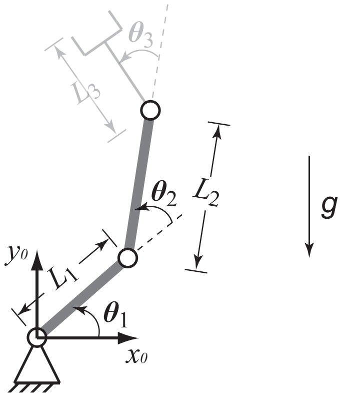
Technische Universität München	**Robot Control Laboratory**
Chair of Automatic Control Engineering	3. Assignment
Dr.-Ing. Marion Leibold	
Chair of Information-Oriented Control	
Dr.-Ing. Stefan Sosnowski	

Exercise 1: Inverse Dynamics

In the following a planar robot with two revolute joints θ_1, θ_2 and two rigid segments $L_1 = L_2 = L = 1\,\text{m}$ is used. The segments have a homogeneous mass distribution and a quadratic cross section with side length $w = 0.05\,\text{m}$. The density of the material is given by $\rho = 7806\,\text{kg/m}^3$. The revolute joints are assumed to be ideal and the rigid body segments are approximated by a point mass.



The robot should move with a constant cartesian velocity $\dot{\boldsymbol{w}}_0$. The task is to compute the torque $\boldsymbol{\tau}$ that is necessary for the movement. Therefore you should implement two routines with the following input and output parameters:

- Joint angles $\boldsymbol{q}(t_i) = \boldsymbol{q}_i = (\theta_{1i}, \theta_{2i})^T$ [rad, rad]
- Joint angular velocity $\dot{\boldsymbol{q}}(t_i) = \dot{\boldsymbol{q}}_{\boldsymbol{i}} = (\dot{\theta}_{1i}, \dot{\theta}_{2i})^T$ [rad/s, rad/s]
- Joint angular acceleration $\ddot{\boldsymbol{q}}(t_i) = \ddot{\boldsymbol{q}}_{\boldsymbol{i}} = (\ddot{\theta}_{1i}, \ddot{\theta}_{2i})^T$ [rad/s^2, rad/s^2]
- Cartesian position of the end effector $\boldsymbol{w}(t_i) = \boldsymbol{w}_i = (r_{xi}, r_{yi})^T$ [m, m]
- Constant cartesian velocity of the end effector $\dot{\boldsymbol{w}}(t_0) = \dot{\boldsymbol{w}}_0 = \dot{\boldsymbol{w}}_i = (\dot{r}_x, \dot{r}_y)^T$ [m, m]
- Joint Torques $\boldsymbol{\tau}(t_i) = \boldsymbol{\tau}_i = (\tau_{1i}, \tau_{2i})^T$ [Nm, Nm]
- Time steps of the simulation $\boldsymbol{t} = (t_0, t_1, t_2, \dots, t_n)$ [s]
- Simulation time step $\triangle t$
- Overall simulation time T [s]

The orientation of the end effector will be omitted (compare $\boldsymbol{w}_i$ and $\dot{\boldsymbol{w}}_0$).

a) Compute the joint angles, velocities and accelerations that result from a motion with a constant cartesian velocity $\dot{\boldsymbol{w}}_0$ with the routine **constVel**. Therefore starting from time $t_0 = 0\,\text{s}$ all values for the time step t_i should be computed. Use the explicit Euler scheme for the numerical integration of the joint angles

$$\dot{\boldsymbol{q}} = \boldsymbol{f}(t, \boldsymbol{q}) \rightarrow \boldsymbol{q}_{k+1} = \boldsymbol{q}_k + f(t_k, \boldsymbol{q}_k)\triangle t.$$

and for the joint accelerations the following finite differences scheme

$$\ddot{\boldsymbol{q}}(t_k) = \frac{\dot{\boldsymbol{q}}(t_{k+1}) - \dot{\boldsymbol{q}}(t_k)}{\triangle t}.$$

