5, May, 2022

Fabio Conti
Mat.4693053

Robot Dynamics and Control

Second assignment: report

Introduction:

This assignment covers the basics fundamental of the **recursive Newton-Euler algorithm** for solving the inverse dynamic problem. The Inverse dynamics problem for rigid body chains is a method for computing forces and moments based on the kinematics of a body and the body's inertial properties. Given the configuration $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ (generalized velocities), $\ddot{\boldsymbol{q}}$ (generalized accelerations), the **inertial properties** (mass and moment of inertia) of a rigid body chain, the effect of external moments and forces applied on it, the inverse dynamics problem will calculate the **joints actuation forces $\boldsymbol{\tau}$** needed to maintain that specific input set.

The **recursive Newton-Euler algorithm** is divided into two main loops:

1- **Forward Recursion**:

This loop will perform a forward iteration from the first link all the way up to the last one to calculate the following data set:

- The **configuration** of the i-th link with respect to the base frame: $\boldsymbol{r}_{i/o}$
- The **rotational matrices** between frames: ${}^{i-1}_{i}\boldsymbol{R}$.
- The **linear velocity** between the i-th link and the origin frame: $\boldsymbol{v}_{i/o}$.
- The **angular velocity** between the i-th link and the origin frame: $\boldsymbol{\omega}_{i/o}$.
- The **linear acceleration** between the i-th link and the origin frame: $\dot{\boldsymbol{v}}_{i/o}$.
- The **angular acceleration** between the i-th link and the origin frame: $\dot{\boldsymbol{\omega}}_{i/o}$.

2- **Backward Recursion**:

Starting from the just retrieved configuration, twist, and acceleration of each link, this loop will perform a backward iteration from the last link of the chain all the way to the first link to obtain the required joint forces and torques $\boldsymbol{\tau}$. Some intermediate computations will calculate the following set of data needed to get to the $\boldsymbol{\tau}$ vector:

- **Acceleration** of the i-th link's **center of mass**: $\dot{\boldsymbol{v}}_{ci/o}$.
- The **Dynamic force**: $\boldsymbol{D}_i$.
- The **dynamic moment**: Δ_i.

The set of joint forces and torques are needed to create the desired joint accelerations at the current joint positions and velocities all performed in **zero time** (all the quantities constant at a given time).

1. Robot data structure:

The robot configurations given at the beginning of each exercise contain info about the topological and structural positioning of the robot in space. To hold all the data about the robot structure, I created a *Matlab struct* containing both topological and dynamical info of the manipulator.

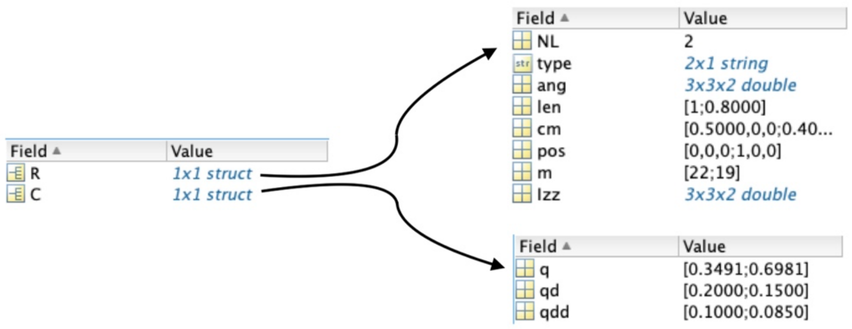
The *struct* is divided in two main *sub-structs*:

1) The first sub-struct (**R**) collects all the topological and constant info about the chain:
   - The section containing the number of links (**NL**) will contain a integer number expressing the number of links contained in the chain. [*int*]
   - The **type** section contains a string value specifying is the i-th joint is ither of type *revolute* or *prismatic.* [*string*]
   - The **ang** section will contain the **static rotational matrices** $\boldsymbol{R}_i$ between links' reference frames.
   - The **len** section will contain the **length** of each link from link 1 to link n. [m]

