Introduction to computational forward dynamics with CarlaTheKraken – first draft revision

1. Introduction

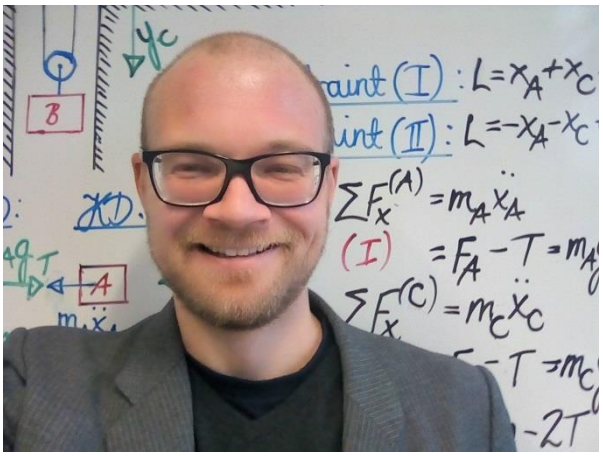
The current document contains a description of the objectives of a Matlab based computer code for computational forward dynamics. Furthermore, the theoretical background for the implemented methods is described in order for the document to function as lecture notes for undergraduate courses on dynamics and machine dynamics.

1.1. The standard formalities

If you are a lecturer …

If you are a student, remember the first three rules required for learning mechanics:

1. Come to the lectures, go to the exercises
2. Come to the lectures, go to the exercises
3. Mechanics is NOT hard, but if it does not hurt when you start learning something new, you are probably not doing it right



Regards,

Prof. Niels Højen Østergaard

Engineering Mechanics
Hochschule Rhein-Waal
niels.ostergaard@hochschule-rhein-waal.de

Table of Content

1.2. The purpose of forward dynamics

Forward dynamics refers to analysis of rigid bodies, where motions are calculated on basis of given forces. These forces may for unconstrained motions be caused by springs, gravity, friction, loads or contacts and for constrained motions additionally by dynamic reactions in pins, traces etc. This approach to dynamics is common in mechanical engineering and is furthermore used in computer simulations (the GTA computer games are an example of open world forward dynamics). These notes concern the theoretical framework required to perform computational forward dynamics of rigid body systems along with an introduction to a general purpose MBD-engine.

Contrary to forward dynamics, inverse dynamics refers to analyses of rigid body motions, for which forces are determined on basis of a prescribed motion. Inverse dynamics will not be considered in the current notes[1].

1.3. CarlaTheKraken Computational RBD

1.3.1. Objective (why are we doing this)

CarlaTheKraken refers to a set of Matlab scripts constituting an engine for computational dynamics with particular focus on forward rigid body dynamics.
The primary objective of the code is to provide an engine for general purpose problem solving to be applied for teaching of dynamics and machine dynamics at HSRW-T&B. Matlab was chosen as platform for the first code revisions due to the fact, that this is the common language for scientific programming used in the basic programming modules.
The secondary objective is to convert the source code to a free (open source based) platform (for example GNU Octave or Python) and make it available for free download online. This would enable students to download and apply the code for problem solving after graduation. The ambition is on a longer time-scale that the code is used for solving industrial problems individually by HSRW graduates as-well as in cooperation with HSRW.

1.3.2. Current revision (alpha) and future revision (beta)

The current revision of CarlaTheKraken[2] is limited to analyses of 2D problems. The following main functionalities have been implemented:

- Circle, line and general (tri-shaped) body elements
- Circle and line world elements
- Rotational and translational springs and dampers
- Spherical bounding boxes for hierarchical contact detection
- Point-to-point and point-to-line contacts
- Pin joints
- Variable gravity and constant external force functions
- Animation and response plotting functions
- 'Rotating world' elements
- Fixed and variable time step higher order integration

Warning and disclaimer !!!

Please note that there's absolutely no warranty for the validity of the code and the obtained results. As the matter of fact, one of the only things guaranteed is that the code still contains bugs. Furthermore, do note that the code to a wide extend still is based on global variables. Though computer scientists are right when arguing, one may cause terrible damage using those, these actually are quite efficient for mechanical engineers, who are accustomed to writing few long scripts rather than a large set of smaller segments and functions.
Currently, a pre-implementation of unconstrained 3D mechanics has been conducted. This is not yet available for downloads and is not described in this document. However, we did do some damage with Carla older sister [6].

[1] For HSRW students: an introduction to inverse dynamics is given in Prof. Brandt's slides on MBD
[2] Matlab source code available for free download here

1.3.3. The name

The idea of a kraken living in the Spoy-Canal outside Building 8 was or originally pinched by a 3rd semester student during the mandatory dynamics course for mechanical engineers in the fall of 2016. The notion of the kraken turned out to be so catchy and fabolous, that it was caught on photo[3], considered brought to life using augmented reality and attempted 3D printed[4]. Eventually, a computer code is now named after the kraken.

The name Carla was adopted from a female mechanic in the local industry, who has been of great help to Carla's father (the author of the current document). The real Carla shall remain anonymous for her own protection, but she's just as fabulous as the kraken.

CarlaTheKraken is a code written by mechanical engineers for mechanical engineers without ever as much as reading a style-guide. The code therefore is and will always remain 'a monster' from a computer science perspective. If you have a problem with this, go bother some computer scientist.

2. Kinematics and body coordinates

Carla the Kraken lives in the Canal outside building 8 and is familiar with the variety of strange beings and curios objects which are common below the water surface

Firstly, the kinematics of a rigid body will be considered. In the classical mechanics framework taught in the introductory dynamics course, the principle of relative motion was applied at velocity and acceleration level for two arbitrarily chosen points inside a rigid body. On the most general form, any plane motion of a rigid body was decomposed into a translation and a rotation. On equation form, the following vectorial formulation was derived

$$\boldsymbol{v}_B = \boldsymbol{v}_A + \boldsymbol{v}_{B/A} \qquad \boldsymbol{a}_B = \boldsymbol{a}_A + \overbrace{(\boldsymbol{a}_{B/A})_t}\ \overbrace{(\boldsymbol{a}_{B/A})_n} \tag{1.}$$

Directions:	$\perp \lvert AB \rvert$	$\perp \lvert AB \rvert$	From A to B	
Magnitudes:	$\lVert \omega l_{AB} \rVert$	$\lVert \omega^2 l_{AB} \rVert$	$\lVert \alpha l_{AB} \rVert$	

[3] Carla the Kraken caught on photo

[4] Partially completed 3D print

While this formulation of rigid body kinematics worked well for manual calculations, it's not really an efficient formulation for implementation in numerical simulation codes. An alternative formulation on a more appropriate form utilizing linear algebra will therefore now be derived. Initially, the coordinates of the the CoG of a rigid body, *x* and *y*, will be considered along with the rotation angle θ, see Figure 1. Note that θ is defined with respect to the positive x-axis and is positive in the counter-clockwise direction. These are functions of time and were this far considered separately, but will now be organized into a vector of generalized coordinates called $(\boldsymbol{q})$. Recalling that the angular velocity is given by $\omega = \dot{\theta}$ and the angular acceleration is $\alpha = \dot{\omega} = \ddot{\theta}$, time differentiation of $\boldsymbol{q}$ gives

$$(\boldsymbol{q}) = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \begin{pmatrix} \boldsymbol{r} \\ \theta \end{pmatrix} \qquad (\dot{\boldsymbol{q}}) = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \boldsymbol{v} \\ \omega \end{pmatrix} \qquad (\boldsymbol{a}) = (\ddot{\boldsymbol{q}}) = \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{pmatrix} = \begin{pmatrix} \boldsymbol{a} \\ \alpha \end{pmatrix} \tag{2.}$$

This will be our preferred way of structuring data in the following.

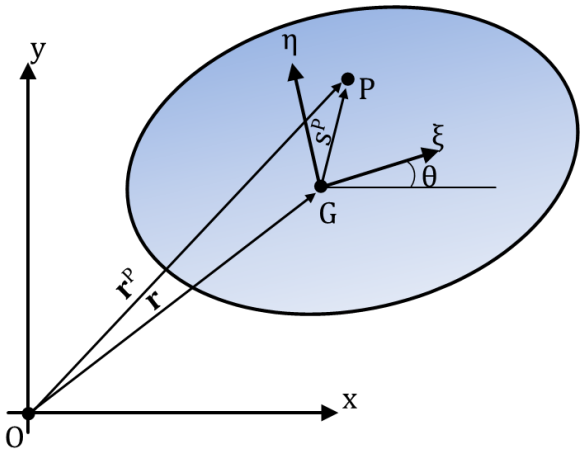


Figure 1 The plane kinematics of a rigid body

With $\boldsymbol{r}$ specified as the position of the CoG (denoted G) of a rigid body, any point P can is given by

$$(\boldsymbol{r})^{\boldsymbol{P}} = (\boldsymbol{r}) + (\boldsymbol{s})^P_{xy} \tag{3.}$$

In which $(\boldsymbol{s})^P_{xy}$ is the local vector between the points P and G with the global Cartesian coordinate system as basis. However, $(\boldsymbol{s})^P_{xy}$ changes dependent on the spatial orientation of the body. An efficient way of obtaining $(\boldsymbol{s})^P_{xy}$, is to attach a local body frame in G rotating along with the body with the local vectors ξ and η as axes. This has the advantage, that the local vector $(\boldsymbol{s})^P$ will be independent of the spatial orientation, if ξ and η and used as basis. We may rewrite equation 3 on the following form:

$$\begin{aligned} \begin{pmatrix} r_x^P \\ r_y^P \end{pmatrix} &= \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} s_x^P \\ s_y^P \end{pmatrix} \\ &= \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} s_\xi^P \cos\theta - s_\eta^P \sin\theta \\ s_\xi^P \sin\theta + s_\eta^P \cos\theta \end{pmatrix} \\ &= \begin{pmatrix} x \\ y \end{pmatrix} + \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} s_\xi^P \\ s_\eta^P \end{pmatrix} \end{aligned} \tag{4.}$$

