

# Belt drive simulation

April 27, 2023



This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 860124.

## Contents

# 1 Description of belt drive simulation

The examined belt drive has two pulleys  $P_1$  and  $P_2$  with identical radius and inertia, see the geometrical setup in Figure ???. The numerical modeling of the belt is based on the Absolute Nodal Coordinate Formulation (ANCF), [?]. The pulleys are simulated as rigid bodies while the contact between belt and pulleys is modeled as described in [?].

This numerical example is similar to the one developed in [?] with some modifications which attempt to eliminate the vibrations in the beginning of the simulation and allow the system to reach the steady state. The angular velocity of pulley  $P_1$  is prescribed by means of an algebraic constraint, while some resistance torque over time is added to pulley  $P_2$ , see the description hereafter. The

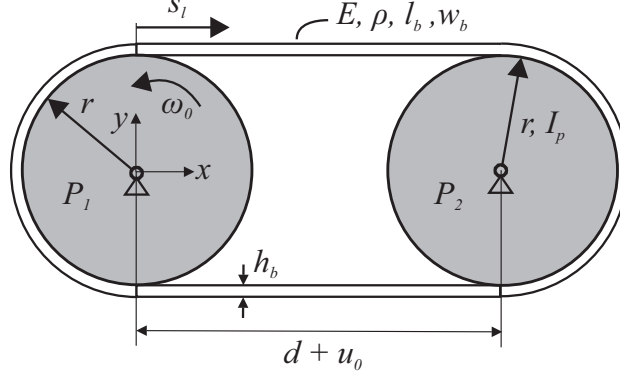


Figure 1: Belt drive with two pulleys, displaced from initial position by  $u_0$ .

Table 1: Main parameters for the belt drive.

Par.	Value	Units	Description	Names in code
$r$	0.09995	m	pulley radius	<code>radiusPulley</code>
$d$	$0.1\pi$	m	distance between two pulleys	<code>distancePulleys</code>
$h_b$	0.0001	m	belt height	<code>hc</code>
$w_b$	0.08	m	belt width	<code>b</code>
$\bar{l}_b$	$0.38\pi$	m	stress-free belt length	(computed)
$l_b$	$0.4\pi$	m	initial, deformed belt length	(computed)
$\varepsilon_{ref}$	-0.05	-	added reference axial strain	<code>preStretch</code>
$EA$	8000	N m	axial stiffness	<code>EA</code>
$EI$	$\frac{4}{3} \cdot 10^{-3}$	N m <sup>2</sup>	bending stiffness	<code>EI</code>
$\rho$	1036	kg/m <sup>3</sup>	beam density	<code>rhoA</code>
$dEA$	1	N/ms <sup>2</sup>	strain proportional damping	<code>dEA</code>
$\omega_{P1}$	12	rad s <sup>-1</sup>	angular velocity of $P_1$	<code>omegaFinal</code>
$d_{P2}$	2	Nm/s	damping at $P_2$	<code>rotationDampingWheels</code>
$t_0$	0.05	s	driving start time	<code>tAccStart</code>
$t_1$	0.60	s	driving end time	<code>tAccEnd</code>
$t_{\tau 0}$	1.0	s	torque $\tau_{P2}$ starts	<code>tTorqueStart</code>
$t_{\tau 1}$	1.5	s	torque $\tau_{P2}$ reaches nominal value	<code>tTorqueEnd</code>
$I_p$	0.25	kg m <sup>-2</sup>	moment of inertia of pulleys	<code>wheelInertia</code>
$g$	9.81	m s <sup>-2</sup>	gravity	<code>gVec</code>

belt is modeled as Bernoulli-Euler beam with bending stiffness  $EI$ , axial stiffness  $EA$ , rectangular cross section with height  $h_b$  and width  $w_b$ , as well as stretch proportional damping, density, and further parameters given in Table ??. A constant acceleration is prescribed to pulley  $P_1$  between  $t_0$  and  $t_1$ :

$$\omega_{P1}(t) = \begin{cases} 0 \frac{\text{rad}}{\text{s}}, & \text{if } t < t_0 \\ \omega_{P1} \frac{t-t_0}{t_0-t_1} & \text{if } t_0 < t < t_1 \\ \omega_{P1} & \text{else.} \end{cases} \quad (1)$$

