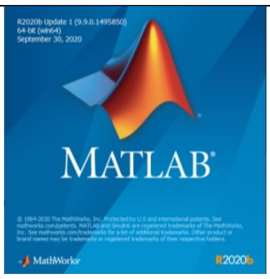
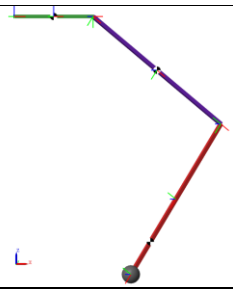
Application exercises on Simscape and Multibody Toolbox of Matlab / Simulink environment

Robotics: Modelling of kinematic chains

Prerequisites: Basics of Matlab / Simulink or another similar graphical simulation environment.

Deliverables: A report that includes your answers to both exercises is to upload on the dedicated Moodle Space. The models and parameters files are also to upload (.slx .m files). This work may be done by pairs.

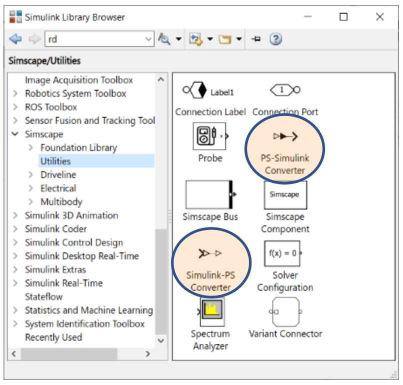




1 - Introduction

Simscape and Multibody are 2 toolboxes of Matlab / Simulink designed to model and simulate physical systems and more especially mechanical chains. For the robotics course, we will use in priority Multibody toolbox that relies on Simscape toolbox to run simulations of physical systems.

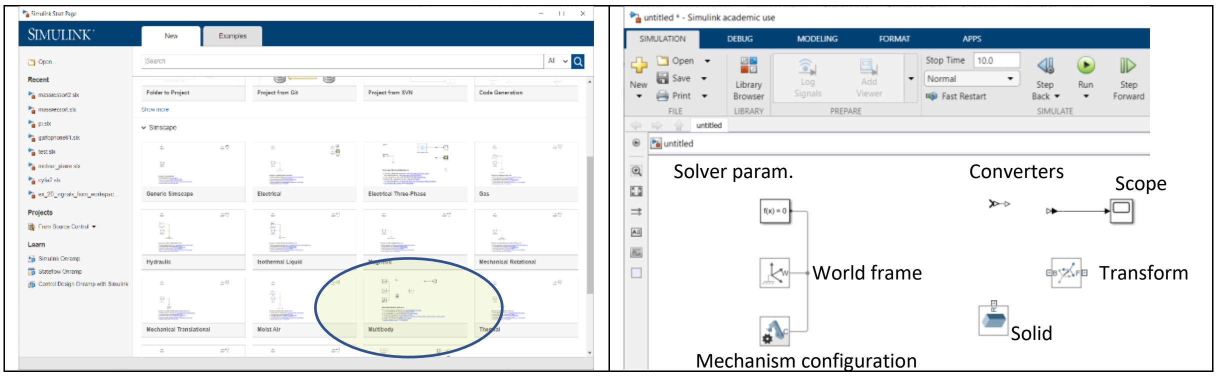
Thanks to these both toolboxes, the links between 2 blocks convey a physical signal instead of mathematical signal (scalar, matrix, vector, …). It means the units must be defined from the previous or next block parameter. To be able to solve the mechanical relations given by each block, it is necessary to close the diagram from a reference to another one (likely electrical potentials). It implies that the links signals are homogeneous to power generated by the product of a flow variable (linear / rotational speed or current for electric model or flow for fluid mechanics) and a force variable (force / torque or voltage for electric or fluid pressure for fluid mechanics).



All the common blocks of Simulink (display, scope, sources, gain…) are connectable to Simscape block provided you convert the Physical signal to Mathematical signal and reciprocally thanks to converters blocks (**Simulink Library / Simscape / Utilities / PS Simulink Converter or Simulink PS Converter**).

