This assignment is concerned with robot dynamics and motion control.

The following exercises are solved with MATLAB and utilize the Robotics System Toolbox from Mathworks. Some tasks involve the use of ROS on a Linux (Ubuntu) system. Note, Matlab Linux uses different keyboard shortcuts per default (copy, paste, etc.). You might change it at *MATLAB - Preferences - Keyboard - Shortcuts - Active Settings: Windows Default Set*.

Before the assignment, read this document entirely and complete the quiz on dynamics and motion control in the Moodle workspace. In case you are unable to answer the questions in the quiz correctly go back to the lecture slides and please carefully study the chapters on dynamics (7.1-7.3) and motion control (8.1-8.5) in the textbook by Siciliano et al [1].

Robot Dynamic Model

Dynamic models of a robot arm are essential for the simulation of arm motion and the conception of advanced motion control schemes. The dynamic model provides the foundation for simulating the manipulator motion according to the forces and torques applied to the joint actuators. Advanced control strategies include an inverse dynamic model to predict and regulate the forces and torques required to achieve the desired arm motion.

The dynamic model of a robotic arm relates forces and torques with the acceleration of robot bodies and joints [1]. Forward dynamics determines joint accelerations given joint torques and states whereas inverse dynamics determines the joint torques that result in a desired joint motion.

In general, the joint space dynamic model includes the following terms:

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{F}_v\dot{\mathbf{q}} + \mathbf{F}_s\,\mathrm{sgn}(\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} - \mathbf{J}^T(\mathbf{q})\mathbf{h}_e \tag{1}$$

in which $\mathbf{B}$ denotes the configuration dependent inertia matrix, $\mathbf{C}$ incorporates centrifugal and Coriolis effects, $\mathbf{g}$ denotes a configuration dependent term, e.g. the moment generated by the presence of gravity.

Additional non-conservative forces/torques are included such like a viscous fiction torque $\mathbf{F}_v\dot{\mathbf{q}}$, Coulomb friction $\mathbf{F}_s\,\mathrm{sgn}(\dot{\mathbf{q}})$, Contact forces/torques $\mathbf{J}^T(\mathbf{q})\mathbf{h}_e$ and the actuation torque/load $\boldsymbol{\tau}$.

For each link, the joint space dynamic model contains the parameters

- m_i : rigid body mass
- $\mathbf{m_i l_{C_i}} = [m_i l_{C_i x}, m_i l_{C_i y}, m_i l_{C_i z}]$: center of mass position of the rigid body relative to the body frame
- $\mathbf{I}_i = \begin{pmatrix} I_{ixx} & I_{ixy} & I_{ixz} \\ I_{ixy} & I_{iyy} & I_{iyz} \\ I_{ixz} & I_{iyz} & I_{izz} \end{pmatrix}$: positive definite and symmetric inertia tensor of the rigid body relative to the body frame

The generalized forces $\boldsymbol{\tau}$ are generated by the actuators in conjunction with a transmission as shown in Figure 1. The relationship between joint motion and actuator motion is captured by

$$\mathbf{K}_r\mathbf{q} = \mathbf{q}_m \tag{2}$$

in which $\mathbf{K}_r$ is a diagonal matrix of gear reductions. Vica versa the relationship between actuator and joint torques is established by

$$\boldsymbol{\tau} = \mathbf{K}_r\boldsymbol{\tau}_m \tag{3}$$

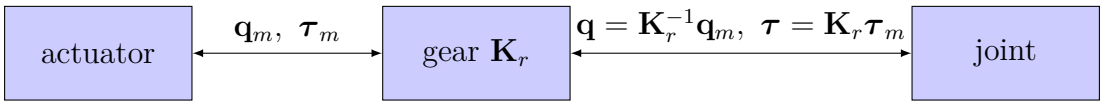

Figure 1: Relationship between actuator position $\mathbf{q}_m$ and joint position $\mathbf{q}$ and actuator torques $\boldsymbol{\tau}_m$ and joint torques $\boldsymbol{\tau}$.

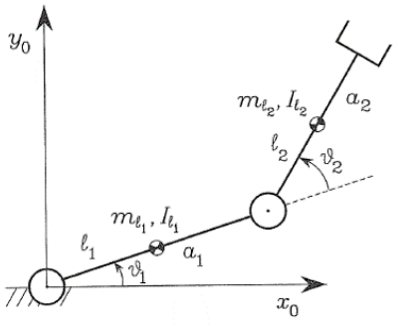

Figure 2: Two link planar arm.

θ	d	a	α
q_1	0	a_1	0
q_2	0	a_2	0

Table 1: DH parameters.

The first part of the assignment is concerned with the dynamic model and motion planning of the two link planar arm shown in figure 2).