The matrix in the last term is recognized as a standard rotation matrix. On shorter form, we write

$$(\boldsymbol{r})^P = (\boldsymbol{r}) + [\boldsymbol{A}](\boldsymbol{s})^P_{\xi\eta} \qquad 5.$$

Time differentiation yields the velocity of point P

$$(\dot{\boldsymbol{r}})^P = \begin{pmatrix} \dot{r}_x^P \\ \dot{r}_y^P \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} + \begin{pmatrix} -s_\xi^P \sin\theta\dot{\theta} - s_\eta^P \cos\theta\dot{\theta} \\ s_\xi^P \cos\theta\dot{\theta} - s_\eta^P \sin\theta\dot{\theta} \end{pmatrix}$$

$$= \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} + \begin{bmatrix} -\sin\theta & -\cos\theta \\ \cos\theta & -\sin\theta \end{bmatrix} \begin{pmatrix} s_\eta^P \\ s_\xi^P \end{pmatrix} \dot{\theta} \qquad 6.$$

$$= \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} + \begin{pmatrix} -s_y^P \\ s_x^P \end{pmatrix} \dot{\theta}$$

On shorter from, equation 6 is written on the form

$$(\dot{\boldsymbol{r}})^P = (\dot{\boldsymbol{r}}) + (\breve{\boldsymbol{s}})^P_{xy}\dot{\theta} \qquad \text{with } (\breve{\boldsymbol{s}})^P_{xy} = \begin{pmatrix} -s_y^P \\ s_x^P \end{pmatrix} \qquad 7.$$

Differentiating with respect to timie once more, provides the accelerations of P. The algebra is not reproduced here[5], but the acceleration expressions are

$$(\ddot{\boldsymbol{s}})^P_{xy} = (\breve{\boldsymbol{s}})^P_{xy}\ddot{\phi} - (\boldsymbol{s})^P_{xy}\dot{\phi}^2 \qquad 8.$$

$$(\ddot{\boldsymbol{r}})^P = (\ddot{\boldsymbol{r}}) + (\breve{\boldsymbol{s}})^P_{xy}\ddot{\phi} - (\boldsymbol{s})^P_{xy}\phi^2$$

In general, we notice that

$$(\dot{\boldsymbol{s}})^P_{xy} = (\breve{\boldsymbol{s}})^P_{xy}\dot{\theta} \qquad 9.$$

This corresponds to, that time differentiation of a local unit basis vector corresponds to a 90 deg counter-clockwise and multiplication of the angular velocity of the frame.

Example 2.1: Kinematic analysis of sliding rod

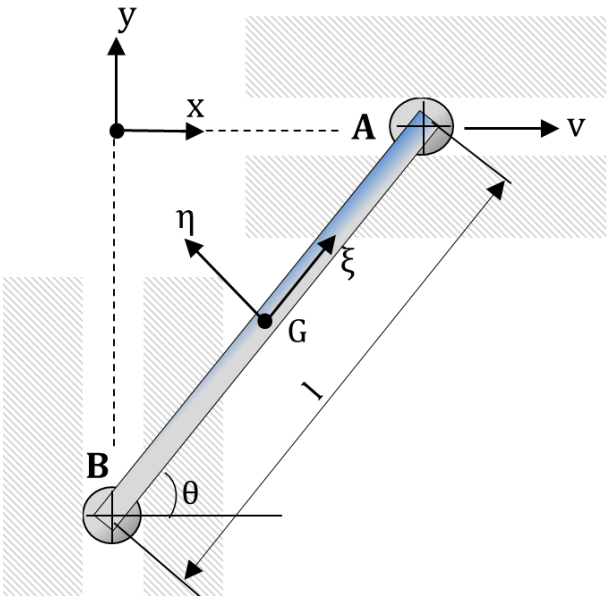


Figure 2: Rod geometry, local body frame and global Cartesian coordinate system

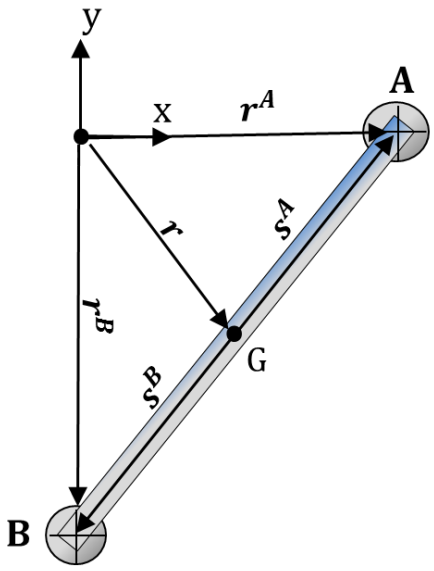


Figure 3: Vectors required for kinematic analysis

[5] The mathematical derivation of equation 8 is contained in Nikravesh's book [1] on pp. 56.

[Example 2.1 continued …]

A rod is in each end point connected to a wheel, which is fixed in traces as shown in Figure 2. The upper right end point moves at a constant horizontal velocity v towards right. Our objective is now to determine the angular velocity of rod and the velocity of the CoG (denoted G). Initially, let's go back to our well-established and field-proven method for solving this type of problems and apply the principle of relative motion from equation 1 at velocity level between the points A and B:

$$\boldsymbol{v}_B = \boldsymbol{v}_A + \boldsymbol{v}_{B/A} \quad \rightarrow \begin{pmatrix} 0 \\ v_B \end{pmatrix} = \begin{pmatrix} v_A \\ 0 \end{pmatrix} + \begin{pmatrix} \omega_{AB} l \sin\theta \\ -\omega_{AB} l \cos\theta \end{pmatrix} \rightarrow \omega_{AB} = -\frac{v_A}{l\sin\theta}$$

$\boldsymbol{v}_B$: ↑ — $\boldsymbol{v}_A$: → v — $\boldsymbol{v}_{B/A}$: θ, $\omega_{AB} l$

Having calculated the angular velocity $\omega_{AB} = \dot{\theta}_{AB}$ without any significant difficulties, we could calculate the velocity of point B if we wanted to. Instead, we move on to calculation of the velocity of G:

$$\boldsymbol{v}_G = \boldsymbol{v}_A + \boldsymbol{v}_{G/A} \quad \rightarrow (\boldsymbol{v})_G = \begin{pmatrix} v_A \\ 0 \end{pmatrix} + \begin{pmatrix} \omega_{AB} \frac{l}{2} \sin\theta \\ -\omega_{AB} \frac{l}{2} \cos\theta \end{pmatrix} = \begin{pmatrix} v_A \\ 0 \end{pmatrix} + \begin{pmatrix} \left(-\frac{v_A}{l\sin\theta}\right) \frac{l}{2} \sin\theta \\ -\left(-\frac{v_A}{l\sin\theta}\right) \frac{l}{2} \cos\theta \end{pmatrix} = \begin{pmatrix} \frac{v}{2} \\ \frac{v}{2\tan\theta} \end{pmatrix}$$

$\boldsymbol{v}_G$: → v — $\boldsymbol{v}_{G/A}$: θ, $\omega_{AB} \frac{l}{2}$

That wasn't hard at all, right? But then again, we have been practicing that for quite some time now. If we want to redo the analysis with our new and fascinating framework, a local $\xi\eta$-frame is attached in G. Measured in the local frame, the end points A and B now have the coordinates

$$(\boldsymbol{s})^A_{\xi\eta} = \begin{pmatrix} \frac{L}{2} \\ 0 \end{pmatrix} \qquad (\boldsymbol{s})^B_{\xi\eta} = \begin{pmatrix} -\frac{L}{2} \\ 0 \end{pmatrix}$$

Transformation to global coordinates can be performed applying the second term in equation 5:

$$(\boldsymbol{s})^A_{xy} = [\boldsymbol{A}](\boldsymbol{s})^A_{\xi\eta} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} \frac{L}{2} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{L}{2}\cos\theta \\ \frac{L}{2}\sin\theta \end{pmatrix} \qquad (\boldsymbol{s})^B_{xy} = [\boldsymbol{A}](\boldsymbol{s})^B_{\xi\eta} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} -\frac{L}{2} \\ 0 \end{pmatrix} = \begin{pmatrix} -\frac{L}{2}\cos\theta \\ -\frac{L}{2}\sin\theta \end{pmatrix}$$

We may now apply equation 7 and calculate the end point velocities

$$(\dot{\boldsymbol{r}})^A = (\dot{\boldsymbol{r}}) + (\check{\boldsymbol{s}})^A_{xy}\dot{\theta} \rightarrow \begin{pmatrix} v^A \\ 0 \end{pmatrix} = \begin{pmatrix} v^G_x \\ v^G_y \end{pmatrix} + \begin{pmatrix} -\frac{L}{2}\sin\theta \\ \frac{L}{2}\cos\theta \end{pmatrix}\dot{\theta} \qquad (\dot{\boldsymbol{r}})^B = (\dot{\boldsymbol{r}}) + (\check{\boldsymbol{s}})^B_{xy}\dot{\theta} \rightarrow \begin{pmatrix} 0 \\ v^B \end{pmatrix} = \begin{pmatrix} v^G_x \\ v^G_y \end{pmatrix} + \begin{pmatrix} \frac{L}{2}\sin\theta \\ -\frac{L}{2}\cos\theta \end{pmatrix}\dot{\theta}$$