- The **cm** section will contain the **center of mass** of the i-th link with respect the i-th link's reference frame. [m]
- The **pos** section will contain the **position** of joint w.r.t. the i-th – 1 link's reference frame. [m]
- The **m** slot will contain the **mass** of every link of the chain. [kg]
- The **I** slot will contain the **moment of inertia** of every link. [kg*m*m]

2) The second subsection (**C**) of the of the robot *struct* will contain the information about the configuration, **generalized velocities**, and **generalized acceleration** of the current instance:
   - The section **q** will contain the values of the links' **configuration**: $\boldsymbol{q}$. [m]
   - The section **qd** will contain the values of the links' **generalized velocities**: $\dot{\boldsymbol{q}}$. [m/s]
   - The section **qdd** will contain the values of the links' **generalized acceleration**: $\ddot{\boldsymbol{q}}$. [m/s*s]

Thanks to the indexing of this structure, the forward and backward recursion could easily go through the data regarding the rigid-body chain.



Field	Value
R	1x1 struct
C	1x1 struct

Field	Value
NL	2
type	2x1 string
ang	3x3x2 double
len	[1;0.8000]
cm	[0.5000,0,0;0.40...
pos	[0,0,0;1,0,0]
m	[22;19]
Izz	3x3x2 double

Field	Value
q	[0.3491;0.6981]
qd	[0.2000;0.1500]
qdd	[0.1000;0.0850]

Fig.1: robot *struct*

2. Exercise 1: implementation of the recursive Newton-Euler algorithm for inverse dynamics.

The algorithm is implemented as a general *Matlab function* which takes as input arguments all the parameters related to the robot structure and trajectory and outputs the vector of the **generalized forces and torques**.

```
function [tau] = NewtEuler(robot,F_ext,M_ext,g)
```

INPUT args:
- **robot**: *struct* of the robot.
- **F_ext**: vector of the **external forces** expressed w.r.t. the base frame.
- **M_ext**: vector of the **external moments** expressed w.r.t. the base frame.
- **g**: vector of the **gravity** expressed w.r.t. the base frame.

OUTPUT args:
- **tau**: vector of forces and torques acting on the joints.

1.1 First section: computation of the rotational matrices

Throughout the first portion of the code I extracted all the values from the robot struct and put them into some arrays of different sizes to more easily iterate the values inside the loops.
The second step involved the actual calculation of the rotational matrices of every link with respect to different reference frames.
The first rotational matrices calculated express the rotation between the i-th link and the i-th – 1 link ($^{i-1}_{i}\boldsymbol{R}$).
Depending on the type of joint the calculation of these "relative" rotational matrices between links changes.
If the joint involved in the calculation is a **revolute** one, the algorithm has to consider not only the static transformation matrix between frames already present in the initial configuration but also the rotation given by the **configuration** $\boldsymbol{q}$**.**
In the case of a **prismatic** joint no extra rotation would be involved in the calculation of the rotational matrix given the movement actuated by the linear joint.
The following lines of code underline what was just underlined:

```
% Rigid rotational matrix between links
 R_i(:,:,i) = robot.R.ang(:,:,i);

  % Relative rotational matrix between frames
 if(robot.R.type(i) == "revolute")
     Rot(:,:,i)= R_i(:,:,i) * axang2rotm([0 0 1 q(i)]);

 elseif (robot.R.type(i) == "prismatic")
     Rot(:,:,i)= R_i(:,:,i);
 end
```

The *axang2rotm()* function will create a rotational matrix expressing the rotation around the *z-axis* ([0,0,1]) of a *q(i)* rotation. For a regular revolute joint represented through the **Denavit-Hartenberg** configuration, the rotation will always be expressed around the *z-axis*.

The third and last step of this section involves the computation of all the absolute rotational matrices relating the position of the links to the base reference frame.
To accomplish the computation of these matrices the algorithm must distinguish two different cases:

- The **first step**:
  Which will just equate the rotational matrix between the base frame and the first link to the one already claculated in the previous computational step.

- The **second step**:
  Where all the previous rotational matrices are post multiplicated by the i-th relative to build the whole chain of rotaitonal matrices and put into relationship all the links to the base all the way uo to the n-th one.