The dynamics model of the two link arm is given by

$$\begin{aligned}
\tau_1 &= (I_{l1} + m_{l1}l_1^2 + I_{l2} + m_{l2}(a_1^2 + l_2^2 + 2a_1l_2\cos(\vartheta_2))\ddot{\vartheta}_1 \\
&\quad +(I_{l2} + m_{l2}(l_2^2 + a_1l_2\cos(\vartheta_2)))\ddot{\vartheta}_2 \\
&\quad -2m_{l2}a_1l_2\sin(\vartheta_2)\dot{\vartheta}_1\dot{\vartheta}_2 - m_{l2}a_1l_2\sin(\vartheta_2)\dot{\vartheta}_2^2 \\
&\quad +(m_{l1}l_1 + m_{l2}a_1)g\cos(\vartheta_1) + m_{l2}l_2g\cos(\vartheta_1 + \vartheta_2) \qquad (4) \\
\tau_2 &= (I_{l2} + m_{l2}(l_2^2 + a_1l_2\cos(\vartheta_2))\ddot{\vartheta}_1 \\
&\quad +(I_{l2} + m_{l2}l_2^2)\ddot{\vartheta}_2 \\
&\quad +m_{l2}a_1l_2\sin(\vartheta_2)\dot{\vartheta}_1^2 \\
&\quad +m_{l2}l_2g\cos(\vartheta_1 + \vartheta_2) \qquad (5)
\end{aligned}$$

Simscape Model Two Link Planar Arm

Simscape is a tool to design and implement dynamic models of mechanical, electrical and hydraulic systems within the Simulink environment. Physical component models are composed of physical connections that directly integrate with block diagrams. Simscape enables you to build your robot from basic mechanical components, in particular link bodies and joints. The robotics system toolbox (with release 2018b) allows it to import Simscape models into RigidBodyTree objects.

Figure 3 illustrates the Simscape model `twolinkarm.slx` of the two link planar arm composed of two revolute joints (`joint1, joint2`) and three link bodies (`link1, link2,`

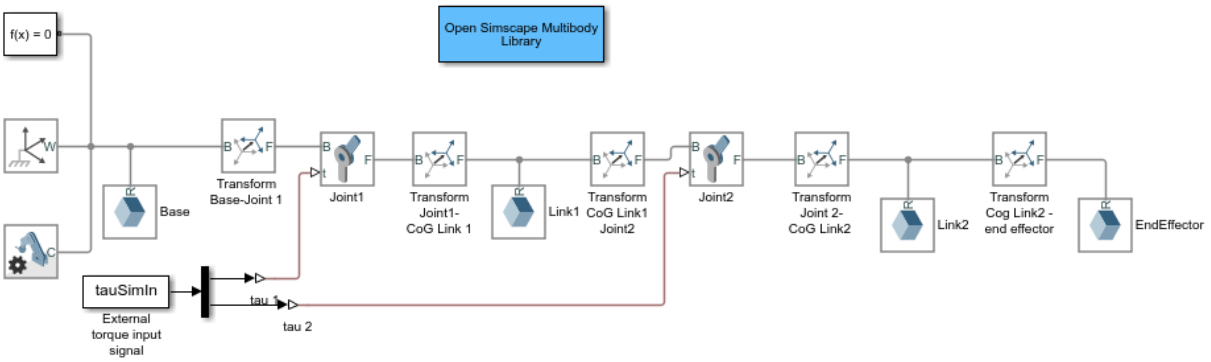

Figure 3: Simscape model of the two link planar arm.

`end effector`). The link bodies `link1, link2` are cylinders of link length a_1, a_2, radius r_1, r_2 and density ρ as shown in figure 4 which illustrates the link bodies, the joint frames and the link frames located at the center of gravity of the arm bodies. The arm motion is restricted to the x (red) - z (blue) -plane in the world frame, the joint axes are oriented along the global y-axis (green). The nominal dynamics parameters are $a_1 = a_2 = 1.0\,\mathrm{m}$, $r_1 = r_2 = 0.05\,\mathrm{m}$ and $\rho = 1000\,\mathrm{kg/m^3}$ which amount to a link mass $m_1 = m_2 = 7.854\,\mathrm{kg}$. The end effector payload is a massless sphere of radius $r_e = 0.1\,\mathrm{m}$ and density $\rho_e = 0\,\mathrm{kg/m^3}$. The center of gravity of the link bodies coincides with the center of the cylinders located at $a_1/2, a_2/2$ along the arm. The inertia is given by $I_{xx}, = I_{zz} = 2.6\,\mathrm{kg\ m^2}$, $I_{yy} = 0.0098\,\mathrm{kg\ m^2}$. The joint model includes a viscous friction term $\boldsymbol{\tau}_\mu = -\mu\dot{\mathbf{q}}$ with the friction coefficient $\mu = 0\,\mathrm{Nm/(deg/s)}$. The actuator torques are provided by external input from the workspace `tauSimIn` specified as a time-series object with time-discrete instances $(t_i, \tau_{i1}, \tau_{i2})$ of time stamps t_i and input torques τ_{i1}, τ_{i2}.