These two equations contain the x and y velocity components of G and the angular velocity $\dot{\theta}$ as unknowns. Isolating the velocity of G in both the left and the right equation, and setting these equal provides us with the following expression allowing for calculation of the angular velocity:

$$\begin{pmatrix} v^A \\ 0 \end{pmatrix} - \begin{pmatrix} -\frac{L}{2}\sin\theta \\ \frac{L}{2}\cos\theta \end{pmatrix}\dot{\theta} = \begin{pmatrix} 0 \\ v^B \end{pmatrix} - \begin{pmatrix} \frac{L}{2}\sin\theta \\ -\frac{L}{2}\cos\theta \end{pmatrix}\dot{\theta}$$

Considering the x-components, it follows that

$$(\mathrm{I}) \rightarrow v^A + \frac{L}{2}\sin\theta\dot{\theta} = -\frac{L}{2}\sin\theta\dot{\theta} \rightarrow \dot{\theta} = -\frac{v^A}{l\sin\theta}$$

Having calculated the angular velocity, the linear velocities of G can now be determined:

$$\begin{pmatrix} v^G_x \\ v^G_y \end{pmatrix} = \begin{pmatrix} v^A \\ 0 \end{pmatrix} - \begin{pmatrix} -\frac{L}{2}\sin\theta \\ \frac{L}{2}\cos\theta \end{pmatrix}\dot{\theta} = \begin{pmatrix} v^A \\ 0 \end{pmatrix} - \begin{pmatrix} -\frac{L}{2}\sin\theta \\ \frac{L}{2}\cos\theta \end{pmatrix}\left(-\frac{v^A}{l\sin\theta}\right) = \begin{pmatrix} \frac{v}{2} \\ \frac{v}{2\tan\theta} \end{pmatrix}$$

3. Unconstrained dynamics and numerical integration

Carla the Kraken maintains diplomatic relations with the mighty Ctulhu and ensures that the peace treaty preventing him from taking over the world is renewed on regular basis

In this chapter, a framework for unconstrained dynamics appropriate for numerical analysis will be derived. By unconstrained dynamics, we refer to the dynamics of bodies which are not subjected to fixed geometrical bindings like pins or hinges. In classical dynamics, the equations of motion were derived of Newton's 2nd law and Euler's law of motion

$$F_x = m\ddot{x} \qquad F_y = m\ddot{y} \qquad M = I\ddot{\theta} \tag{10.}$$

Note that summation signs on the left hand side of the equations from now on will not be included, since these make it hard to maintain overview in vector and matrix equations. In order to ensure consistency of the signs of forces and kinematic parameters, the inertial vectors on the right hand side of the equations were drawn positive in the coordinate directions in a kinetic diagram, while forces were drawn in a free-body diagram before doing the equations of motion, see Figure 4.

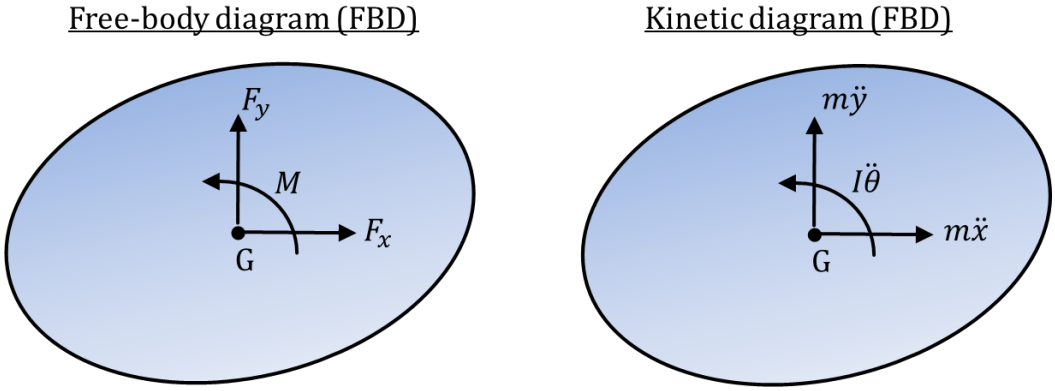


Figure 4 Diagrams for derivation of equations of motion for a rigid body in classical dynamics

The method we will apply for numerical analysis of unconstrained dynamic systems does essentially not differ from the classical framework. However, it is preferable to use linear algebra for formulation of the equations of motion. Rearranging equation 10 onto matrix form, we obtain

$$\begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix} \begin{Bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{Bmatrix} = \begin{Bmatrix} F_x \\ F_y \\ M \end{Bmatrix} \qquad 11.$$

or on short form

$$[\mathbf{M}](\ddot{\boldsymbol{q}}) = (\boldsymbol{f}) \qquad 12.$$

We refer to the matrix $[\boldsymbol{M}]$ as the mass matrix and the vector $(\boldsymbol{f})$ as the external force vector.

Example 3.1: Unconstrained motion of a slender rod subjected to gravity

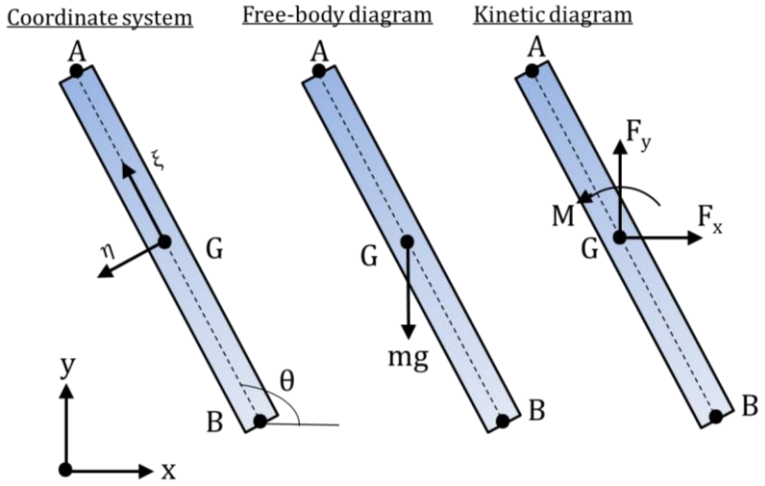


Figure 5 Unconstrained dynamics

We will now derive the equations of motion on matrix form similar to equation 11 for a long and slender rod. Firstly, it's convenient to be able to determine the position of the end points, A and B. The global position vector from G respectively to A and B can be obtained in terms of the local position vector and the rotation matrix. This provides the following expressions

$$(\boldsymbol{s})^A_{xy} = [\boldsymbol{A}](\boldsymbol{s})^A_{\xi\eta} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} \frac{L}{2} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{L}{2}\cos\theta \\ \frac{L}{2}\sin\theta \end{pmatrix} \qquad (\boldsymbol{s})^B_{xy} = [\boldsymbol{A}](\boldsymbol{s})^B_{\xi\eta} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} -\frac{L}{2} \\ 0 \end{pmatrix} = \begin{pmatrix} -\frac{L}{2}\cos\theta \\ -\frac{L}{2}\sin\theta \end{pmatrix}$$

In the global Cartesian frame, A and B will now have the following positions

$$(\boldsymbol{r})^A = (\boldsymbol{r}) + (\boldsymbol{s})^A_{xy} \qquad (\boldsymbol{r})^B = (\boldsymbol{r}) + (\boldsymbol{s})^B_{xy}$$

This is convenient if we want to plot the motion of the rod and not only consider the motion of G. The equations of motion are now on matrix form given by

$$[\boldsymbol{M}](\ddot{\boldsymbol{q}}) = (\boldsymbol{f}) \rightarrow \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & \frac{1}{12}ml^2 \end{bmatrix} \begin{Bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{Bmatrix} = \begin{pmatrix} 0 \\ -mg \\ 0 \end{pmatrix}$$

That's a mighty fine thing to have. Now we just have to figure out how to develop a numerical algorithm for solving those for the general case.

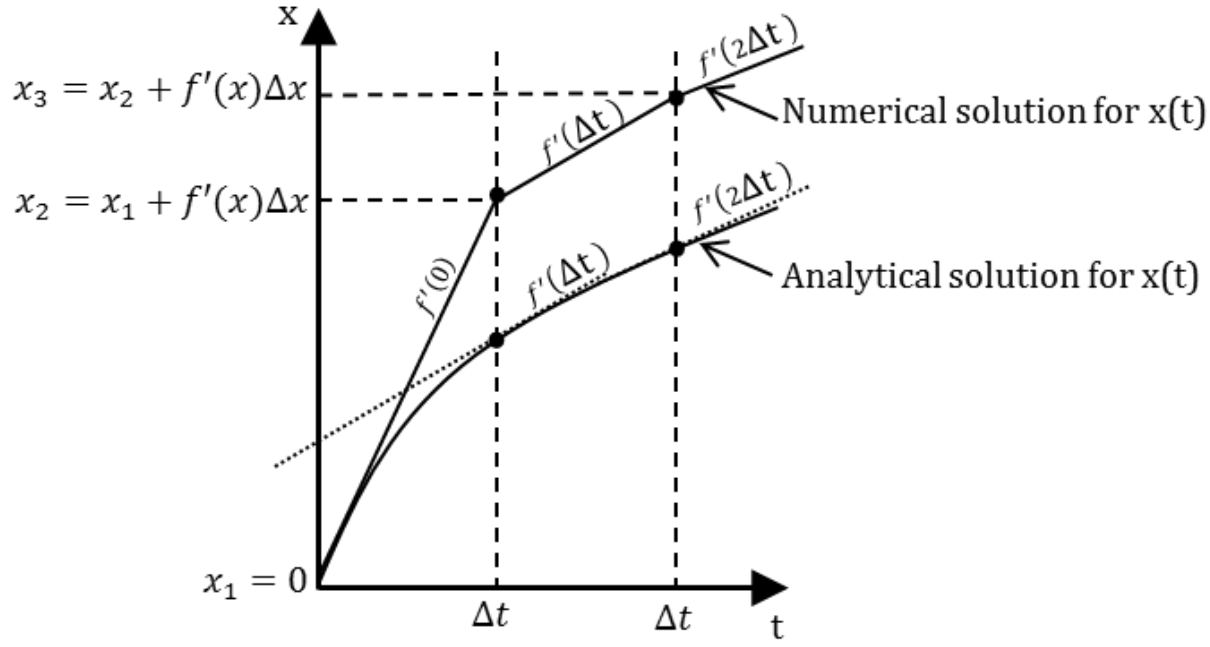


Figure 6 The principle of a 1st order time integration algorithm applied to a differential equation of 1st order

All there is left now is to develop an algorithm which for arbitrary external loads can solve the equations of motion numerically. In formard dynamics, equation 11 would be solved with respect to accelerations for known forces. Recalling the basic definition of velocity and positions, we have

$$\frac{d(\boldsymbol{v})}{dt} = (\boldsymbol{a}) \rightarrow (\boldsymbol{v}) = \int (\boldsymbol{a}) dt \qquad \frac{d\omega}{dt} = \alpha \rightarrow v = \int a dt \tag{13.}$$

$$\frac{d(\boldsymbol{r})}{dt} = (\boldsymbol{v}) \rightarrow (\boldsymbol{r}) = \int (\boldsymbol{v}) dt \qquad \frac{d\theta}{dt} = \omega \rightarrow \theta = \int \omega dt \tag{14.}$$

These are the analytical differential equations, we would require to solve. In classical mechanics, we would do so by separating the variables before integrating twice.