1.2 Second section: forward recursion

As already mentioned, this section of the code will retrieve all the information of the position, velocities, and acceleration (angular and linear) with respect to the base frames' space. To obtain these values the algorithm will directly pre-multiply each quantity with their relative absolute rotational matrix. This computation will directly refer all the quantities to reference frame making the global computations of the total forces and moments of the backward computation easier to perform. This loop is divided into two different subsections depending on the type of joint involved. Each of the two subsections are divided in two other subsubsections to distinguish the first step of the iteration to the following ones.

1.2.1 Forward recursion: revolute joint

This section will output the the absolute position of the i-th frame w.r.t. the base and also the angular and lienar velocities and accellerations of every revolute joint of the rigid body chain.

Postion computation:

```
r(:,i) = R_tot(:,:,i−1) ∗ robot.R.pos(i,:)';
```

The *robot.R.pos(i,:)'* column vector expresses the position of the i-th link w.r.t. the previous one's referance frame. Therefore, the absolute rotational matrix is refered to the i-th -1 frame instead of the current one.

Angular velocity computation:

```
w_i_0(:,i) = w_i_0(:,i−1) + R_tot(:,:,i) ∗ z ∗ qd(i);
```

The angular velocity of the i-th links depend on the previous link's angular velocity. Therefore, the *qd* rotational velocity is summed with the angular velocity of the previous link. Only the angular velocity component of the current joint is pre-multiplied by the *z-axis* rotated according to the base frame because the previous one has already been referred to the global reference frame.

Linear velocity computaiton:

```
v_i_0 (:,i) = v_i_0(:,i−1) + cross(w_i_0(:,i−1),r(:,i));
```

The linear velocity component for the revolute joint depends only on the effect given by the previous links velocities. The first element of the sum represents the previous joint's linear velocity. The second one is the linear velocity produced by the rotation of the previous joint and the relative distance between the current one and the previous one.

Angular acceleration computation:

```
wd_i_0(:,i) = wd_i_0(:,i−1) +cross(w_i_0(:,i−1),R_tot(:,:,i) ∗ z)∗ qd(i) +...
                    R_tot(:,:,i) ∗ z ∗ qdd(i);
```

The angular acceleration of the i-th joint is simply expressed as the first derivative of the velocity in time.

Linear acceleration computaiton:

```
vd_i_0 (:,i) = vd_i_0 (:,i−1) + cross(wd_i_0(:,i−1),r(:,i)) + cross(w_i_0(:,i−1), ...
                    cross(w_i_0(:,i−1),r(:,i)));
```

Same thing holds for the linear acceleration but concerning the linear velocity.

1.2.2 Forward recursion: prismatic joint

This section will output the absolute position of the i-th frame w.r.t. the base and the angular and linear velocities and accelerations of every prismatic joint of the rigid body chain.

Postion computation:

```
r(:,i) = R_tot(:,:,i−1) ∗ robot.R.pos(i,:)' + R_tot(:,:,i) ∗ z ∗ q(i);
```

Just like for the rotational joint, also in the prismatic case the position of the joint is referred to the previous joint's frame. Therefore, the first piece of the sum is pre-multiplied by the rotational matrix between the base frame and the i-th – 1's joint. Since the configuration displacement moves the prismatic joint upwards along its *z-axis* (**Denavit-Hartenberg**) the algorithm must take into account also this distance in the calculation. The distance must be referred to the base frame link with a pre-multiplication of the i-th global rotational matrix. In this case the displacement is expressed w.r.t. the current frame, therefore the rotational matrix used is expressed w.r.t. the i-th joint's reference frame.

Angular velocity computation:

```
w_i_0(:,i) = w_i_0(:,i−1);
```

In the case of the prismatic joint no angular velocity is added to the chain. Only the previous angular velocity created by the effect of the other joints will act on the current prismatic one.

Linear velocity computaiton:

```
v_i_0 (:,i)=v_i_0(:,i−1) + cross(w_i_0(:,i−1),r(:,i)) + R_tot(:,:,i)∗ z ∗qd(i);
```

The i-th linear velocity is represented as the sum of three components. The first one representing the velocity component given by the previous joints before the current one. The second one is the velocity component given by the rotational velocity of the previous joints crossed with the length of the i-th link. The last component is the instantaneous current velocity expressed by the current link projected onto the base frame.