A torque proportional to the angular velocity is applied to the pulley  $P_2$  which represents

Table 2: Default values for parameters

Par.	Value	Units	Description	Name in code
$t_{end}$	2.45	s	evaluation time	P.tEnd
$\mu$	0.5	-	dry friction coefficient	P.dryFriction
$n_e$	240	-	number of elements	P.nANCFnodes
$dt$	$5 \cdot 10^{-5}$	s	time step size	P.stepSize
$n_{seg}$	4	-	number of segments	P.nSegments
$k_c$	$4 \cdot 10^9$	N/m <sup>3</sup>	normal contact stiffness	P.contactStiffnessPerArea *40
$\mu_k$	$5 \cdot 10^9$	N/m <sup>3</sup>	tangential contact stiffness	P.frictionStiffnessPerArea
$d_c$	$8 \cdot 10^4$	Ns/m <sup>3</sup>	normal contact damping	contactDamping
$\mu_v$	$\sqrt{m_{seg}\mu_k} \approx 3.22 \cdot 10^6$	Ns/m <sup>3</sup>	tangential contact velocity penalty	frictionVelocityPenalty

damping of rotational motion:

$$\tau_{P2}(t) = \begin{cases} 0 \text{ Nm,} & \text{if } t < 1 \\ 25 (0.5 - 0.5 \cdot \cos(2(t-1)\pi)) \text{ Nm} & \text{if } 1 < t < 1.5 \\ 25 \text{ Nm} & \text{else.} \end{cases} \quad (2)$$

As compared to [?], we use a much smaller belt height  $h_b$  in order to exclude bending effects, a higher pre-tension (due to pre-stretch), while keeping the axial stiffness  $EA$  the same. Furthermore, the bending stiffness is lowered by a factor of 50, which reduces bending effects, as it would lead to significant deviations from an analytical solution otherwise. The support of pulley  $P_1$  is not displaced during the first 0.05s of the simulation, but the pre-stretch  $\varepsilon_{ref}$  is applied before running a static computation, which defines a static equilibrium for the dynamic simulation hereafter. The contact stiffness has been increased by a factor of 40 and a tangential stiffness (bristle) model has been included in order to retrieve highly accurate contact behavior.

## 2 Description of code

For simulating the system we are using the multibody dynamics code Exudyn [?], see the documentation of Exudyn<sup>1</sup>. The code is divided into sections (1, 2, ..., 8) and subsections (A, B, ...) for easier documenting and processing:

- In section 1, we import necessary modules.
- Section 2 creates a multibody system, `mbs`.
- Section 3 consists of the Parameter Function. This function will be repeatedly called from Parameter Variation to update the value of the variables for which we perform variations.
  - We create a class `P` which contains all parameters for which we can perform Parameter variations. First the parameters are given their default values, see Table ???. Then we update the values of varying parameters through:

```
for key,value in parameterSet.items():
    setattr(P,key,value)
```

where `setattr()` is a Python function which sets the value of the attribute of an object.

- We create the model with respect to the parameter values given in Table ???.
- For prescribing the angular velocity, we are using the following user function:

```
def UFvelocityDrive(mbs, t, itemNumber, lOffset):
    if t < tAccStart: # driving start time
        v = 0
    if t >= tAccStart and t < tAccEnd:
        v = omegaFinal/(tAccEnd-tAccStart)*(t-tAccStart)
    elif t >= tAccEnd:
        v = omegaFinal
    return v
```

<sup>1</sup><https://github.com/jgerstmayr/EXUDYN>

Table 3: Input for ObjectANCFcable2D

Input	Value
physicsMassPerLength	$\rho A$
physicsBendingStiffness	$EI$
physicsAxialStiffness	$EA$
physicsBendingDamping	$dEI$
physicsAxialDamping	$dEA$
physicsReferenceAxialStrain	$\varepsilon_{ref}$
physicsReferenceCurvature	0
useReducedOrderIntegration	2
strainIsRelativeToReference	False