If we want to conduct the integrations in equation 13 and 14 numerically for given initial conditions (positions and velocities for time equal to zero), the simplest algorithm available is the Forward-Euler method – you should by now have heard about it the maths courses. We will on basis of a fixed time step size approximate the solution to a differential equation. This principle is visualized in Figure 6. With time step of size Δt, we for a differential equation of 2nd order (like the equations of motion) obtain

$$(\dot{\boldsymbol{q}})_{i+1} = (\dot{\boldsymbol{q}})_i + (\ddot{\boldsymbol{q}})_i \Delta t \qquad (\boldsymbol{q})_{i+1} = (\boldsymbol{q})_i + (\dot{\boldsymbol{q}})_i \Delta t \tag{15.}$$

The corresponds to linear interpolation of the solution in a two step process (once to integrate accelerations and obtain velocities and once more additionally to integrate velocities and obtain positions). The Forward-Euler method is simple to program, but has very low accuracy. This means, that a very small time step is required to obtain an accurate solution to a differential equation. If we instead of linear interpolation, had applied a polynomial of higher order using more than two mesh points in the time grid, we would say that the integrator was of higher order. It is common to rewrite differential equations of higher order to a first order system

allowing for the integration to be performed in a single step. For the 2nd order system in equation 15, the corresponding first order system is

$$(\boldsymbol{z}) = \begin{pmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{q} \end{pmatrix} \qquad (\boldsymbol{z})_{i+1} = (\boldsymbol{z})_i + (\dot{\boldsymbol{z}})_i \Delta t = \begin{pmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{q} \end{pmatrix}_i + \begin{pmatrix} \ddot{\boldsymbol{q}} \\ \dot{\boldsymbol{q}} \end{pmatrix}_i \Delta t \tag{16.}$$

This is the common form of differential equations applied for numerical integration when standard solvers in scientific programming languages are applied. The most commonly applied Matlab/GNU Octave solver is ODE45 and requires a differential equation in a separate function file on a form similar to equation 17. When applying ODE45 and similar solvers, it is of great importance to note, that these contrary to the formulation in equation 15, run on basis of a variable time step. We shall in chapter 0 discover, that it is highly inconvenient when simulating contact mechanics and it therefore for many dynamic problems in mechanical engineering is convenient to use fix-step time integrators. These are not included in the standard Matlab distribution[6]. While it is rather simple to write an Forward-Euler algorithm yourself, higher order fixed step solvers require a bit more effort to program yourself.

When writing your own first order solvers by using the two step process in equation 15, there is trick that improves the accuracy significantly, namely to use the updated velocity when integrating in the last step.

$$(\dot{\boldsymbol{q}})_{i+1} = (\dot{\boldsymbol{q}})_i + (\ddot{\boldsymbol{q}})_i \Delta t \qquad (\boldsymbol{q})_{i+1} = (\boldsymbol{q})_i + (\dot{\boldsymbol{q}})_{i+1} \Delta t \tag{17.}$$

This in many cases improve the accuracy of the solver from being rubbish to being just poor.

Just to make sure we are all on the same page here, numerical integration using equation 17. becomes

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix}_{i+1} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix}_i + \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{pmatrix}_i \Delta t \tag{18.}$$

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix}_{i+1} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}_i + \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix}_{i+1} \Delta t \tag{19.}$$

This is sufficient to get going and write our own simple solver. For higher order integrators, it is advisable to use standard solvers in scientific programming suites[7].

Code Example 3.1: unconstrained dynamics of a rod in the field of gravity

```
%Set initial conditions
x0=0; y0=0; tht0=pi/4;
vx0=2; vy0=10; omg0=5;
IniCnd=[x0,y0,tht0,vx0,vy0,omg0];
%Generate line body
m=2; l=1.2; I=1/12*m*l^2;
GenLnsBdy(IniCnd,m,I,l)
PltLim=[-1,6,-1,7];      %Set plot window size
nt=500;                  %Number of integration points
tlim=2;                  %Time limit
PlotCoGTrcs="on"         %Plot the motion track of CoG
PltBdyRsp(1,1)           %Plot 1st body, 1 coordinate (x)
PltBdyRsp(1,2)           %Plot 1st body, 2 coordinate (y)
```

[6] Fixed-step differential equation solvers of higher order for Matlab can be downloaded here

[7] Numerical integration of ODE's, methods of various order on MIT-Open Courseware

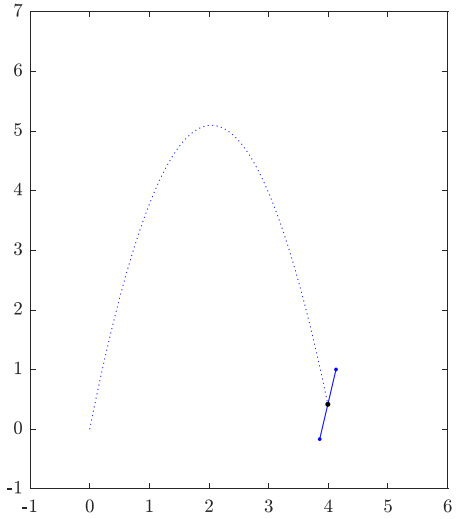


Figure 7 Visualization of simulated results

4. Springs and dampers

Carla the Kraken has solved crimes with Sherlock and travelled with the Doctor. As Stephen Moffat offered her an own show, she instead borrowed the Tardis and went back in time to safe Amy

Force elements like translation and rotation springs are **not** to be considered kinematic constraints. These will introduce an external force for which magnitude and direction are governed by the spatial position and orientation of the connection points. Considering the massless translation spring mounted between the two rigid bodies in Figure 8, the relative position between the two mounting points, P_i and P_j, is given by

$$(\boldsymbol{r})_{ij} = (\boldsymbol{r})_j - (\boldsymbol{r})_i \qquad 20.$$

This vector governs the direction of the spring forces. Forward dynamic analysis can be performed on basis of integration of the unconstrained Newton-Euler equations 11-12. The external spring forces are given by

$$(\boldsymbol{F})^{(i)}_{spring} = \frac{\boldsymbol{r}_{ij}}{\|\boldsymbol{r}_{ij}\|} k\left(\|\boldsymbol{r}_{ij}\| - l\right) \qquad (F)^{(j)}_{spring} = -(F)^{(i)}_{spring} \qquad 21.$$

in which the first term is a unit vector representing the spring force reaction, the second term k denotes the spring stiffness and the last term represents the spring elongation, in which l denotes the undeformed spring length. If a massless damper is mounted in parallel to the spring, the damping force is given by

$$(\boldsymbol{F})^{(i)}_{damping} = -\frac{\boldsymbol{v}_{ij}}{\|\boldsymbol{v}_{ij}\|} c\boldsymbol{v}_{ij} \qquad (\boldsymbol{F})^{(j)}_{damping} = -(\boldsymbol{F})^{(i)}_{damping} \qquad 22.$$

in which the relative velocity $\boldsymbol{v}_{ij}$ can be obtained by differentiation of equation 20 and the c denotes the damping constant. The positions and velocities of the points where the springs are attached can be determined by equation 5 and 7.

These equations are directly transferable to rotation springs and dampers as shown in Figure 9. However, the external actions generated are moments, the position measures are angles, the velocity measures are angular velocities and the spring constant is measured in force per angle.