Angular acceleration computation:

```
wd_i_0(:,i) = wd_i_0(:,i−1);
```

Just like in the case of the angular velocity, also the angular acceleration does not depend on the current link but only on the previous ones. The angular acceleration is of course the derivative of the previous angular velocity.

Linear acceleration computaiton:

```
vd_i_0 (:,i) = vd_i_0 (:,i−1) + (wd_i_0(:,i−1),r(:,i)) + ...
       cross(w_i_0(:,i−1),cross(w_i_0(:,i−1),r(:,i)))+ ...
       2 ∗ cross(w_i_0(:,i−1),R_tot(:,:,i) ∗ z) ∗ qd(i) +...
       R_tot(:,:,i) ∗ z ∗ qdd(i);
```

Even in the case of the prismatic joint, the linear velocity is represented as the first derivative of the current angular velocity.

1.3 Third section: backward recursion

As previously mentioned, this section of the computation will finally retrieve the value of the actuation forces and toques actuated by the joints to satisfy the current configuration ($\boldsymbol{\tau}$). As suggest by the name the step, the backward recursion is an iteration for the end of the chain (n-th joint) all the way down to the first joint. The backward recursion step of the algorithm is almost identical for both the revolute and the prismatic joint. The only difference is presented at the end of the algorithm where the only component of the total forces and moments acting on the i-th joint is the one **not reciprocal** to the actual joint. Namely, the total force

along the relative z axis for the prismatic joint, and the total moment acting around the relative z axis for the revolute joint.
The main objective for this section of the computation is to find the total moment and total forces acting on the i-th joint w.r.t. the base frame. Some side steps must be processed before calculating the final values for both the total moment and forces.

1.3.1 Backward recursion: Acceleration of the links' center of masses

The computations of the total moment and forces are all related to the center of mass of each link of the chain. This choice is made to later deal with easier computations of the moment of inertia related to the links and in general to make it easy to compute the whole effect of the forces and toques applied on the chain.

```
% Position of the links' center of mass
r_c_i(:,i) = R_tot(:,:,i) * robot.R.cm(i,:)';

% Acceleration of the center of mass
vd_c_i (:,i)= (vd_i_0(:,i) + ...
    cross(wd_i_0(:,i), r_c_i(:,i))+ ...
    cross(w_i_0(:,i),cross(w_i_0(:,i), r_c_i(:,i))));
```

Given all the data referred to the accelerations and velocities acting on the link at the joint position it is easy to calculate the acceleration of the center of mass of the link. The first element of the summation is the component of linear acceleration acting on the link. Since all the links are **rigid bodies** the linear acceleration of one point is the same once transposed to another position. The second element is related to the additional acceleration component given by the rotation of the link. Since the cm is not located in the fulcrum of the joint the rotation of the i-th joint will also cause a linear acceleration of the center of mass.

1.3.2 Backward recursion: Dynamic forces and Dynamic Moments

The computation of the final moments and torques requires the computation of the **Dynamic forces** and **Dynamic torques**. These elements are the effect of the movement of the whole chain on the i-th link.

Dynamic forces:

```
D_i(:,i) = robot.R.m(i) * vd_c_i(:,i);
```

The computation of the dynamic force is simply the force produced by the link given the acceleration of its center of mass coming from the Newton equation.

Dynamic torques:

```
delta_i(:,i)= R_tot(:,:,i)*(robot.R.Izz(:,:,i))*R_tot(:,:,i)' * wd_i_0(:,i)+ ...
cross(w_i_0(:,i),R_tot(:,:,i) *(robot.R.Izz(:,:,i))* R_tot(:,:,i)' *w_i_0(:,i));
```

The dynamic moment component is composed by two different elements. The first component of the summations comes directly form the Euler equation while the second one represents the centripetal effect. The moment of inertia is pre and post multiplied by the global rotational matrix to interpret the inertial property form the base frame.

1.3.3 Backward recursion: Total forces and total torques

The calculation of the total forces and toques acting on each link is a summation of all the forces and moments effects acting on the single rigid-body. The algorithm distinguishes the first step of the iteration from the other once since the contributions are different from all the other steps.