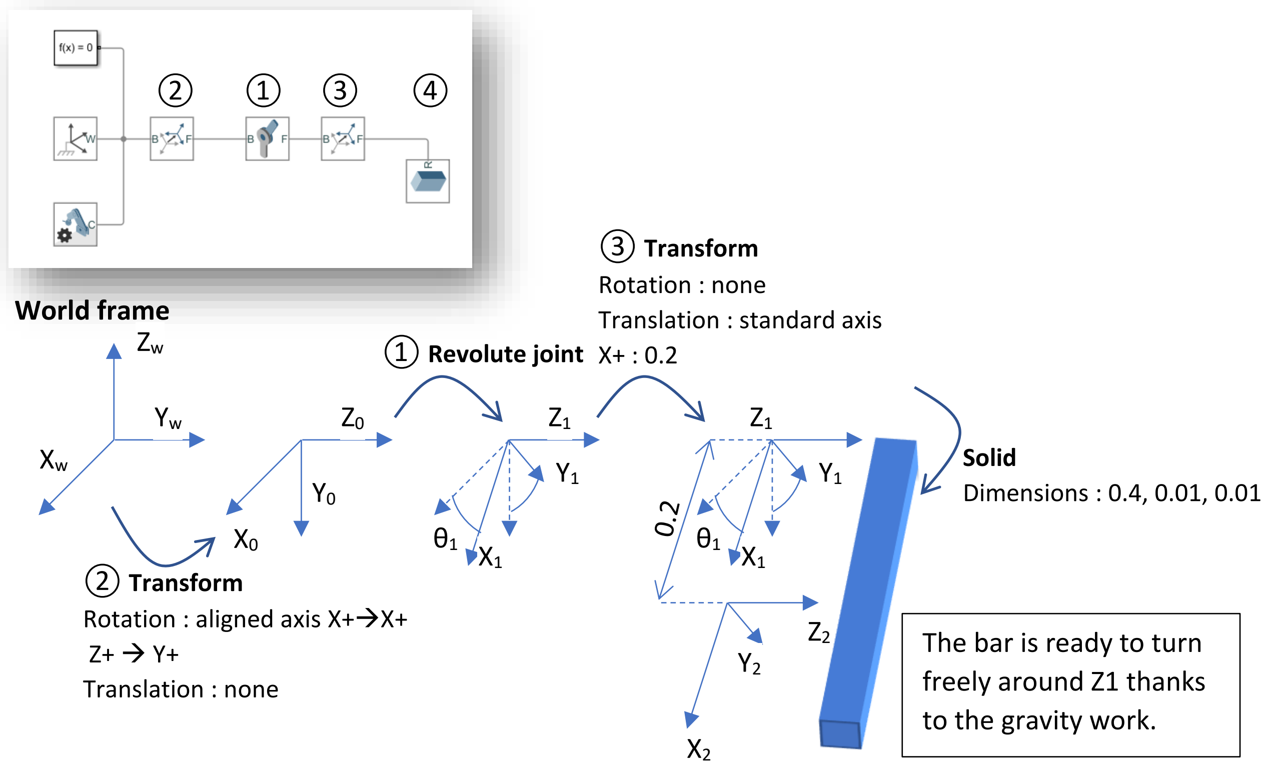
Because we focus on how to model mechanical chain with a kinematic point of view (Simscape Driveline or Simscape electrical toolboxes do it with a power transmission point of view), we will create a new Simscape multibody model (it already includes common and useful blocks which we could add to a blank model).

2 - First model of a kinematic chain



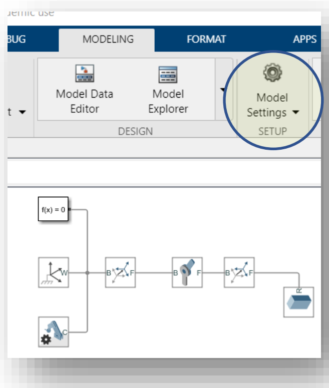
Launch Matlab and then click Simulink button to choose Multibody in order to open a new model. The new model already contains the required **Solver configuration** block (Simscape / Utilities / as many solver parameter block as kinematic chains), a **World Frame** block (Simscape / Multibody / Frames and Transforms / it can be used several time in the model), the **Mechanism configuration** block (Simscape / Multibody / only one block in the model to define the gravity vector for example), a couple of **converters** blocks as mentioned above, a **Rigid Transform** block (Simscape / Multibody / Frames and Transforms / this block will be used many times), a **Scope block** (Simulink / Sinks / very common), and a **Solid** block (Simscape / Multibody / Body Elements / it defines the solid parameters ; i.e. body or link in robotics ; used as many times as number of bodies).

