Belt drive simulation

May 10, 2023

THREAD
EUROPEAN TRAINING NETWORK

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 Belt drive simulation

In this section, we investigate the dynamics of a belt drive system featuring two pulleys, P_1 and P_2, both with identical radius and inertia, as illustrated in Figure 1. The numerical modeling of this system relies on the Absolute Nodal Coordinate Formulation (ANCF), with the pulleys simulated as rigid bodies and the belt-pulley contact modeled as per [1]. The repository for the code related to this section is located at:

`https://github.com/THREAD-2-3/beltDriveSimulation`

1.1 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 1. The numerical modeling of the belt is based on the Absolute Nodal Coordinate Formulation, [2]. The pulleys are simulated as rigid bodies while the contact between belt and pulleys is modeled as described in [1].

This numerical example is similar to the one developed in [3] 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

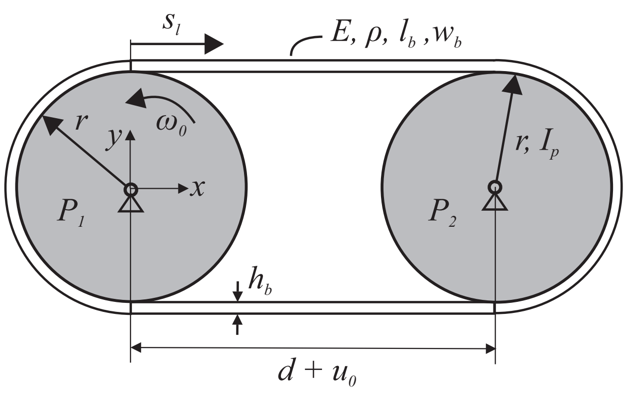

Figure 1: Belt drive with two pulleys, displaced from initial position by u_0.

Table 1: Main parameters for the belt drive.

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

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

Table 2: Default values for parameters

Par.	Value	Units	Description	Name in code
t_{end}	2.45	s	evaluation time	`P.tEnd`
μ	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^3	normal contact stiffness	`P.contactStiffnessPerArea *40`
μ_k	$5 \cdot 10^9$	N/m^3	tangential contact stiffness	`P.frictionStiffnessPerArea`
d_c	$8 \cdot 10^4$	Ns/m^3	normal contact damping	`contactDamping`
μ_v	$\sqrt{m_{seg}\mu_k} \approx 3.22 \cdot 10^6$	Ns/m^3	tangential contact velocity penalty	`frictionVelocityPenalty`

further parameters given in Table 1. A constant acceleration is prescribed to pulley P_1 between t_0 and t_1:

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

A torque proportional to the angular velocity is applied to the pulley P_2 which represents damping of rotational motion:

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

As compared to [3], 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 0.05 s of the simulation, but the pre-stretch ε_{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.

1.2 Description of code

For simulating the system we are using the multibody dynamics code Exudyn [4], see also the GitHub repository of Exudyn[1]. The code[2] 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 `ParameterFunction(...)`. This function will be repeatedly called from `ProcessParameterList()` 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 2. 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 1.

[1]https://github.com/jgerstmayr/EXUDYN
[2]https://github.com/THREAD-2-3/beltDriveSimulation/blob/main/src/beltDriveParameterVariation.py

Table 3: Input for ObjectANCFCable2D

Input	Value
`physicsMassPerLength`	ρA
`physicsBendingStiffness`	EI
`physicsAxialStiffness`	EA
`physicsBendingDamping`	dEI
`physicsAxialDamping`	dEA
`physicsReferenceAxialStrain`	ε_{ref}
`physicsReferenceCurvature`	0
`useReducedOrderIntegration`	2
`strainIsRelativeToReference`	False

- 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
```

- For the ANCF beam elements modeling the belt we are using `ObjectANCFCable2D`, see the documentation[3] of Exudyn [4]. The input of `ObjectANCFCable2D` is given in Table 3.
- 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/1
    e7)))
fileClassifier += '-fs'+str(int((P.frictionStiffnessPerArea/1
    e7)))
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

[3]`theDoc.pdf`: `https://github.com/jgerstmayr/EXUDYN/blob/master/docs/theDoc/theDoc.pdf`

of some parameters such as `frictionCoefficient`, `frictionStiffness` during the static simulation and to activate or deactivate constrains 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.
- 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.

1.3 Installation and running

1.3.1 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 (Pip Installs Packages). Pre-built versions of Exudyn are hosted on `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

```
pip install exudyn
```

On Linux (only tested on UBUNTU 18.04 and 20.04, but should work on many other Linux platforms), **update pip to at least 20.3** and use

```
pip3 install exudyn
```

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[4].

1.3.2 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. Note that, 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 use an Anaconda prompt, following the steps:

[4] `https://github.com/jgerstmayr/EXUDYN`

- Open an Anaconda prompt.
- Navigate to the directory containing the Python file: Use the "cd" command to change the current directory to the one that contains the Python file.
- Run the Python file using the command "python beltDriveSingleRun.py" for performing a single run or "python beltDriveParameterVariation.py" for performing parameter variations.

The second option, which enable the user to edit and modify the code, is to use a Python environment. We recommend Spyder, see 1.3.1.

1.4 Conclusions

The repository related to the current section provides the code necessary for reproducing the belt drive simulation and obtaining identical results. This, in combination with the respective section of the deliverable D2.2, provides a full understanding of the implemented methods. The open-source code is accompanied by a detailed description of the experimental setup, explanatory comments, and examples to aid in comprehending the code, as well as instructions for installation.

Note, that the belt drive simulation is based on the open-source multibody dynamics Exudyn. This multibody dynamics code is extensive and has been developed outside of the THREAD project, hence was not directly included in the project repository. However, we provide the necessary description for running and understanding the code for the belt drive simulation.

As a further step, we consider adding the code used for other numerical investigations published in publications related to THREAD to the THREAD repository; `https://github.com/THREAD-2-3`.

References

[1] K. Ntarladima, M. Pieber, and J. Gerstmayr, "Contact modeling between axially moving beams and sheaves," under submission.

[2] J. Gerstmayr and H. Irschik, "On the correct representation of bending and axial deformation in the absolute nodal coordinate formulation with an elastic line approach," *Journal of Sound and Vibration*, vol. 310, no. 3, pp. 461–487, 2008.

[3] A. Pechstein and J. Gerstmayr, "A Lagrange-Eulerian formulation of an axially moving beam based on the absolute nodal coordinate formulation," *Multibody System Dynamics*, vol. 30, no. 3, pp. 343–358, 2013.

[4] J. Gerstmayr, "Exudyn – A C++ based Python package for flexible multibody systems." preprint, Research Square, `https://www.researchsquare.com/article/rs-2693700/v1`, 2023.