- For the ANCF beam elements modeling the belt we are using `ObjectANCFcable2D`, see the documentation of Exudyn<sup>2</sup>, theDoc. The input of `ObjectANCFcable2D` is given in Table ??.
- During the simulation we measure the angular velocity and torque for both pulleys over time, as well as, the axial velocity, the tangential contact stresses, the normal contact stresses and the axial forces over the length of the belt. For this we use `mbs.AddSensor()`. For saving the solution in a different file for each parameter variation we name the solution files using a string which is generated according to the used values for the parameters which can vary:

```

fileClassifier = ''
fileClassifier += '-tt'+str(int(P.tEnd*100))
fileClassifier += '-hh'+str(int(P.stepSize/1e-6))
fileClassifier += '-nn'+str(int(P.nANCFnodes/60))
fileClassifier += '-ns'+str(P.nSegments)
fileClassifier += '-cs'+str(int((P.contactStiffnessPerArea/1e7)))
fileClassifier += '-fs'+str(int((P.frictionStiffnessPerArea/1e7)))
fileClassifier += '-df'+str(int(P.dryFriction*10))
fileClassifier += '-'

```

For example, for measuring and saving the torque applied to  $P_1$  we use:

```

sTorquePulley0 = mbs.AddSensor(SensorObject(objectNumber=
    velControl, fileName=fileDir+'torquePulley0'+fileClassifier+'.
    txt', outputVariableType=exu.OutputVariableType.Force))

```

- In section 4, simulation settings and visualization settings are defined.
- In section 5, we perform the static and dynamic simulation. We use `mbs.SetObjectParameter` to change objects' parameters after `mbs.Assemble`. This allows us to change the value of some parameters such as `frictionCoefficient`, `frictionStiffness` during the static simulation and to activate or deactivate constraints before and after the static simulation. For example, before the static simulation we deactivate the constraint which is used for prescribing the angular velocity, `velControl`, by:

```

mbs.SetObjectParameter(velControl, 'activeConnector', False)

```

After the static simulation we activate it again. Note also that we set `updateInitialValues=True` in

```

exu.SolveStatic(mbs, simulationSettings, updateInitialValues=True)

```

which allows us to use the static solution as the initial solution for the dynamic simulation.

- In section 6, the obtained results are post-processed and saved in files.
- In section 7, one can choose between performing single simulation and performing parameter variation. The option for plotting figures can be chosen as well. All solutions from parameter variations have already been added in the solution folder. Solutions from new runs are stored by default in `solutionNosync`.

<sup>2</sup><https://github.com/jgerstmayr/EXUDYN>

- In section 8, stored results are plotted. Cases given in `iCases = [1, ..., 4]` correspond to different varying quantities; number of elements, step size, other quantities (number of segments, normal contact stiffness, tangential contact stiffness, dry friction coefficient) and evaluation time.

## 3 Installation and running

### 3.1 Installing Python and Exudyn

The code was tested in a Windows pc using Anaconda, 64bit, Python 3.7.6 and Spyder 4.0.1 which is included in the Anaconda installation.

Exudyn was installed using PIP INSTALLER (pypi.org). Pre-built versions of Exudyn are hosted on [pypi.org](https://pypi.org), see the project

- <https://pypi.org/project/exudyn>

For installing Exudyn using pip, as with most other packages, in the regular case (if your binary has been pre-built) you just need to do<sup>3</sup>

```
pip install exudyn
```

On Linux (currently only pre-built for UBUNTU, but should work on many other linux platforms), **update pip to at least 20.3** and use

```
pip3 install exudyn
```

For pre-releases (use with care!), add '`--pre`' flag:

```
pip install exudyn --pre
```

Results added in src folder were obtained using Exudyn V1.2.32.dev1. For installing this version do

```
pip install exudyn==1.2.32.dev1
```

For more information for installing Exudyn see the theDoc<sup>4</sup>.

### 3.2 Running the code

Two python files are added in src folder. One for performing the belt drive simulation with the default values and another for performing variations and plotting figures. (The two files are identical with the only differences being in the flags which are enabling the operations of the code.)

For running these files the first option is to open an Anaconda prompt and copy paste the file location. The second option is to use Spyder and should be selected for making modifications in the code.

---

<sup>3</sup>If the index of pypi is not updated, it may help to use `pip install -i https://pypi.org/project/ exudyn`

<sup>4</sup><https://github.com/jgerstmayr/EXUDYN>