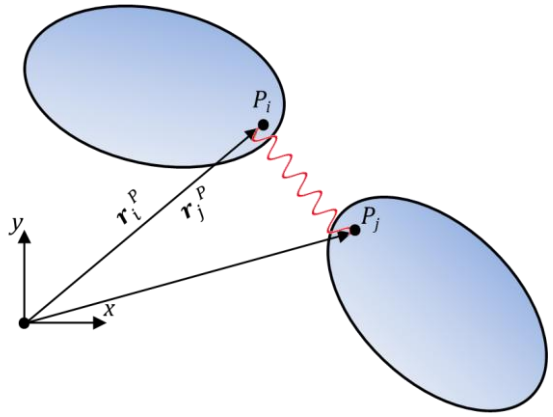


Figure 8: Translation spring

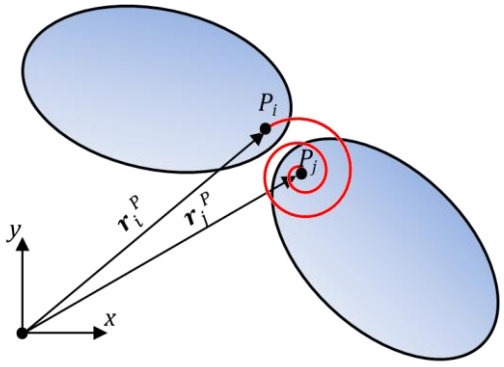


Figure 9: Rotation spring

Code Example 4.1: Three rigid bodies connected by springs

```
nt=1000;              %Number of time steps
tlim=2;               %Time limit
SolverType="Fix";     %Fix time step solver
Bdy2BdyCts="on";      %Body-to-Body contacts on/off
Bdy2WldCts="on";      %Body-to-World contact on/off
PlotBdbs= "off";      %Plot bounding boxes off
%Body 1, attached to world
PltLim=[-2,8,-2,8]*1.2;
Pts_Geb1=[0,0;0,1;1,1;1,0];
TriLns_Geb1=[1,2,3; 1,3,4];
```

```
IniCnd=[4,5,0,0,0,0];
GenGebBdy(IniCnd,2,0.2,Pts_Geb1,TriLns_Geb1)
Bdy2WldTrsSpr(100,5,1,1,[-0.25,0.25],[IniCnd(1)-0.2,IniCnd(2)+1]);
Bdy2WldTrsSpr(100,5,1,1,[0.25,0.25],[IniCnd(1)+0.2,IniCnd(2)+1]);
PltBdyRsp(BdyNum,2)
%Body 2, attached to body 1
Pts_Geb1=[0,0;0,1;1,1;1,0]*1.05;
TriLns_Geb1=[1,2,3; 1,3,4];
IniCnd=[4,3,0,0,0,0];
GenGebBdy(IniCnd,1,0.2,Pts_Geb1,TriLns_Geb1)
Bdy2BdyTrsSpr(100,5,1,1,[-0.25,-0.25],2,[-0.25,0.25]);
Bdy2BdyTrsSpr(100,5,1,1,[0.25,-0.25],2,[0.25,0.25]);
PltBdyRsp(BdyNum,2)
%Body 3, attached to body 2
Pts_Geb1=[0,0;0,1;1,1;1,0]*1.1;
TriLns_Geb1=[1,2,3; 1,3,4];
IniCnd=[4,1,0,0,0,0];
GenGebBdy(IniCnd,1,0.2,Pts_Geb1,TriLns_Geb1)
Bdy2BdyTrsSpr(100,5,1,2,[-0.25,-0.25],3,[-0.25,0.25]);
Bdy2BdyTrsSpr(100,5,1,2,[0.25,-0.25],3,[0.25,0.25]);
PltBdyRsp(BdyNum,2)
```

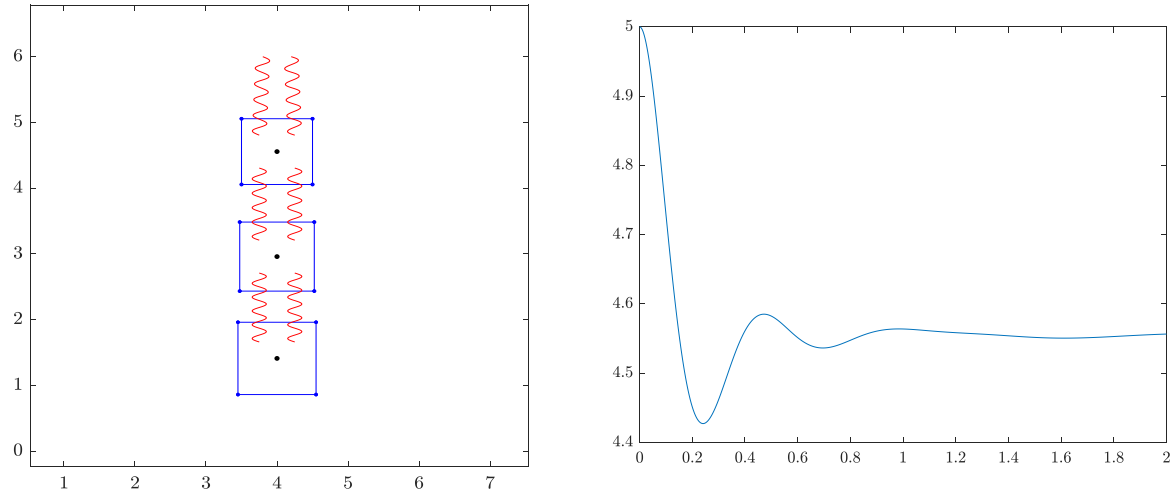
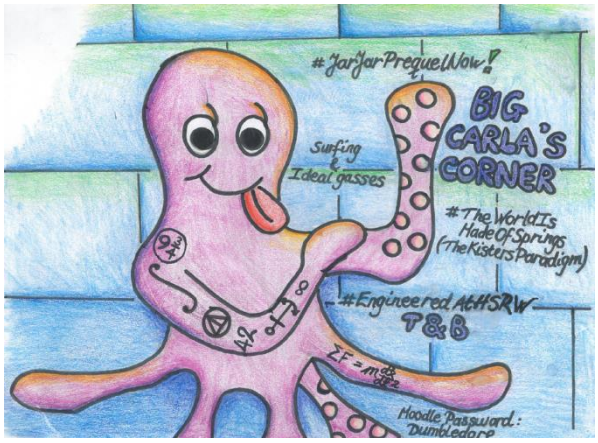


Figure 10 Simulated results
Left: Model visualization, **Right:** Position response in time of top (upper) body

5. Contact simulation



Carla the Kraken has awesome ink and is really good at street art and free-style

In this section, simple modeling techniques for effects caused by contacts will be described.

5.1. Friction

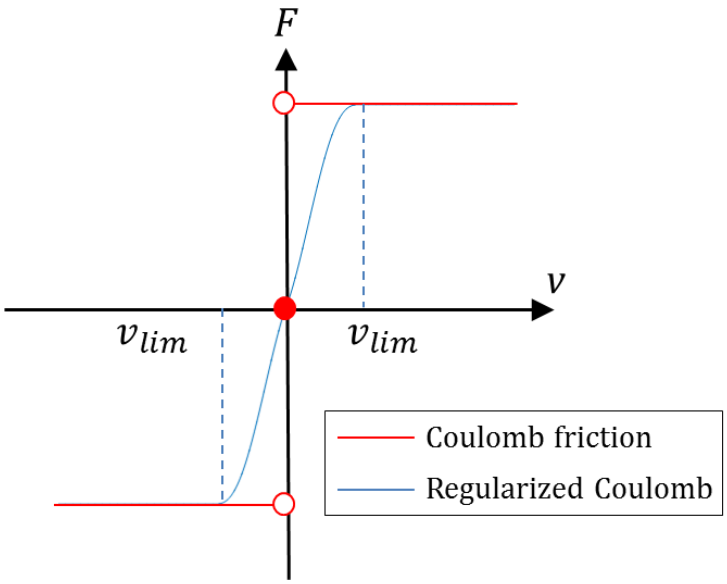


Figure 11 Coulomb friction, original and regularized model

Frictional effects are of nature complex and difficult to model detailed with high accuracy. The simplest approach to calculation of external forces due to friction, is Coulomb's friction model, in which the friction is simulated as speed independent. Thereby, the velocity only effects the friction forces by its direction, since friction counteracts the motion. Coulomb's friction model is visualized in Figure 11. Coulomb friction is defined by the equation

$$(\boldsymbol{F})_{friction} = \begin{cases} -\dfrac{\boldsymbol{v}}{\|\boldsymbol{v}\|}\mu N & \|\boldsymbol{v}\| > 0 \\ 0 & \|\boldsymbol{v}\| = 0 \end{cases} \qquad 23.$$

The discontinuity in this model is however not well-posed for numerical simulations, since the friction will switch signs for low velocity, which in any cases will cause vibrating motions when

simulating objects that should be at rest. Since these are non-physical, it is common to regularize equation 31 by simulating a transition between zero friction and full friction. This is commonly done by modifying Coulomb's original equation to be on the form

$$(\boldsymbol{F})_{friction} = -p(\|\boldsymbol{v}\|)\frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}\mu N \qquad 24.$$

In which the continuous function $p(v)$ constitutes a transition between the two levels and has as function of velocity has to fulfill the following requirements

$$p(0) = 0 \qquad p(v_{lim}) = 1 \qquad \frac{dp(v_{lim})}{dv} = 0 \qquad 25.$$

In equation 26, v_{lim} denotes the slip velocity, for which full friction is obtained, see Figure 11. Various functions can be chosen to simulate the transition, and a common choice is *arctan*. However, with three conditions available, a polynomial of 2nd order is also a logical choice

$$p(v) = av^2 + bv + c \qquad 26.$$

Substitution of the conditions in equation 25 into equation 26 enables us to solve for the constants a, b and c

$$a = -\frac{1}{v_{lim}^2} \qquad b = \frac{2}{v_{lim}} \qquad c = 0 \qquad 27.$$

The current formulation is well-posed for numerical implementation. However, Coulomb friction is for many problems not of sufficient accuracy to obtained good correlations between measurements and simulated results[8]

Code Example 5.1: Cylinder rolling and slipping on incline

```
PlotCoGs= "on";      PlotCoGTrcs="on";    PlotBdbs= "off";     Bdb="on";
Bdy2BdyCts="on";     Bdy2WldCts="on";     SolverType="Fix";
%Contact Parameters
dlt=0.05;            %Total thickness of contact layer
k_b2b=10^4*2;        %Body-to-Body contact stiffness
c_b2b=200;           %Body-to-Body contact damping
my_b2b=0.5;          %Friction coefficient, Body-to-Body
k_b2w=10^4*2;        %Body-to-World contact stiffness
c_b2w=200;           %Body-to-World contact damping
my_b2w=0.5;          %Friction coefficient, Body-to-World
CtsStp=1;            %How often contact checks are performed
PltFct=1;            %Plot each time step
nt=800;              %Number of time steps
tlim=3;              %Time limit
%Geometry
IniCnd=[9,7,0,0,0,0];
GenCrcBdy(IniCnd,1,0.1,0.5);
GenLnsWld([0,0],[10,6])
```

[8] An overview of various more sophisticated friction models with references to the original publications is available in [4].