A kinematic chain is a series alternatively composed of bodies and joints. We will start with the modelling of a free pendulum. ① To do so, insert a **Revolute joint** (Multibody / joints /). The vertical axis of the World frame is Z by default, and because of DH principle the follower frame, {X_1, Y_1, Z_1}, of a revolute joint turns around Z axis of the base frame, {X_0, Y_0, Z_0}. ② So we insert a rigid Transform to turn Z_0 into Y_w and thus create a revolute joint with horizontal axis. ③ Insert a new rigid transform to place a frame at the position of the centre of gravity of the next body. ④ Deploy the body from its centre of gravity with the desired dimensions.



Run Your model is now ready to be simulated, so run the simulation and observe the oscillating behaviour of the body in the **Mechanics Explorer Window**.

<u>Improvements of your model</u>:



a) Modify the **Model Settings** to set the Max Step Value to 0.01 s. It makes the model run more smoothly.

b) Open the **Revolute joint** block and add a sensing output for the position. Then connect it to the **Scope** via the **Converter PS-S**. After simulation, open the Scope to see the sinusoidal signal.

c) Open the Revolute joint block and **Specify a State Target Position** to 30° as initial value for the angle θ_1 and a **Damping Coefficient** to 0.01 N.m/(rad/s) as **Internal Mechanics** (mind the units!). Run the simulation and observe the new behaviour of the body via the video rendering (Mechanics Explorer Window) and the scope as well.

In next steps, we will have to specify the actuation modes to the joints thanks to input signals that will describe the desired trajectory (because we work on kinematics and not dynamics here).

3 - Modelling of a hexapod leg

Based on the exercise we worked on the last session, we will model the leg to simulate its behaviour and check whether it can generate a forward step without wobbling the whole robot body.

Answers to questions 1, 2, 3 of the exercise are required to start here the simulation.

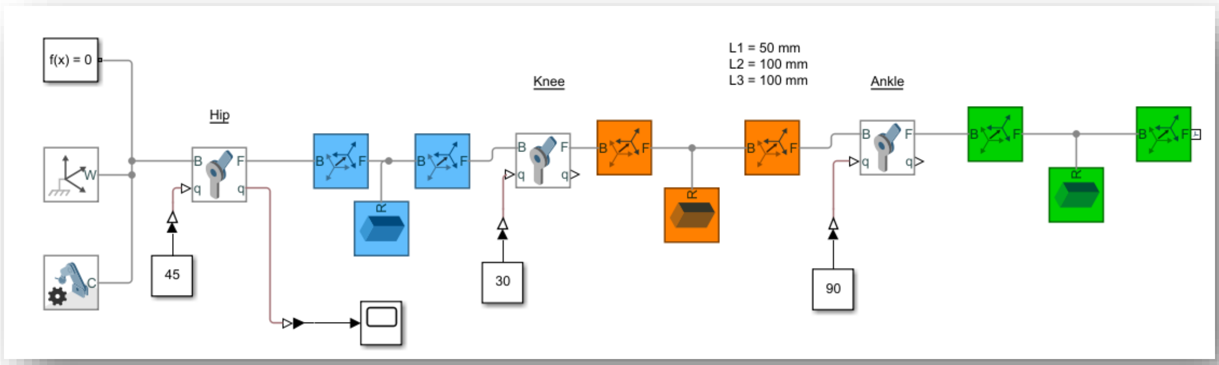
a) Modelling

The model of the kinematic chain of the leg is supposed to include one base frame fixed on the robot body, one frame fixed on the foot (tip of the leg, but it is not necessary), and probably 3 other intermediate frames to express the 3 joints contributions.