(10)	$[\boldsymbol{Q}, \dot{\boldsymbol{Q}}, \ddot{\boldsymbol{Q}}, \boldsymbol{W}, \boldsymbol{t}] =$ constVel$(\boldsymbol{q}_0, \dot{\boldsymbol{w}}_0, \triangle t, T)$	MAT
Input:	$\boldsymbol{q}_0$ 2×1 vector	
	$\dot{\boldsymbol{w}}_0$ 2×1 vector	
	$\triangle t$ scalar quantity	
	T scalar quantity	
Output:	$\boldsymbol{Q} = \begin{bmatrix}\boldsymbol{q}_0 & \boldsymbol{q}_1 & \dots & \boldsymbol{q}_n\end{bmatrix}$ 2×(n+1) matrix	
	$\dot{\boldsymbol{Q}} = \begin{bmatrix}\dot{\boldsymbol{q}}_0 & \dot{\boldsymbol{q}}_1 & \dots & \dot{\boldsymbol{q}}_n\end{bmatrix}$ 2×(n+1) matrix	
	$\ddot{\boldsymbol{Q}} = \begin{bmatrix}\dot{\boldsymbol{q}}_0 & \dot{\boldsymbol{q}}_1 & \dots & \dot{\boldsymbol{q}}_n\end{bmatrix}$ 2×(n+1) matrix	
	$\boldsymbol{W} = \begin{bmatrix}\boldsymbol{w}_0 & \boldsymbol{w}_1 & \dots & \boldsymbol{w}_n\end{bmatrix}$ 2×(n+1) matrix	
	$\boldsymbol{t} = \begin{bmatrix}t_0, t_1, t_2, \dots, t_n\end{bmatrix}$ 1×(n+1) matrix	
Allowed routines:	Link(), SerialLink(), fkine(), jacob0()	

Test: Compute the output for

- $\boldsymbol{q}_0 = (45°, 90°)^T$
- $\dot{\boldsymbol{w}}_0 = (0, 0.5)^T$ [m/s, m/s]
- $\triangle t = 0.01$ [s]
- $T = 1$ s

Check,...
... if the final cartesian position computed from the joint angles $\dot{\boldsymbol{q}}_n$ is identical to $\boldsymbol{w}_n$.
... if t_n is equal to the overall simulation time T.
... if the number of columns of $\boldsymbol{Q}$, $\dot{\boldsymbol{Q}}$, $\ddot{\boldsymbol{Q}}$, $\boldsymbol{W}$ and $\boldsymbol{t}$ is identical.
Plot the x and y position of the end effector as well as the joint angular velocities over time. If the data seems incorrect, redo the simulation with a smaller step size and check the results for feasibility. Additionally the movement can be checked using the $\mathbf{plot()}$ method of the *Robotics Toolbox for Matlab* to visualize the robots movement.

b) Implement the simulation of the inverse dynamics of the robot in the routine **invdyn**. Therefore use the solution of tutorial six of the Robotics class and the gravitational constant $g = +9.81$ [m/s^2] ($g \geq 0$).

(10)	$\boldsymbol{\Gamma} =$ invdyn$(\boldsymbol{Q}, \dot{\boldsymbol{Q}}, \ddot{\boldsymbol{Q}})$	MAT
Input:	$\boldsymbol{Q} = \begin{bmatrix}\boldsymbol{q}_0 & \boldsymbol{q}_1 & \dots & \boldsymbol{q}_n\end{bmatrix}$ 2×(n+1) matrix	
	$\dot{\boldsymbol{Q}} = \begin{bmatrix}\dot{\boldsymbol{q}}_0 & \dot{\boldsymbol{q}}_1 & \dots & \dot{\boldsymbol{q}}_n\end{bmatrix}$ 2×(n+1) matrix	
	$\ddot{\boldsymbol{Q}} = \begin{bmatrix}\dot{\boldsymbol{q}}_0 & \dot{\boldsymbol{q}}_1 & \dots & \dot{\boldsymbol{q}}_n\end{bmatrix}$ 2×(n+1) matrix	
Output:	$\boldsymbol{\Gamma} = \begin{bmatrix}\boldsymbol{\tau}_0 & \boldsymbol{\tau}_1 & \dots & \boldsymbol{\tau}_n\end{bmatrix}$ 2×(n+1) matrix	
Allowed routines:	Link(), SerialLink(), fkine(), jacob0()	

Test: Use the computed values from the **constVel** test to test this routine. Compare your results with those of the function $\mathbf{rne()}$ of the *Robotics Toolbox for Matlab*. The output of the function $\mathbf{rne()}$ will be slightly different, due to a different integrator. Compute the maximal error between your results and those of the *Robotics Toolbox*. It should not exceed 0.1 Nm. Plot the torques $\boldsymbol{\tau}_{\mathrm{rne}} = (\tau_1, \tau_2)^T$ [Nm, Nm] of both computations and compare them.