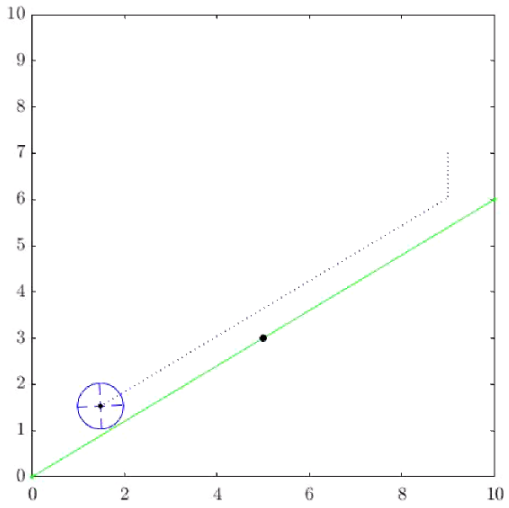


Figure 12 Visualization of simulated results

5.2. Detection and forces by penalization

Contact simulations are in classical dynamics performed using the principle of impulse and momentum enabling impact calculations. For computational dynamics, this principle does have several disadvantages. First of all, this calculation method is not appropriate for simulation of more permanent contacts. Secondly, the solution to contact problems including more than two bodies, do not have a unique solution. It is therefore desirable to apply a different framework without the short comings listed above.

The most common method for contact simulations is penalization, by which contact forces are added based on a mass-spring-damper system acting only when contact is detected. While penalization methods requires numerical solves with fixed time step, and the time steps furthermore must be small to simulate the impact with sufficient accuracy, these are well-posed for numerical simulations.

Each rigid body will be considered surrounded by a surface known as the contact layer with thickness ε. Contact is detected when the contact layer of one body penetrate the contact layer of another.

The simplest example of contact simulation by penalization is an impact between two cylindrical objects, see Figure 13. Contact checks may be performed by calculating the norm of the position vector between the two center points. If this distance is smaller than the sum of both radii plus the thickness of the contact layers, contact is detected.

For bodies of general shape, contact detection is slightly more complex. These can however be considered constituted by lines and points. It is sufficient to check all points in each body against all lines in another and vice versa, see Figure 14.

In order for the point P_i to be in contact with the line spanned by the points Q_j and R_j, the projection of P_i onto the line segment must lie between Q_j and R_j and the distance between $P_i$and the line must be smaller than the sum of the total thickness of the contact layers.

The length of the projection of P_i onto the line segment can along with the perpendicular distance between point and line be calculated by

$$(\boldsymbol{a})_1 = |\boldsymbol{a}|cos\theta = (\boldsymbol{a}) \cdot \frac{(\boldsymbol{b})}{|\boldsymbol{b}|} \qquad (\boldsymbol{a})_2 = (\boldsymbol{a}) - (\boldsymbol{a})_1 \tag{28.}$$

The tangent to the line and the corresponding unit tangent are given by

$$(\boldsymbol{t}) = (\boldsymbol{R})_j - (\boldsymbol{Q})_j \qquad (\boldsymbol{e})_t = \frac{(\boldsymbol{R})_j - (\boldsymbol{Q})_j}{|(\boldsymbol{R})_j - (\boldsymbol{Q})_j|} \tag{29.}$$

The length of the projection can now be written

$$|p| = \left((\boldsymbol{P})_i - (\boldsymbol{Q})_j\right) \cdot \left(\frac{(\boldsymbol{R})_j - (\boldsymbol{Q})_j}{|(\boldsymbol{R})_j - (\boldsymbol{Q})_j|}\right) \tag{30.}$$

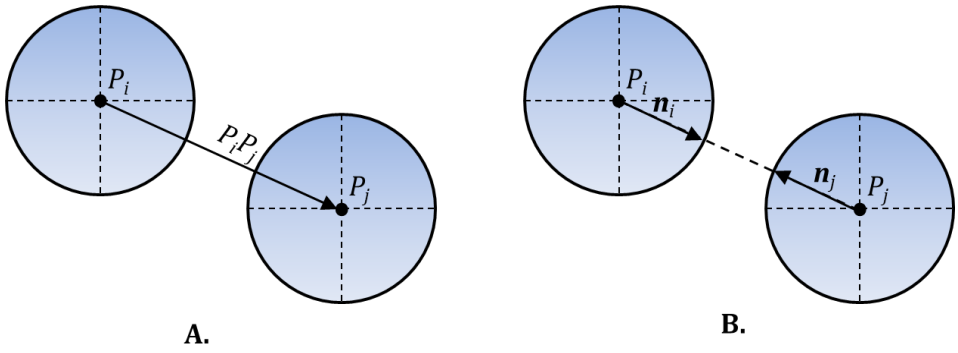


Figure 13 Contact detection between two cylindrical objects

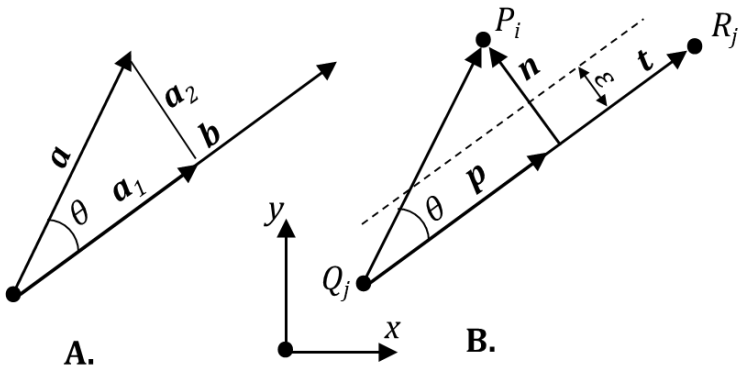


Figure 14 Point-to-Line contact detection

The projection can as a vector be written as

$$(\boldsymbol{p}) = |p|(\boldsymbol{e})_t \tag{31.}$$

The normal vector from the line segment to the point is now given by

$$(\boldsymbol{n}) = \left((\boldsymbol{P})_i - (\boldsymbol{Q})_j\right) - (\boldsymbol{p}) \tag{32.}$$

Having those vectors defined, the position of the point can be written as a linear combination on the following form

$$(\boldsymbol{P})_i = (\boldsymbol{Q})_j + k_1(\boldsymbol{t}) + k_2(\boldsymbol{n}) \rightarrow (\boldsymbol{P})_i - (\boldsymbol{Q})_j = k_1(\boldsymbol{t}) + k_2(\boldsymbol{n}) \tag{33.}$$

Using matrix algebra, this is rewritten onto the following linear system of equations

$$\begin{pmatrix} x \\ y \end{pmatrix}_{(\boldsymbol{P}_i)} - \begin{pmatrix} x \\ y \end{pmatrix}_{(\boldsymbol{Q}_j)} = \left[\begin{pmatrix} x \\ y \end{pmatrix}_{(\boldsymbol{t})} \quad \begin{pmatrix} x \\ y \end{pmatrix}_{(\boldsymbol{n})}\right] \begin{pmatrix} k_1 \\ k_2 \end{pmatrix} \tag{34.}$$

Solving for k_1 and k_2, contact is detected if the following condition is met:

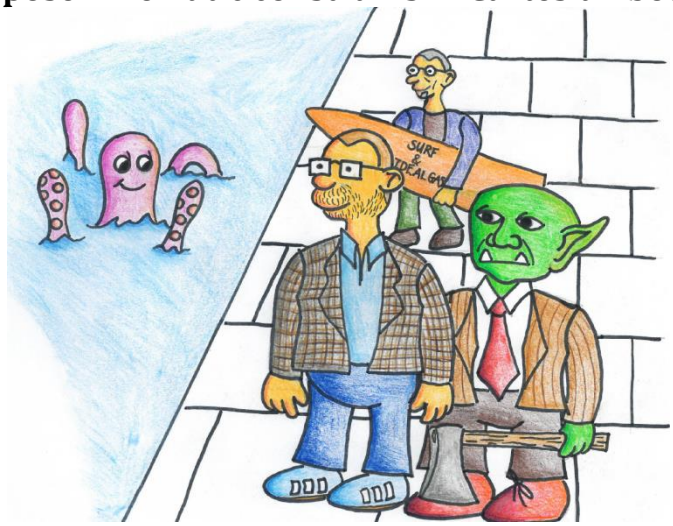
$$0 \leq k_1 \leq 1, k_2 < \varepsilon \tag{35.}$$

The developed framework is efficient for numerical implementation.

If a large number of bodies of general shape are simulated, the number of contact checks for each time step will obviously become large. Actually, contact detection often requires more computational time than time integration. In order to reduce the number of contact checks, it is common to apply a bounding box for example of circular shape around each body. Detailed contact checks are not performed before the bounding boxes intersect. This forms a hierarchical contact detection structure and significantly increases the computational speed.

6. Constrained dynamics

6.1. Specific purpose kinematic constrains in Cartesian body coordinates



Carla the Kraken understands most languages common on campus. This includes sophisticated tongues like German and High Valyrian, but also more primitive dialects like Danish and Flabibabdab – the common language among stock trading orcs.