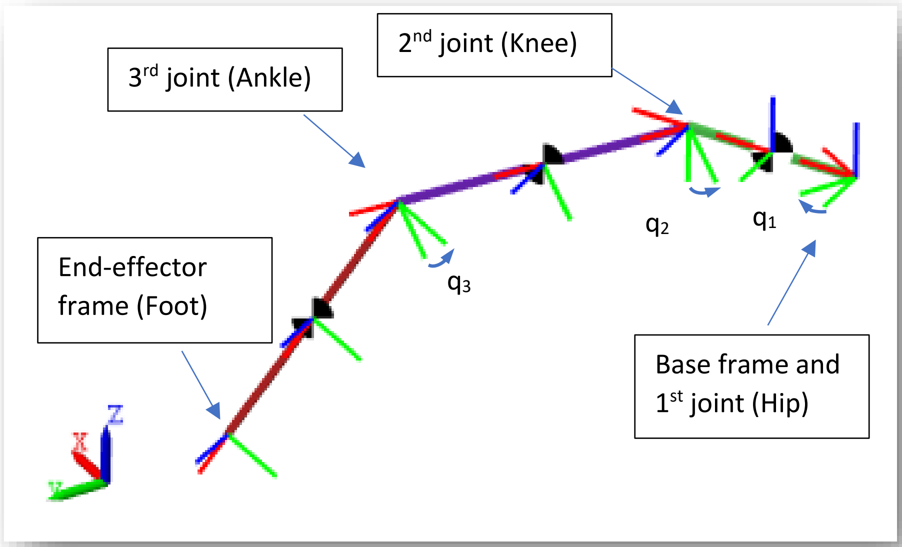
Note that for the simulated model, we will create many more frames to describe the geometry of each solid (at least one frame on the gravity centre of the concerned body, and another one on the tip of the body if linked to a joint). Do not be confused with these additional frames that do not appear in the modelling you created according to Khalil and DH principle.

Hint: Keep an eye on your 3D model of the kinematic chain (question #2 of the exercise) to create the digital model with the same configuration related to the 3 dof q_1, q_2, q_3.

Digital model



Rendering in Mechanics Explorer Window



b) Jacobian

Each joint can provide its current position as output signal (parameter of the joint related to "**Sensing**"). These current values are necessary to compute the Jacobian matrix at each step time. According to the desired motion of the foot with the respect to the base frame, the inverse of the Jacobian matrix will give the joint velocities.

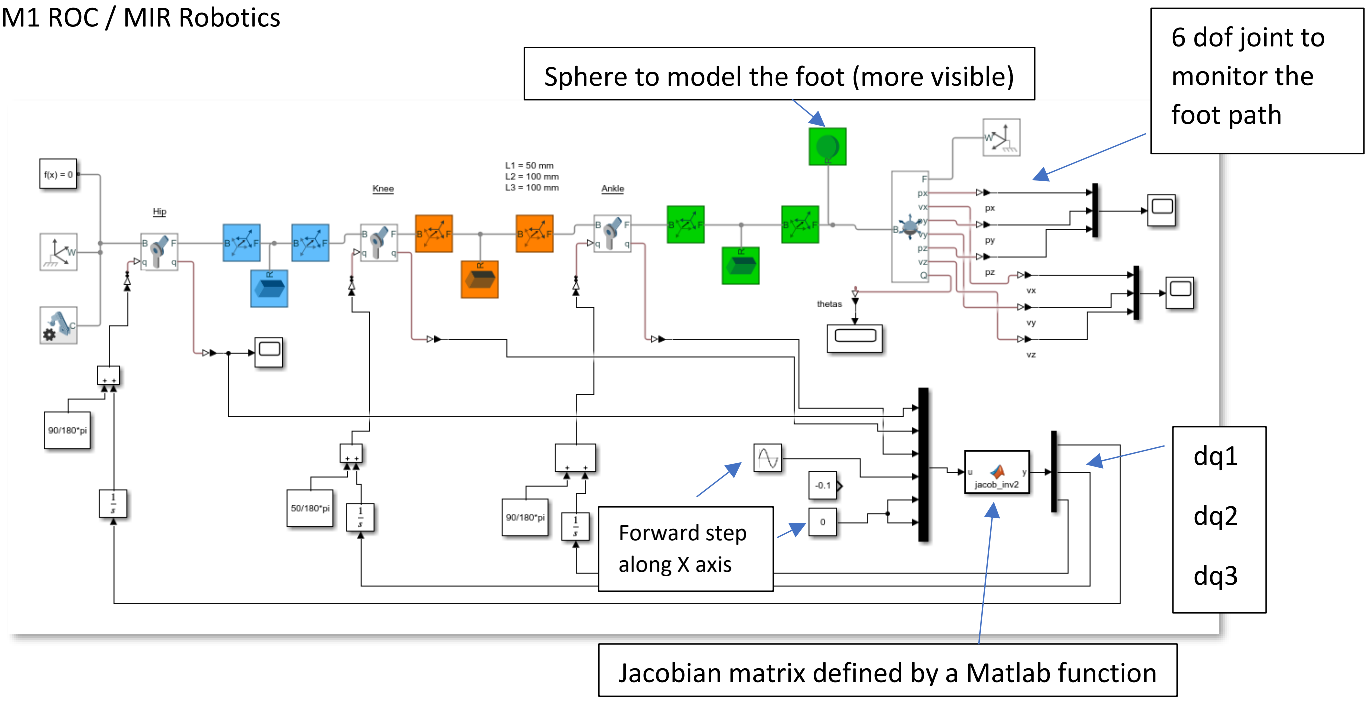
So we will create a **Matlab function** (Simulink library / User defined functions / Matlab Function) which takes the joint positions as the first 3 input signals (q_i to build the Jacobian) and the desired motion (speed) of the foot relative to the base frame as the last 3 input signals (dX_j). The Jacobian matrix and its inverse are computed within the function, then the product of the inverse Jacobian and the dX_j give the joint speeds as the 3 output signals of this function.