Total Force:

```
F_i(:,i) = F_i(:,i+1) − robot.R.m(i) ∗ g − F_ext + D_i(:,i);
```

The total force computation takes into accout other then the dynamic force also the effect of the external forces acting on the i-th body. The first component is the effect of the previous link acting on the current one. The second force is expressed by the gravity acceleration vector mutiplied by the mass of the current link. The Third component is the external force acting on the body w.r.t. the base frame. As previously mentioned, the last component is the dynamic force component.
Total torque:

```
M_i(:,i) = M_i(:,i+1) − M_ext − cross(− r_c_i(:,i), F_i(:,i)) + cross(r(:,i+1) −
                          r_c_i(:,i), F_i(:,i+1)) + delta_i(:,i);
```

Regarding the total forces acting on the i-th body, all the external moments acting on the body will be considered. The first element of the equation represents the moment applied by the previous link of the iteration on the current one. The following elements regards the external moments applied directly on the body w.r.t. the reference frame. The third component represents the moment component caused by the total force applied on the i-th link. The fourth element represents the moment component applied by the total force applied on the previous link on the center of mass of the current one. As previously mentioned, the last component is the dynamic moment component.

1.3.4 Backward recursion: computation of τ

As previously mentioned at the beginning of this paragraph, the computation of the τ components differs between links based on their type of joint. All the reciprocal effects acting on each joint are not considered in the final computation. Therefore, the reciprocal effects are different between joint types:

```
% Calculation of tau for the revolute joints
if(robot.R.type(i) == "revolute")
    tau(i) = dot(M_i(:,i) , (R_tot(:,:,i) ∗ z));

% Calculation of tau for the prismatic joints
elseif(robot.R.type(i) == "prismatic")
    tau(i) = dot(F_i(:,i) , (R_tot(:,:,i) ∗ z));

end
```

For the revolute joint only the total moment component acting on its own z axis is considered. While, the only effect considered for the prismatic is the force along its z-axis.

3. Exercise 2: Finding the actuated forces and torques for each link of the given robot configurations

The **recursive Newton-Euler algorithm** was later tested on three different rigidbody chains. To test most of its sections, the manipulators presented different kind of features in terms of number of links, types of links, and their positioning in space. For each of the configuration the solution provided are computed with and

without gravity. None of the exercises explicitly asked for any kind of external forces or moments applied on the rigidbody chain.
The following tables show all the various features of each robot configuration and the values of $\boldsymbol{\tau}$ obtained **with** and **without gravity**:

Ex.1

Structural features:

Robot features	*Values*
Number of links [int]	2
Types of links [string]	"revolute", "revolute"
Length of links [m]	1, 0.8
Mass of links [kg]	22, 19
*Moment of inertia [kg*m*m]*	*diag([0.4,0.4,0.4]), diag([0.3,0.3,0.3])*

Configurations' features:

Configurations	*Values*
Configuration 1	(q_1, q_2): *20°, 40°* $(\dot{q}_1, \dot{q}_2)$: *0.2 rad/s, 0.15 rad/s* $(\ddot{q}_1, \ddot{q}_2)$: *0.1 rad/s*s, 0.085 rad/s*s*
Configuration 2	(q_1, q_2): *90°, 45°* $(\dot{q}_1, \dot{q}_2)$: *-0.8 rad/s, 0.35 rad/s* $(\ddot{q}_1, \ddot{q}_2)$: *-0.4 rad/s*s, 0.1 rad/s*s*