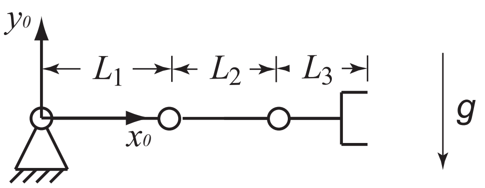
Additional Notes: During the generation of Link L objects in *Robotics Toolbox for Matlab* the dynamical parameters of each link have to be assigned:

Mass of a Link:	$L.m$, here $m = \rho w^2 L$
COG vector:	$L.r$, with $r = [x, y, z]^T$, here $r = [-L/2, 0, 0]^T$
Inertia of the links:	$L.I$, with $\boldsymbol{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix}$
	here $I_{xx} = \frac{m}{6} w^2; I_{yy} = I_{zz} = \frac{m}{12}(L^2 + w^2); I_{xy} = I_{xz} = I_{yz} = 0$
Inertia of the motor:	$L.Jm = 0$
Gear ratio:	$L.G = 1$

Additionally the gravity vector of the SerialLink Rob object has to be corrected. Pleas pay attention to the fact that the gravitational vector has to be defined in the opposite direction of the base coordinate frame $Rob.gravity = [0, +9.81, 0]^T$.

Exercise 2: Forward dynamics

For this exercise the robot from the first Matlab assignment will be used as shown in the figure below. The length of each link is given by $L_1 = 4$ m, $L_2 = 3$ m und $L_3 = 2$ m. The mass of the links is $m_1 = 20$ kg, $m_2 = 15$ kg, und $m_3 = 10$ kg. The Inertias are equal to $I_{zz1} = 0.5$ kgm^2, $I_{zz2} = 0.2$ kgm^2, und $I_{zz3} = 0.1$ kgm^2 (the rest of the inertias have no effect on the dynamics and therefore can be set to zero). Furthermore the following assumptions hold: the center of gravity is assumed to be in the middle of each link, the motor and gears are assumed to be ideal (no frictions and inertias and a gear ratio of one) and the gravitational acceleration is defined in negative y_0 direction. Simulate the dynamic motion of the robot in the gravitational



field without additional excitation. The following parameters will be used:

- Joint angles $\boldsymbol{q}(t_i) = \boldsymbol{q}_i = (\theta_{1i}, \theta_{2i}, \theta_{3i})^T$ [rad, rad, rad]
- Joint angular velocity $\dot{\boldsymbol{q}}(t_i) = \dot{\boldsymbol{q}}_i = (\dot\theta_{1i}, \dot\theta_{2i}, \dot\theta_{3i})^T$ [rad/s, rad/s, rad/s]
- Joint angular acceleration $\ddot{\boldsymbol{q}}(t_i) = \ddot{\boldsymbol{q}}_i = (\ddot\theta_{1i}, \ddot\theta_{2i}, \ddot\theta_{3i})^T$ [rad/s^2, rad/s^2, rad/s^2]
- Joint torques (opposed to gravitation) $\boldsymbol{\tau}(t_i) = \boldsymbol{\tau}_i = (\tau_{1i}, \tau_{2i}, \tau_{3i})^T$ [Nm, Nm, Nm]
- Time steps of the simulation $\boldsymbol{t} = (t_0, t_1, t_2, \dots, t_n)$ [s]
- Simulation time step $\triangle t$
- Overall simulation time T [s]

The generated joint torques should **only** oppose the gravitational acceleration, i.e. the torques necessary for the robot to stay in the same configuration.

a) Generate the SerialLink object *Rob* with the function **createRobot**. Use the additional notes from exercise 1 b) and the gravitational acceleration $g = 9.81$ [m/s^2].