In general, we write kinematic constraints for the positon vector *r* as

$$(\boldsymbol{\Phi}) = 0 \tag{36.}$$

Our aim is to link the accelerations in the Newton-Euler equations to the kinematic constraints. I.e. we better start differentiating. We write

$$\left(\dot{\boldsymbol{\Phi}}\right) = [\boldsymbol{D}](\dot{\boldsymbol{r}}) = \boldsymbol{0} \tag{37.}$$

in which $\boldsymbol{D}$ is the *Jacobian matrix* and $\dot{\boldsymbol{r}}$ are the time derivatives of the position functions. We continue our differentiating quest and write

$$\ddot{\boldsymbol{\Phi}} = [\boldsymbol{D}](\ddot{\boldsymbol{r}}) + \left[\dot{\boldsymbol{D}}\right](\dot{\boldsymbol{r}}) = \boldsymbol{0} \tag{38.}$$

Reranging this, we may introduce a general expression for the right hand side of the acceleration constraint:

$$[\boldsymbol{D}](\ddot{\boldsymbol{r}}) = -\left[\dot{\boldsymbol{D}}\right](\dot{\boldsymbol{r}}) = (\boldsymbol{\gamma}) \tag{39.}$$

We now modify the Newton-Euler equations to the following form

$$[\boldsymbol{M}](\ddot{\boldsymbol{q}}) = (\boldsymbol{f}) + {}^{(c)}(\boldsymbol{f}) \rightarrow [\boldsymbol{M}](\ddot{\boldsymbol{q}}) - {}^{(c)}(\boldsymbol{f}) = (\boldsymbol{f}) \tag{40.}$$

In which ${}^{(c)}(\boldsymbol{f})$ are the reaction forces introduced by the kinematic constraints and $\boldsymbol{h}$ remains the vector containing the of external forces.

We may write ${}^{(c)}(\boldsymbol{f}) = [\boldsymbol{D}]^{\boldsymbol{T}}(\boldsymbol{\lambda})$

We want to solve the equations $[\boldsymbol{M}](\ddot{\boldsymbol{q}}) - [\boldsymbol{D}]^{\boldsymbol{T}}(\boldsymbol{\lambda}) = (\boldsymbol{f})$ and $(\boldsymbol{\gamma}) = [\boldsymbol{D}](\dot{\boldsymbol{r}})$ simultaneously, so we write

$$\begin{bmatrix}[\boldsymbol{M}] & [\boldsymbol{D}]^{\boldsymbol{T}} \\ [\boldsymbol{D}] & \boldsymbol{0}\end{bmatrix}\begin{pmatrix}(\ddot{\boldsymbol{q}}) \\ (\boldsymbol{\lambda})\end{pmatrix} = \begin{pmatrix}(\boldsymbol{f}) \\ (\boldsymbol{\gamma})\end{pmatrix} \tag{41.}$$

This system of linear equations most be solved for known forces $(\boldsymbol{f})$ and $(\boldsymbol{\gamma}) = -[\dot{\boldsymbol{D}}](\dot{\boldsymbol{r}})$ for each timestep in the numerical integration. Once this has been done, the none linear differential equations in time can be integrated, for example by an implicit-Euler integrator

$$\begin{bmatrix}[\boldsymbol{M}] & [\boldsymbol{D}]^{\boldsymbol{T}} \\ [\boldsymbol{D}] & \boldsymbol{0}\end{bmatrix}\begin{pmatrix}(\ddot{\boldsymbol{q}}) \\ (\boldsymbol{\lambda})\end{pmatrix} = \begin{pmatrix}(\boldsymbol{f}) \\ (\boldsymbol{\gamma}) - 2\alpha(\dot{\boldsymbol{\Phi}}) - \beta^2(\boldsymbol{\Phi})\end{pmatrix} \tag{42.}$$

Example 6.1: A physical pendulum with a long and slender rod

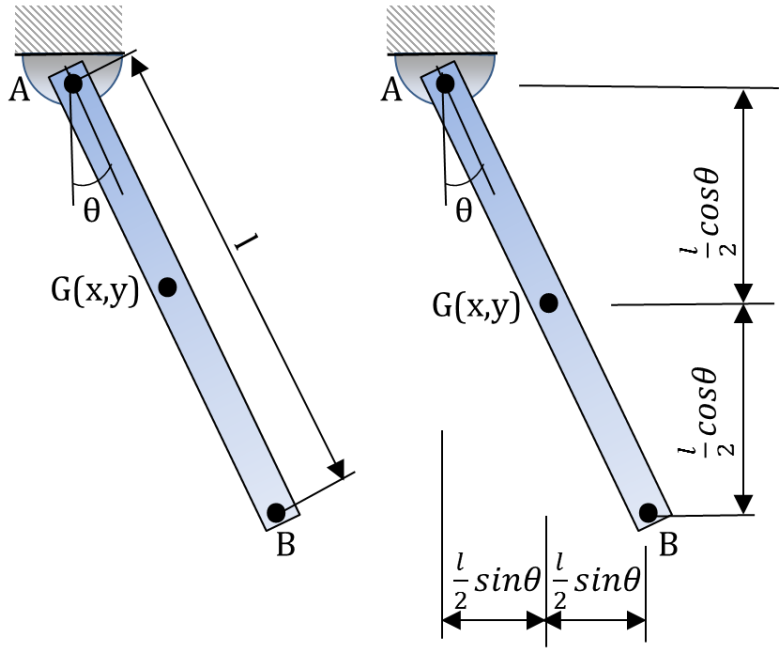


Figure 15: A rod shaped physical pendulum

In this example, the motion of a physical pendulum constituted by a long and slender rod will be considered, see Figure 15. Since the considered problem contains a pin-joint in point A, the system is constrained. The constraint equation for the pin joint is given by

$$(\boldsymbol{\Phi}) = \begin{pmatrix}x_A \\ y_A\end{pmatrix} = \begin{pmatrix}x_G - \sin\theta\frac{l}{2} \\ y_G + \cos\theta\frac{l}{2}\end{pmatrix} = \boldsymbol{0}$$

It is noted that the coordinate system does not correspond to the commonly chosen definition in computational dynamics. The Jacobian derived later, in section 6.2, will therefore deviate, but both results may be implemented for numerical analysis when accounting for how θ is defined.

In which the coordinates of A, which are not available directly in the mathematical formulation are expressed in terms of the coordinates of the center of gravity and the spatial orientation angle of the rigid body. We differentiate this in time and rearrange:

$$(\dot{\boldsymbol{\Phi}}) = \begin{pmatrix} \dot{x}_G - \cos\theta\dot{\theta}\frac{l}{2} \\ \dot{y}_G - \sin\theta\dot{\theta}\frac{l}{2} \end{pmatrix} = \begin{bmatrix} 1 & 0 & -\cos\theta\frac{l}{2} \\ 0 & 1 & -\sin\theta\frac{l}{2} \end{bmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = [\boldsymbol{D}] \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix}$$

Differentiating this expression once more, we obtain the following expression

$$(\ddot{\boldsymbol{\Phi}}) = \frac{d}{dt}\begin{pmatrix} \dot{x}_G - \cos\theta\dot{\theta}\frac{l}{2} \\ \dot{y}_G - \sin\theta\dot{\theta}\frac{l}{2} \end{pmatrix} = \begin{pmatrix} \ddot{x}_G - \cos\theta\ddot{\theta}\frac{l}{2} + \sin\theta(\dot{\theta})^2\frac{l}{2} \\ \ddot{y}_G - \sin\theta\ddot{\theta}\frac{l}{2} - \cos\theta(\dot{\theta})^2\frac{l}{2} \end{pmatrix}$$

$$= \begin{bmatrix} 1 & 0 & -\cos\theta\frac{l}{2} \\ 0 & 1 & -\sin\theta\frac{l}{2} \end{bmatrix} \begin{Bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{Bmatrix} + \begin{bmatrix} 0 & 0 & \sin\theta\dot{\theta}\frac{l}{2} \\ 0 & 0 & -\cos\theta\dot{\theta}\frac{l}{2} \end{bmatrix} \begin{Bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{Bmatrix} = [\boldsymbol{D}] \begin{Bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{Bmatrix} + [\dot{\boldsymbol{D}}] \begin{Bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{Bmatrix}$$

We here recognize the Jacobian $[\boldsymbol{D}]$ and it's first order time derivative $[\dot{\boldsymbol{D}}]$, which by the way also could have been obtained directly by differentiation of $[\boldsymbol{D}]$. The system of equations to solve for each time step is now in accordance with equation 42 given by

$$\begin{bmatrix} [\boldsymbol{M}] & [\boldsymbol{D}]^{\boldsymbol{T}} \\ [\boldsymbol{D}] & \boldsymbol{0} \end{bmatrix} \begin{pmatrix} (\ddot{\boldsymbol{q}}) \\ (\boldsymbol{\lambda}) \end{pmatrix} = \left[\begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & \frac{1}{12}ml^2 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -\cos\theta\frac{l}{2} & -\sin\theta\frac{l}{2} \end{bmatrix} \right] \begin{Bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \\ \lambda_1 \\ \lambda_2 \end{Bmatrix} = \begin{Bmatrix} 0 \\ -mg \\ 0 \\ -\dot{\boldsymbol{D}}\dot{\boldsymbol{r}} \end{Bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -\cos\theta\frac{l}{2} \\ 0 & 1 & -\sin\theta\frac{l}{2} \end{bmatrix} \quad \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Code Example 6.1: The physical pendulum

```
PlotCoGs= "on";      PlotCoGTrcs="on";    PlotBdbs= "off";     Bdb="off";
Bdy2BdyCts="off";    Bdy2WldCts="off";    SolverType="Var";
PltLim=[0.7,1.2,0.6,1.5];
%Contact Parameters
PltFct=1;            %Plot each time step
nt=800;              %Number of time steps
tlim=3;              %Time limit
%Generate geometry
Pts_Geb1=[0,0;0,0.05;0.7,0.05;0.7,0];
TriLns_Geb1=[1,2,3; 1,3,4];
IniCnd=[1,1,0-pi/2*0.9,0,0,0];
GenGebBdy(IniCnd,2,1/12*2*0.7^2,Pts_Geb1,TriLns_Geb1)
%Generate Joint
Bdy2WldPinJnt(BdyNum,[-0.35,0])
%Plot rotation
PltBdyRsp(BdyNum,3)
```