The dynamic parameters of the Simcape model are defined in the model workspace and can be modified either in Simulink (Tools -> Model Explorer-> Model Workspace) or directly from the command line.

Now, please complete tasks 1 to 5 in the template.

Notes:

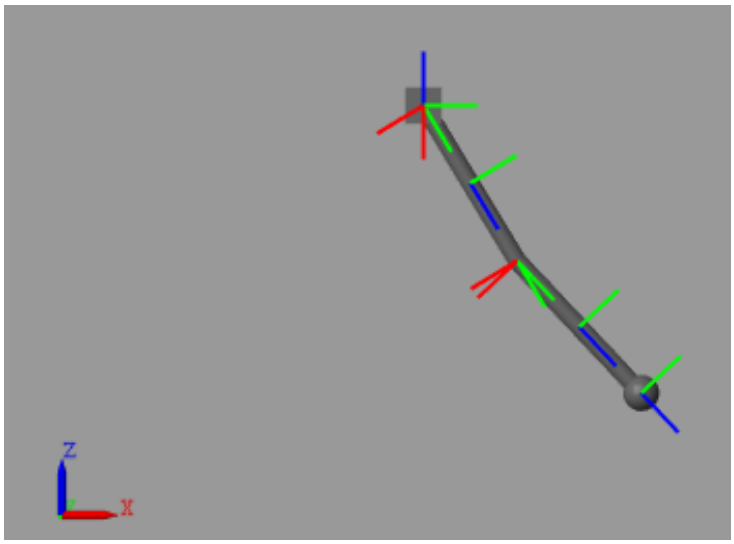

Figure 4: Simscape visualization of the two link planar arm with joint and link frames.

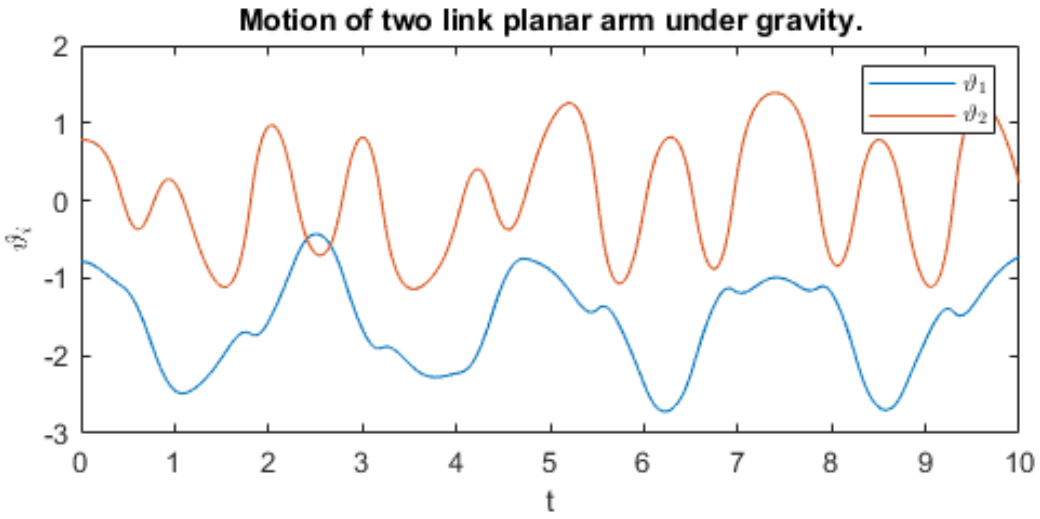

Figure 5: Ego motion of the two link planar arm with gravity.