(6)	Rob = createRobot()	MAT
Input:	—	
Output:	Rob SerialLink object (according to the description above)	
Allowed routines:	Link(), SerialLink()	

b) Next the function **fwddyn** should be implemented to determine the behavior of the robot in the gravitational field without excitation. Therefore use the functions **gravload()** and **accel()** of the *Robotics Toolbox*. Starting time is $t_0 = 0$ s and the robot is assumed to be at rest, i.e. $\dot{\boldsymbol{q}} = \boldsymbol{0}$[rad/s]. To compute the joint angles and angular velocities use the following integration schemes:

$$\dot{\boldsymbol{q}}_{k+1} = \dot{\boldsymbol{q}}_k + \ddot{\boldsymbol{q}}_k \triangle t$$
$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \dot{\boldsymbol{q}}_k \triangle t + \ddot{\boldsymbol{q}}_k \frac{(\triangle t)^2}{2}$$

(12)	$[\boldsymbol{Q}, \dot{\boldsymbol{Q}}, \ddot{\boldsymbol{Q}}, \boldsymbol{t}, \boldsymbol{\Gamma}] =$ fwddyn$(Rob, \boldsymbol{q}_0, \triangle t, T)$	MAT
Input:	Rob SerialLink object (according to description above)	
	$\boldsymbol{q}_0$ 3×1 vector	
	$\triangle t$ scalar quantity	
	T scalar quantity	
Output:	$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{q}_0 & \boldsymbol{q}_1 & \dots & \boldsymbol{q}_n \end{bmatrix}$ 3×(n+1) matrix	
	$\dot{\boldsymbol{Q}} = \begin{bmatrix} \dot{\boldsymbol{q}}_0 & \dot{\boldsymbol{q}}_1 & \dots & \dot{\boldsymbol{q}}_n \end{bmatrix}$ 3×(n+1) matrix	
	$\ddot{\boldsymbol{Q}} = \begin{bmatrix} \dot{\boldsymbol{q}}_0 & \dot{\boldsymbol{q}}_1 & \dots & \dot{\boldsymbol{q}}_n \end{bmatrix}$ 3×(n+1) matrix	
	$\boldsymbol{t} = \begin{bmatrix} t_0 & t_1 & \dots & t_n \end{bmatrix}$ 1×(n+1) matrix	
	$\boldsymbol{\Gamma} = \begin{bmatrix} \boldsymbol{\tau}_0 & \boldsymbol{\tau}_1 & \dots & \boldsymbol{\tau}_n \end{bmatrix}$ 3×(n+1) matrix	
Allowed routines:	gravload(), accel()	

Test: Use the following parameters for your test.

- $\boldsymbol{q}_0 = (0°, 0°, 0°)^T$
- $T = 0.5$ [s]
- $\triangle t = 0.01$ [s]
- $g = 9.81$ [m/s^2]

Compute the joint angles and angular velocities for the same initial conditions using the function **fdyn()** of the *Robotics Toolbox for Matlab*. Compare the results of the toolbox function and your implementation by plotting them.

Hint:
The function **fdyn()** of the *Robotics Toolbox for Matlab* uses a Runge-Kutta integration scheme. Therefore it generates slightly different results for $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. However, the trajectories are the same during the first few seconds. The maximum error of the joint angles should not exceed $5°$.

c) Next, the behavior of the robot will be evaluated by using Simulink. Therefore download the Simulink model *simdyn.slx* from Moodle. There you will find the prepared block $\boxed{\textbf{Robot}}$. It is taken from the *Robotics Toolbox for Matlab*, however, it was separated from the library and modified. Please do not change the block or any parameters of this Simulink file. However, you should get accustomed to the mask parameters, so double click on the mask. Next follow these steps:

- The robot should not be actuated. Therefore use a constant torque of $[0\ 0\ 0]^T$ [Nm, Nm, Nm] as input for the block $\boxed{\textbf{robot}}$, by using the block $\boxed{\textbf{constant}}$.
- Plot the joint angles $\boldsymbol{q}$ with respect to time by using the block $\boxed{\textbf{scope}}$. To scale the boundaries of the plot, you can click on the button **autoscale**. If some points cannot be seen, go to the tab "History" of the scope options and uncheck the option "Limit data points to last:".
- **Important:** Save the resulting joint angles $\boldsymbol{q}$ by using the block $\boxed{\textbf{to workspace}}$. The resulting variable should be named $\boldsymbol{Q}$ and it should be saved in the *Save format: Timeseries* format.

A Simulink model with those output parameters will be defined as follows in the future.

(0)	$\boldsymbol{Q} =$ simdyn()	SIM
Input:	—	
Output:	$\boldsymbol{Q}$	

This model has only introductory purposes for the next assignment and therefore has not to be submitted.