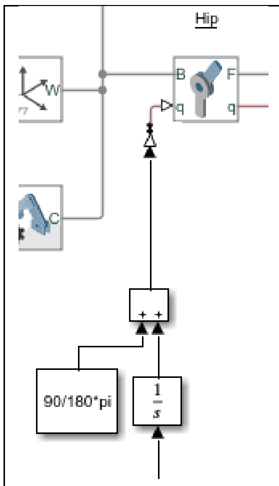
Connect these signals to the actuation input ports of each joint (see the figure on the next page) to make your robot move as desired (expected at least!).

Matlab function

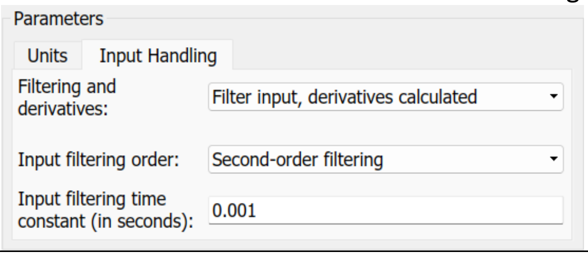
```
function y = jacob_inv2(u)
L1 = 50;
L2 = 100;
L3 = 100;

"u(i) are the inputs"

q1=u(1);s1=sin(q1);c1=cos(q1);
q2=u(2);s2=sin(q2);c2=cos(q2);
q3=u(3);s3=sin(q3);c3=cos(q3);
c23=cos(q2+q3);
s23=sin(q2+q3);

dx=u(4);
dy=u(5);
dz=u(6);

jacob3=[blablabla  blablabla  blablabla_again;...
    more_blablabla   blablabla  always_blablabla ; ...
    still_blablabla   only_blablabla finally_blablabla];

"y is the output vector i.e. joint speeds"

y = (inv(jacob3))*[ dx ; dy ; dz ]
```

Hints about diagram are given next page to help you to run the simulation of the hexabot leg and to answer the original question… Can we control the leg to generate a strictly forward step (without generating any other motion to the whole robot body)?


Sphere to model the foot (more visible)
6 dof joint to monitor the foot path
dq1
dq2
dq3
Forward step along X axis
Jacobian matrix defined by a Matlab function
f(x) = 0
Hip
Knee
Ankle
L1 = 50 mm
L2 = 100 mm
L3 = 100 mm
90/180*pi
50/180*pi
90/180*pi
thetas
px
py
pz
vx
vy
vz
-0.1
0
jacob_inv2



Hint:
To control the robot thanks to the Jacobian matrix (its inverse actually), provide each joint with an input signal of position defined in time.
The joint trajectory has to be defined in position, speed and acceleration (to help Matlab to compute the required torque thanks to the dynamics law). To do so in the **S-PS Converter block (double arrow)**, choose Filter input, derivatives calculated and Second-order filtering.


Parameters

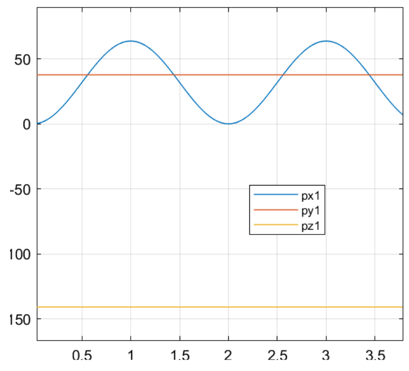
Units | Input Handling

Filtering and derivatives: Filter input, derivatives calculated

Input filtering order: Second-order filtering

Input filtering time constant (in seconds): 0.001

Foot path



This graphic depicts the foot trajectory in time with the respect of the 3 axis of the base frame when a sinusoidal speed along X axis is desired.