Robot Dynamics in Robotics System Toolbox

Since Matlab2018b, Simscape models can be directly imported as `RigidBodyTree` objects. In case your release is older than 2018b load the two link arm rigid body tree object from the file `twolinkarm.mat`.

```
if verLessThan('matlab','9.5')  % Release 2018b
  load 'twolinkarm.mat'  % Load RigidBodyTree robot from MAT-File
else
  robot=importrobot(gcs);  % import Rigid Body Tree from Simscape
  % add body for end effector
  body3=robotics.RigidBody('Body3');
  jnt3=robotics.Joint('Joint3','fixed');
  body3.Mass=1e-6;
  body3.Inertia=[1e-6 1e-6 1e-6 0 0 0];
  jnt3.setFixedTransform(trvec2tform([0 a2 0]));
  body3.Joint=jnt3;
  robot.addBody(body3,'Body2');
  % add visuals for link bodies to RigidBodyTree object
  addVisual(robot.Bodies{1,2},'Mesh','cylinder.stl',eul2tform([0 0 -pi/2]));
  addVisual(robot.Bodies{1,1},'Mesh','cylinder.stl',eul2tform([0 0 -pi/2]));
  addVisual(robot.Bodies{1,3},'Mesh','sphere.stl');
end
robot.DataFormat='column';
```

The RigidBodyTree object does not import the Simscape model visuals, neither the sphere at the end effetor. These need to be added manually in case you import the `robot` directly from Simscape rather than the MAT-file. Finally the data format for the RigidBodyTree `robot` is set to column such that member functions accept ordinary Matlab vectors or arrays rather than joint configuration objetcs.

The function

```
jointAccel = forwardDynamics(robot,configuration,jointVel,jointTorq)
```

or as member function

```
jointAccel = robot.forwardDynamics(configuration,jointVel,jointTorq)
```

computes the joint accelerations due to gravity at the robot `configuration`, due to centrifugal and Coriolis torques depending on `jointvel` and due to actuator torques `jointTorq`. It calculates the resultant joint accelerations for a given robot configuration with forces due to the forward dynamic model. The resulting robot arm motion is simulated with `ode` with state vector $[\mathbf{q}\ \dot{\mathbf{q}}]$ as $2 \times n$ set of first order differential equations.

This code simulates the forward dynamics of the two link planar arm under gravity without friction and with a constant torque input τ by numerical integration of the joint acceleration calculated according to `robot.forwardDynamics` over a period $t \in [0, 10]$ with

```
  [t,q] = ode45(odefun,tspan,q0)
```

(see lab Simulink I in Scientific Programming in Matlab or Matlab Documentation).

`ode45` integrates the differential equation according to the function handle `odefun` from `t0` to `tf` with initial conditions `q0`. In case of the two link arm, `q` is the joint state vector given by $\mathbf{q} = [\vartheta_1, \vartheta_2, \dot{\vartheta}_1, \dot{\vartheta}_2]$ governed by the set of differential equations

$$\begin{aligned} \dot{q}_1 &= \dot{\vartheta}_1 = q_3 \\ \dot{q}_2 &= \dot{\vartheta}_2 = q_4 \\ \dot{q}_3 &= \ddot{\vartheta}_1 \\ \dot{q}_4 &= \ddot{\vartheta}_2 \end{aligned}$$

with $\ddot{\vartheta}_1$, $\ddot{\vartheta}_2$ calculated according to the forward dynamics in `robot.forwardDynamics`. The initial arm configuration is at the lower equilibrium with zero velocity which corresponds to the joint state $\mathbf{q}_0 = [q_1, q_2, \dot{q}_1, \dot{q}_2] = [-\pi/2, 0, 0, 0]$ exposed to a constant torque $\boldsymbol{\tau} = [50.0\ 20.0]$.

```
% constant torque
tsteps=linspace(0,tfinal,tfinal/dt); % time steps discretization of torque profile
tau=zeros(length(tsteps),2);  % zero torque
tau(ceil(length(tau)/10):length(tau),1)=50.0; % external torque joint 1
tau(ceil(length(tau)/10):length(tau),2)=20.0; % external torque joint 2
% initial state ode q_1 q_2 dot q_1 dot q_2
qinitialstate=[-pi/2 0 0 0]; % lower equilibrium
[t,q] = ode45(@(t,q) [q(3) q(4) robot.forwardDynamics(q(1:2), q(3:4), ...
   interp1(tsteps,tau,t)')']'  , [0 tfinal], qinitialstate);
```

The function `interp1` linearly interpolates the signal $\tau(t)$ from the discrete time series $\tau(t_i)$ according to

$$\tau(t) = \frac{(t_{i+1} - t)\tau(t_i) + (t - t_i)\tau(t_{i+1})}{t_{i+1} - t_i} \tag{6}$$

Now, please complete tasks 6 to 9 in the template.

