbol{q}_{k+1}$ for the predefined time instants t_{k+1} (k=0,1,...).

(d) Solve the implicit Euler scheme for $\boldsymbol{u}_{k+1}$. Exploit, that the model is described in natural coordinates.

(e) Implement the result from (d) to compute the velocities of the system.

(f) Implement the inverse dynamics algorithm to find the forces $\boldsymbol{\tau}_{k+1}$.

(g) Plot the biceps force and the shoulder moment over time.

(h) Do the same analysis using the "noisy" data. Can you explain the result?

10 Implicit constraints

10.1 Joints

In this exercise, we consider the joints described in exercise 6.1. The (local) coordinates involved in describing these joints using implicit constraints are $\boldsymbol{q}_j = (\boldsymbol{q}_1, \boldsymbol{q}_2)$ and $\boldsymbol{u}_j = (\boldsymbol{u}_1, \boldsymbol{u}_2)$, where $(\boldsymbol{q}_1, \boldsymbol{u}_1)$ and $(\boldsymbol{q}_2, \boldsymbol{u}_2)$ describe the local coordinates of the two bodies connected by the joint.

(a) Formulate the position level constraints describing the spherical joint.

(b) Derive the generalized force direction for the spherical joint.

(c) Which velocity level constraints have to be added to the spherical joint in order to describe the revolute joint?

(d) Formulate the constraints describing the revolute joints as a set of velocity level constraints.

(e) Derive the generalized force direction for the revolute joint.

(f) Describe the revolute joint using position level constraints only.

(g) Discuss the pros and cons of the different constraint formulations of the revolute joint from a practical point of view.

10.2 Implicit Euler scheme for DAEs

The implicit Euler discretization of a general DAE for the state variables $(\boldsymbol{x}, \boldsymbol{\lambda})$ is given by

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{\lambda}) \\ 0 &= \boldsymbol{c}(t, \boldsymbol{x}) \end{aligned} \quad \Longrightarrow \quad \begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \Delta t\, \boldsymbol{f}(t_{k+1}, \boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_{k+1}) \\ 0 &= \boldsymbol{c}(t_{k+1}, \boldsymbol{x}_{k+1}) \end{aligned}$$

It can be used to solve the equations of motion of a constrained mechanical system, which have the form

$$\begin{aligned} \boldsymbol{M}\dot{\boldsymbol{u}} - \boldsymbol{h} &= \boldsymbol{W}_g \boldsymbol{\lambda}_g + \boldsymbol{W}_\gamma \boldsymbol{\lambda}_\gamma \\ \dot{\boldsymbol{q}} &= \boldsymbol{B}\boldsymbol{u} + \boldsymbol{\beta} \\ \boldsymbol{g}(t, \boldsymbol{q}) &= 0 \\ \boldsymbol{\gamma}(t, \boldsymbol{q}, \boldsymbol{u}) &= 0 \end{aligned}$$

(a) Relate the equations of motion to the general form of a DAE by stating $\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{f}$ and $\boldsymbol{c}$ in terms of mechanical quantities.

(b) Discretize the equations of motion of the mechanical system by using the implicit Euler scheme and write it in residual form.