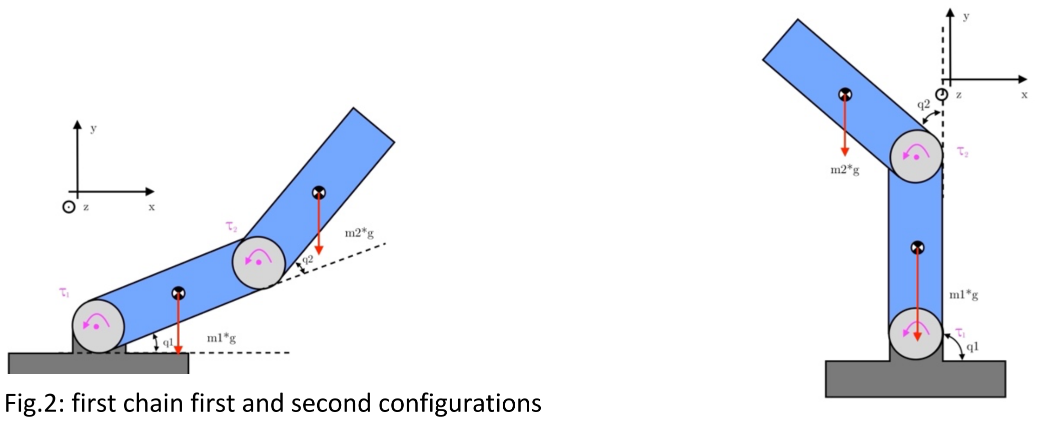


Fig.2: first chain first and second configurations

Values of the actuation forces and torques $\boldsymbol{\tau}$ w.r.t. the base frame:

Configurations	*Values of* $\boldsymbol{\tau}$ *with gravity*	*Values of* $\boldsymbol{\tau}$ *with no gravity*
Configuration 1	$\tau_1 = 318.1937$ [Nm] $\tau_2 = 38.6735$ [Nm]	$\tau_1 = 4.3641$ [Nm] $\tau_2 = 1.3955$ [Nm]
Configuration 2	$\tau_1 = -65.0917$ [Nm] $\tau_2 = -52.4313$ [Nm]	$\tau_1 = -12.3727$ [Nm] $\tau_2 = 0.2878$ [Nm]

Comments: In the first configuration the biggest difference between the two cases is the module of toques. In the gravity case, the rigidbody chain must counter the gravity force which acts against the joints' acceleration. Therefore, the module of the torque is higher than the no gravity case. In the second configuration, the actuation torques apply a negative acceleration for the first link and a positive one for the second. In the gravity case, both joints have negative torques even though the second one should rotate in the direction of the gravity force. This is because the acceleration to obtained is lower than the gravity would provide without actuated torques. This point is clarified on the no-gravity case where the sign of the torque is positive since the weight force is not acting anymore.

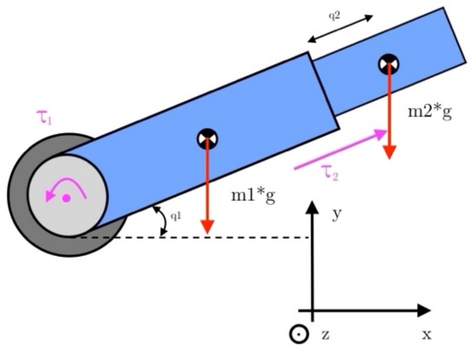
Ex.2

Structural features:

Robot features	*Values*
***Number** of links* [int*]*	2
***Types** of links* [string*]*	"revolute", "prismatic"
***Length** of links* [m*]*	1, 0
***Mass** of links* [kg*]*	10, 6
Moment of inertia [kg*m*m*]*	*diag([0.4,0.4,0.4]), diag([0.3,0.3,0.3])*

Configurations' features:

Configurations	*Values*
Configuration 1	$(\boldsymbol{q_1}, \boldsymbol{q_2})$*: 20°, 0.2 m* $(\boldsymbol{\dot{q}_1}, \boldsymbol{\dot{q}_2})$*: 0.08 rad/s, 0.03 m/s* $(\boldsymbol{\ddot{q}_1}, \boldsymbol{\ddot{q}_2})$*: 0.1 rad/s*s, 0.01m/s*s*
Configuration 2	$(\boldsymbol{q_1}, \boldsymbol{q_2})$*: 120°, 0.6 m* $(\boldsymbol{\dot{q}_1}, \boldsymbol{\dot{q}_2})$*: -0.4 rad/s, -0.08 m/s* $(\boldsymbol{\ddot{q}_1}, \boldsymbol{\ddot{q}_2})$*: -0.1 rad/s*s, -0.01m/s*s*