Notes:

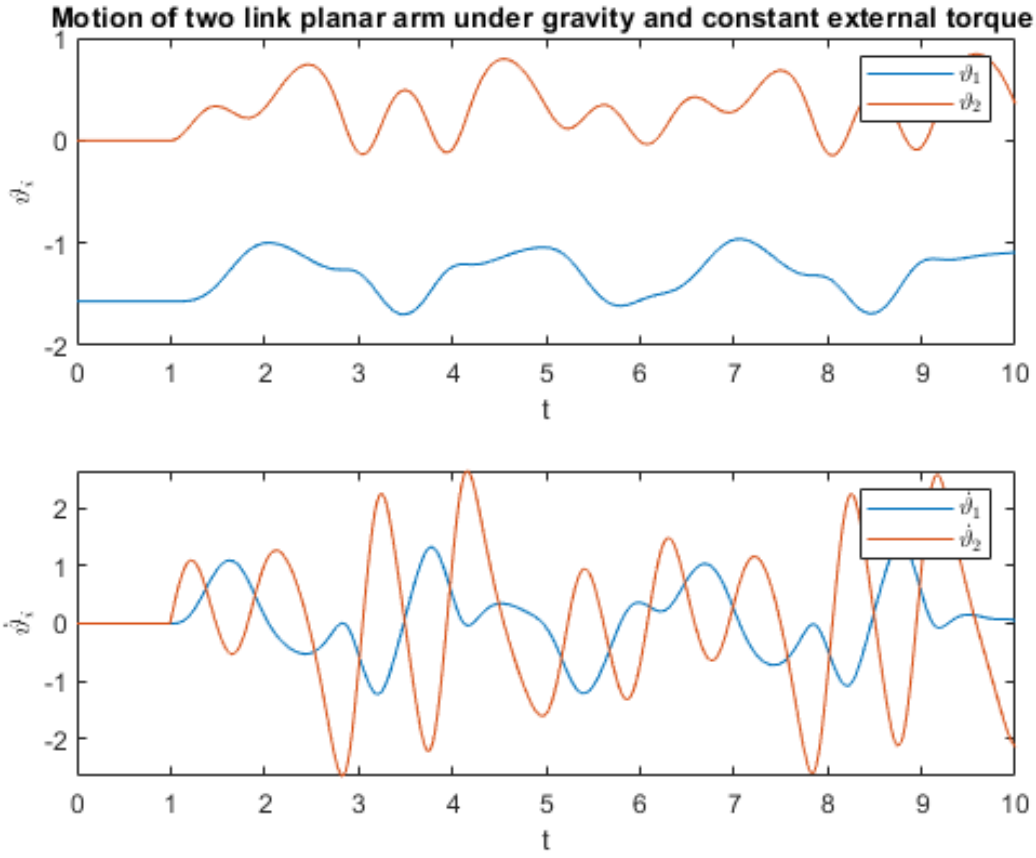

Figure 6: Motion of the two link planar arm under gravity with no friction.

Inverse Dynamic Model

Computed torque control is a model-based nonlinear control strategy with the objective to enhance trajectory tracking performance of joint motion control. The disturbances caused by the interactions between joints are compensated by a centralized torque command which is calculated from the nominal trajectory according to the robot's inverse dynamic model.

$$\boldsymbol{\tau}_d = \mathbf{B}(\mathbf{q_d})\ddot{\mathbf{q}}_{md} + \mathbf{C}(\mathbf{q}_d, \dot{\mathbf{q}}_d)\dot{\mathbf{q}}_{md} + \mathbf{g}(\mathbf{q}_d)$$

The function

```
jointTorq = robot.inverseDynamics(configuration,jointVel,jointAccel)
```

computes the joint torques $\boldsymbol{\tau}$ required for the robot to compensate gravity torques, generating inertia torques and Coriolis and centrifugal torques to achieve the desired joint configuration, velocity and acceleration.

Now, please complete tasks 10 to 14 in the template.

Notes:

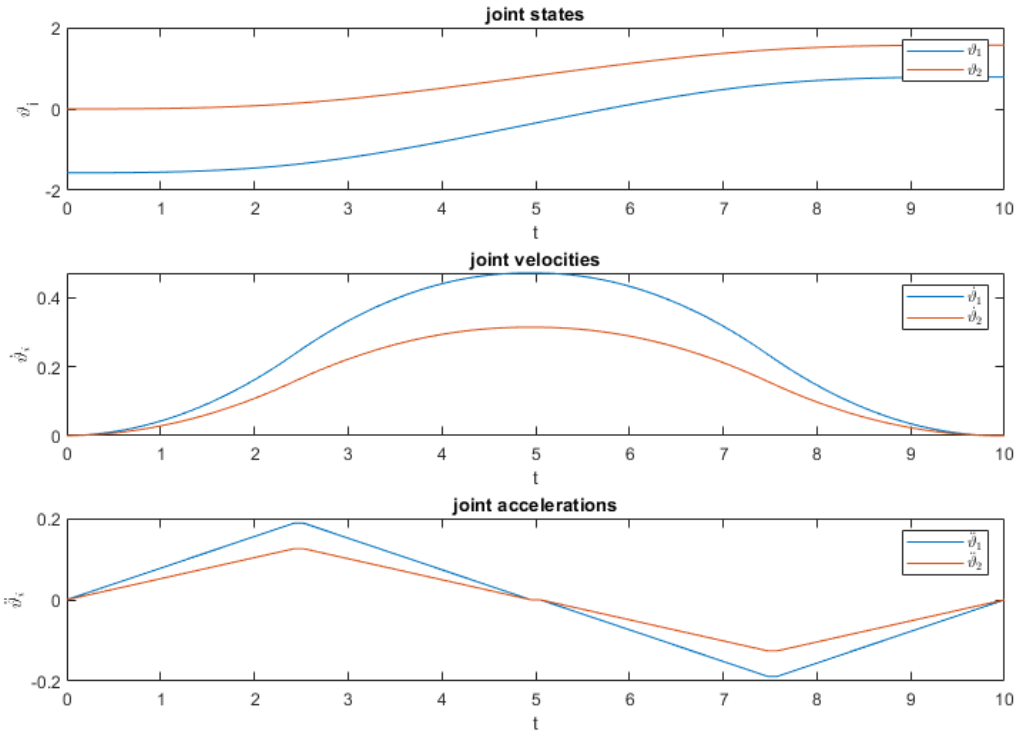

Figure 7: Joint motion reference profile for a two link planar arm.

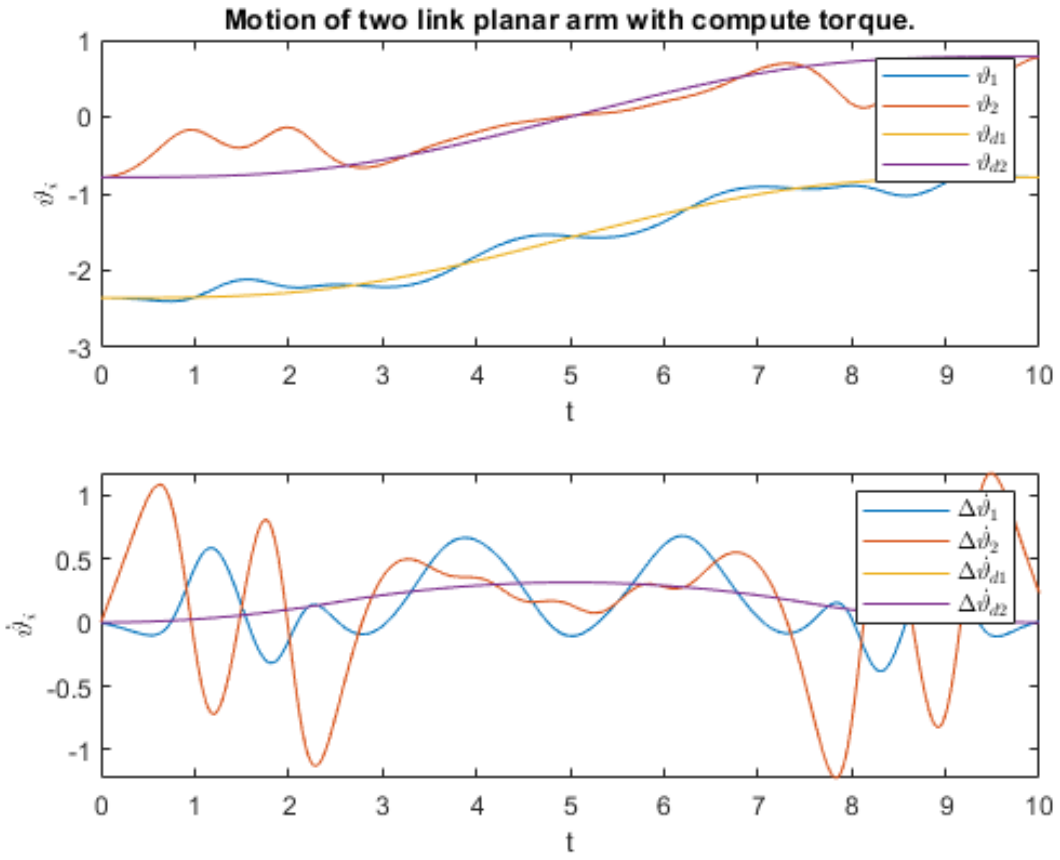

Figure 8: Joint motion reference profile and joint motion from computed torque for the two link planar arm.

Decentralized Joint Motion Control

A decentralized control strategy considers the joint motions of the manipulator as independent. The motion of each joint axis is regarded as a SISO system, and the dynamic couplings resulting from the motion of the remaining joints are treated as disturbances.

The joint motion controller tracks a joint reference trajectory $\mathbf{q}_r, \dot{\mathbf{q}}_r$. The proportional-derivative control law determines the torque input $\boldsymbol{\tau}$

$$\boldsymbol{\tau} = \mathbf{K}_p(\mathbf{q}_r - \mathbf{q}) + \mathbf{K}_d(\dot{\mathbf{q}}_r - \dot{\mathbf{q}}) + \mathbf{K}_{dd}\ddot{\mathbf{q}}_\mathbf{r} \tag{1}$$

from the joint error $\mathbf{e}_q = \mathbf{q}_r - \mathbf{q}$ and its time derivative $\dot{\mathbf{e}}_q = \dot{\mathbf{q}}_r - \dot{\mathbf{q}}$. The controller includes a feedforward term to generate the torque required for the reference joint acceleration $\ddot{\mathbf{q}}_r$. The diagonal gain matrices $\mathbf{K}_p$, $\mathbf{K}_d$, $\mathbf{K}_{dd}$ are obtained by linearizing the forward dynamics and model-based control design. The overall control scheme, therefore includes a position and a velocity feedback loop.

Notice, that a PD controller does not eliminate the steady error caused by the gravity disturbance torque. The integral action $\mathbf{K}_i \int_{t'=0} t(\mathbf{q}_r - \mathbf{q})dt'$ of a PID controller is able to eliminate the steady state error.

The template file `'jointFeedbackControllerTemplate.m'` provides sample code to implement joint feedback controllers. After download from Moodle rename it to `'jointFeedbackController.m'` such that file name and function name match.

```
function [tau] = jointFeedbackController(robot, q, dq, qref, dqref, ddqref,
  controllertype)
% Simple decentralized PID controller for joint motion control
% q, dq, current joint state
% qref, dqref, reference joint state

if (nargin<6)
ddqref=[0 0];
end

if (nargin<7)
controllertype='inverse dynamics';
end

qe=qref-q;
dqe=dqref-dq;

switch controllertype
%% task 15
case 'PD'
Kp=[200 0; 0 100];
Kd=[100 0; 0 5];
Kdd=[5 0; 0 2];
tau = (Kp*qe + Kd*dqe + Kdd*ddqref);

%% task 17-19
```

```
case 'PD with gravity compensation'

%%
case 'computed torque'

%% task 22-23
case 'inverse dynamics'

end
end
```

The input parameters `q, dq` refer to the current joint state, whereas `qref, dqref, ddqref` refer to the joint reference profile. `qe` and `dqe` denote the joint error and its derivative.

To simulate the closed-loop system replace the torque signal in forward dynamics simulation with `ode` by the feedback control signal determined by `jointFeedbackController`. Use `interp1` to generate the reference signals at simulation time `t` from the time series `qref`, `dqref`, `ddqref`.

```
robotmodel=robot.copy();
qinitial=[-pi/2 pi/2];
qinitialstate=[qinitial 0 0];
[t,q] = ode45(@(t,q) [q(3) q(4) forwardDynamics(robot, q(1:2), q(3:4),
    jointFeedbackController(robotmodel,q(1:2),q(3:4),interp1(tsteps,qref,t)',interp1(
    tsteps,dqref,t)',interp1(tsteps,ddqref,t)',controllertype))']'  , [0 tfinal],
    qinitialstate);
```

Now, please complete tasks 15 and 16 in the template.

PD Control with Gravity Compensation

PD control with gravity compensation ensures global asymptotic stability of the reference joint configuration. Figure 9 illustrates the block scheme of joint space PD control with gravity compensation. The control action u is composed of a term $\mathbf{g}(\mathbf{q})$ that balances the gravitational terms in (1) and a proportional-derivative control action to compensate disturbances.