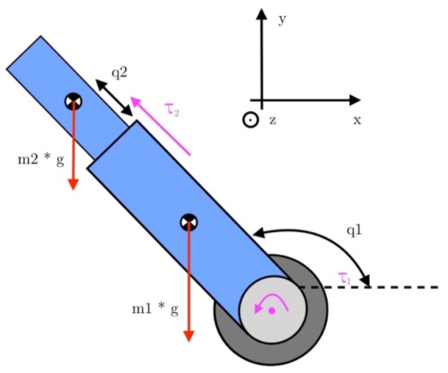


Fig.3: second chain first and second configurations

Values of the actuation forces and torques $\boldsymbol{\tau}$ w.r.t. the base frame:

Configurations	*Values of* $\boldsymbol{\tau}$ *with gravity*	*Values of* $\boldsymbol{\tau}$ *with no gravity*
Configuration 1	$\boldsymbol{\tau_1} = 113.6829$ [Nm] $\boldsymbol{\tau_2} = 20.1452$ [N]	$\boldsymbol{\tau_1} = 1.2186$ [Nm] $\boldsymbol{\tau_2} = 0.0139$ [N]
Configuration 2	$\boldsymbol{\tau_1} = -72.8546$ [Nm] $\boldsymbol{\tau_2} = 49.3783$[N]	$\boldsymbol{\tau_1} = -1.2416$ [Nm] $\boldsymbol{\tau_2} = -1.5960$ [N]

Comments: the change of sign between configurations of the torque can be explained in the same way as for the first exercise. In the second instance there's a different sign of τ_2 once gravity is "turned off". Therefore, we can conclude that the manipulator must apply a positive force even if the acceleration of the prismatic joint is negative. In this configuration the joint force has to counterbalance the gravitational force even if the acceleration direction has to be negative.

Ex.3

Structural features:

Robot features	*Values*
Number of links [int]	3
Types of links [string]	"revolute", "revolute", "revolute"
Length of links [m]	1, 0.8, 0.35
Mass of links [kg]	20, 20, 6
*Moment of inertia [kg*m*m]*	*diag([0.2,0.2,0.8]), diag([0.2,0.2,0.8]), diag([0.08,0.08,0.1])*

Configurations' features:

Configurations	*Values*
Configuration 1	(q_1, q_2, q_3): 20°, 40°, 10° $(\dot{q}_1, \dot{q}_2, \dot{q}_3)$: 0.2 rad/s, 0.15 rad/s, -0.2 rad/s $(\ddot{q}_1, \ddot{q}_2, \ddot{q}_3)$: 0.1 rad/s*s, 0.085 rad/s*s, 0 rad/s*s

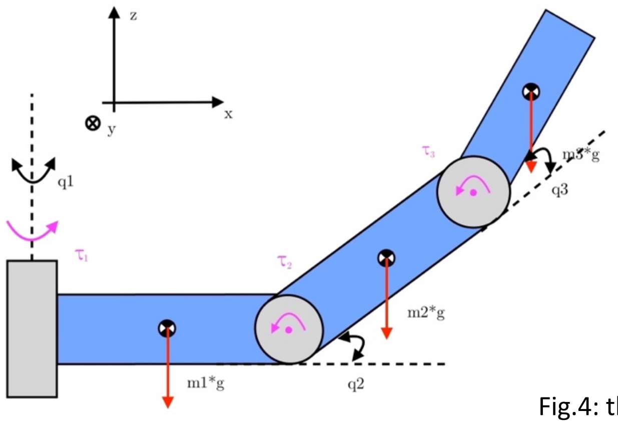


Fig.4: third chain

Values of the actuation forces and torques τ w.r.t. the base frame:

Configurations	*Values of τ with gravity*	*Values of τ with no gravity*
Configuration 1	$\tau_1 = 5.1128$ [Nm] $\tau_2 = 104.1829$ [Nm] $\tau_3 = 6.7743$ [Nm]	$\tau_1 = 5.1128$ [Nm] $\tau_2 = 1.3712$ [Nm] $\tau_3 = 0.1532$ [Nm]

Comments: this is the only case where the chain has more than two links and they are not in a planar configuration. The noticeable things of the resulting $\boldsymbol{\tau_1}$ is the fact that it doesn’t change between the gravity and no gravity case. Therefore, no external perpendicular forces would change the behavior of the first joint since all of them would be reciprocal to it.