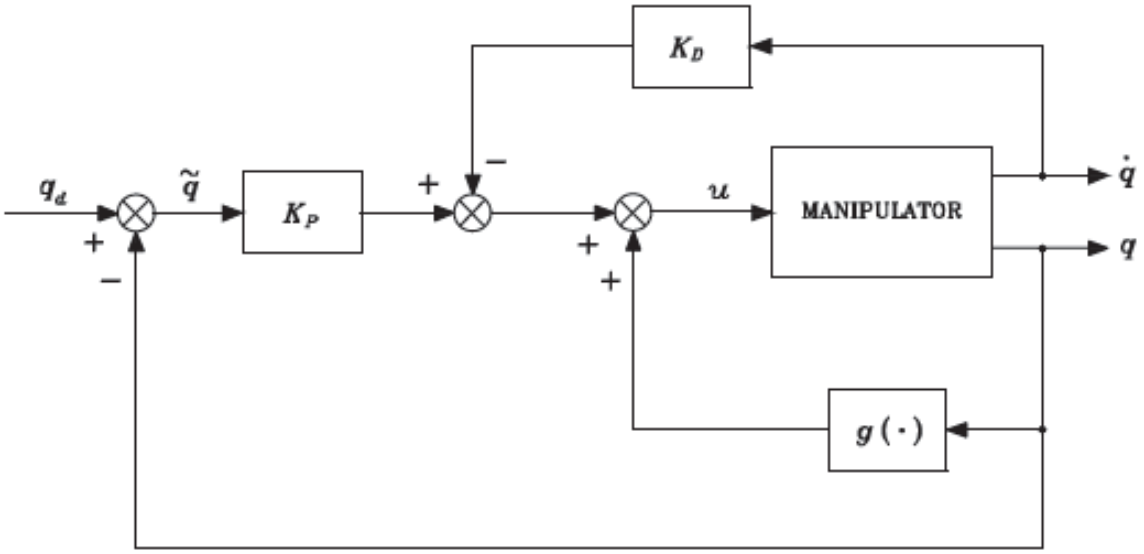

Figure 9: Joint space PD control with gravity compensation. [1]

Now, please complete tasks 17 to 21 in the template.

Inverse Dynamics Control

Figure 10 illustrates the block scheme of inverse dynamics control. Inverse dynamics control achieves a linearization of the system dynamics by compensating the nonlinear terms in (1). The term *inverse dynamics control* refers to the fact that the controller implicitly computes the inverse dynamics $\boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$. The system dynamics are linear and decoupled w.r.t. the new control input $\mathbf{y}$

$$\ddot{\mathbf{q}} = \mathbf{y} \tag{2}$$

with

$$u = \mathbf{B}(\mathbf{q})\mathbf{y} + \mathbf{n}(\mathbf{q}, \dot{\mathbf{q}}) \tag{3}$$

in which $\mathbf{n}(\mathbf{q}, \dot{\mathbf{q}})$ denotes the torque contributions of gravity, Coriolis and centrifugal forces in (1)

$$\mathbf{n}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) \tag{4}$$

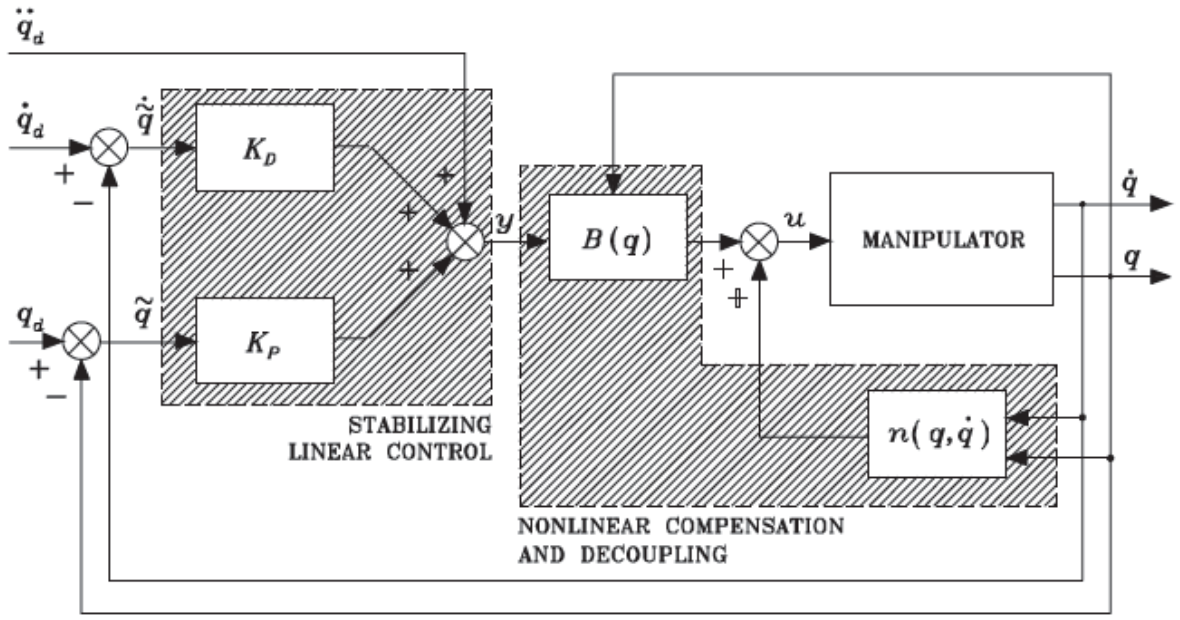

Figure 10: Inverse dynamics control scheme. [2]

Now, please complete tasks 22 to 25 in the template.

[2]source: Robotics : Modelling, Planning and Control [1]

References

[1] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. 2008.