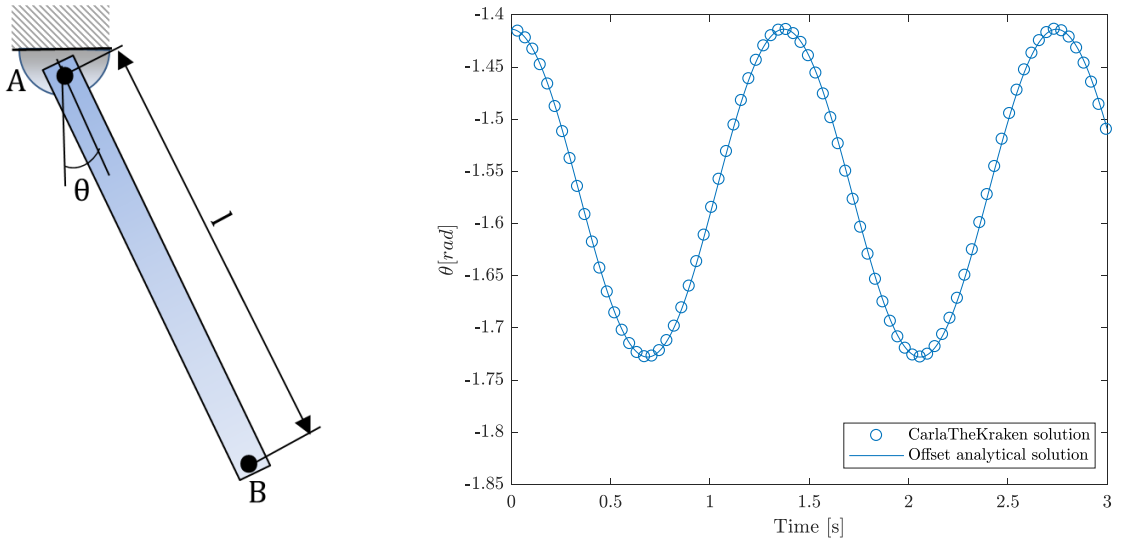
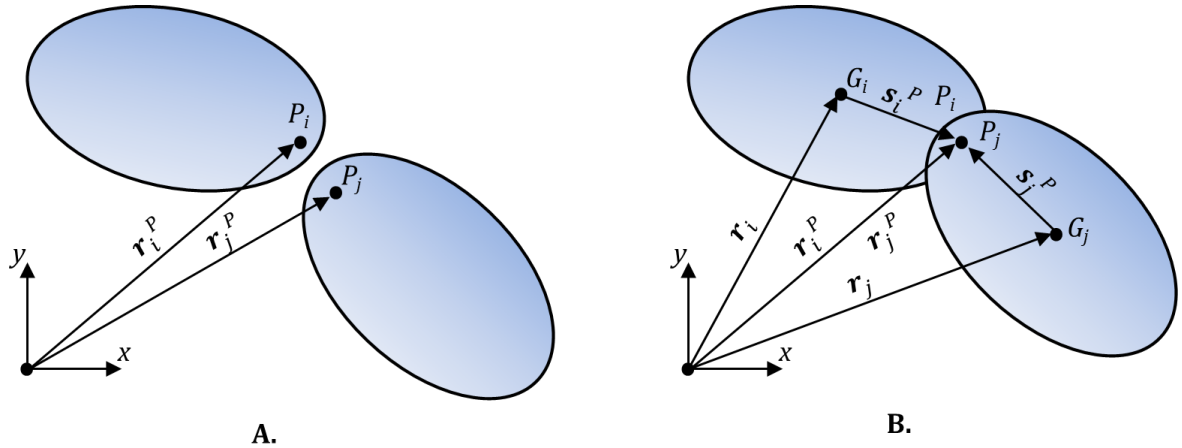


Figure 16: A rod shaped physical pendulum

6.2. General purpose kinematic constrains in local body frames

In section 6.1, it was shown how kinematic constrains are formulated for special cases. However, it is possible to define constrain equations parametrically. In the current section, it will be shown how to develop general constraint equations for a pin-joint for direct implementation in equation 42. The derivation is equivalent to the one given by Nikravesh [1]. For generic constrain-equations, in particular for a translation- or sliding joint as seen in the piston in a crank-slider mechanism, you may look these up in [1] or [2], and the general purpose expressions are not included in these notes.



A. **B.**

Figure 17 Vector definitions for a general purpose kinematic joint constraint

Considering the two bodies in Figure 17, a pin- or revolute joint can be formed using the kinematic constraint equations

$$(\boldsymbol{\Phi}) = (\boldsymbol{r})_j^P - (\boldsymbol{r})_i^P \qquad 43.$$

By differentiation in time, the following expression is obtained

$$\begin{aligned}\dot{\boldsymbol{\Phi}} &= \dot{\boldsymbol{r}}_j^P - \dot{\boldsymbol{r}}_i^P \\ &= \dot{\boldsymbol{r}}_j + \breve{\boldsymbol{s}}_j^P \dot{\theta}_j - \dot{\boldsymbol{r}}_i - \breve{\boldsymbol{s}}_i^P \dot{\theta}_i \\ &= \begin{bmatrix} -\boldsymbol{I} & -\breve{\boldsymbol{s}}_i^P & \boldsymbol{I} & \breve{\boldsymbol{s}}_j^P \end{bmatrix} \begin{pmatrix} \dot{\boldsymbol{r}}_i \\ \dot{\theta}_i \\ \dot{\boldsymbol{r}}_j \\ \dot{\theta}_j \end{pmatrix} = \boldsymbol{0}\end{aligned} \qquad 44.$$

Differentiating once more, we obtain

$$\begin{aligned}\ddot{\boldsymbol{\Phi}} &= \begin{bmatrix} -\boldsymbol{I} & -\breve{\boldsymbol{s}}_i^P & -\boldsymbol{I} & -\breve{\boldsymbol{s}}_j^P \end{bmatrix} \begin{pmatrix} \ddot{\boldsymbol{r}}_i \\ \ddot{\theta}_i \\ \ddot{\boldsymbol{r}}_j \\ \ddot{\theta}_j \end{pmatrix} + \begin{bmatrix} \boldsymbol{0} & -\dot{\breve{\boldsymbol{s}}}_i^P & \boldsymbol{0} & \dot{\breve{\boldsymbol{s}}}_j^P \end{bmatrix} \begin{pmatrix} \dot{\boldsymbol{r}}_i \\ \dot{\theta}_i \\ \dot{\boldsymbol{r}}_j \\ \dot{\theta}_j \end{pmatrix} = \boldsymbol{0} \\ \rightarrow & \begin{bmatrix} -\boldsymbol{I} & -\breve{\boldsymbol{s}}_i^P & -\boldsymbol{I} & -\breve{\boldsymbol{s}}_j^P \end{bmatrix} \begin{pmatrix} \ddot{\boldsymbol{r}}_i \\ \ddot{\theta}_i \\ \ddot{\boldsymbol{r}}_j \\ \ddot{\theta}_j \end{pmatrix} = \dot{\breve{\boldsymbol{s}}}_i^P \dot{\theta}_i - \dot{\breve{\boldsymbol{s}}}_j^P \dot{\theta}_j\end{aligned} \qquad 45.$$

The Jabobian and the gamma vector are now recognized to be on the form

$$\begin{aligned}\boldsymbol{D} &= \begin{bmatrix} -\boldsymbol{I} & -\breve{\boldsymbol{s}}_i^P & \boldsymbol{I} & \breve{\boldsymbol{s}}_j^P \end{bmatrix} \\ \boldsymbol{\gamma} &= \dot{\breve{\boldsymbol{s}}}_i^P \dot{\theta}_i - \dot{\breve{\boldsymbol{s}}}_j^P \dot{\theta}_j\end{aligned} \qquad 46.$$

These expressions are generally valid and can be implemented directly

Calculated example 6.2: joint constrains on a slender rod

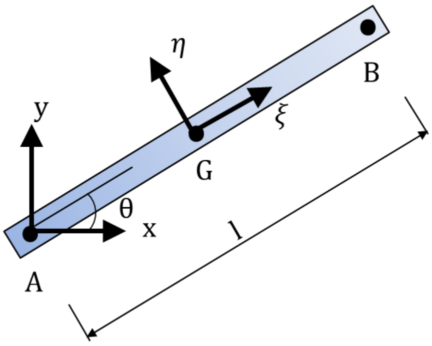


Figure 18 A slender rod constrained by a pin-joint

In this example, the physical pendulum from the previous example will be considered using the coordinate definition commonly applied for computational rigid body dynamics. It will now be demonstrated that the Jacobian derived using specific- and general purpose methods are equal.

Using the specific purpose approach for formulation of kinematic constrains, we obtain

$$\boldsymbol{\Phi} = \begin{pmatrix} x_G - \cos\theta \frac{l}{2} \\ y_G - \sin\theta \frac{l}{2} \end{pmatrix} = \mathbf{0}$$

We differentiate this in time and rearrange:

$$\dot{\boldsymbol{\Phi}} = \begin{pmatrix} \dot{x}_G + \sin\theta\dot{\theta}\frac{l}{2} \\ \dot{y}_G - \cos\theta\dot{\theta}\frac{l}{2} \end{pmatrix} = \begin{bmatrix} 1 & 0 & \sin\theta\frac{l}{2} \\ 0 & 1 & -\cos\theta\frac{l}{2} \end{bmatrix} \begin{Bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{Bmatrix}$$

In which we recognize the Jacobian as the matrix in the last equation.
Now utilizing the approach for general purpose codes from the present section, the vector from the CoG to the constrained point in A is in the local body frame given by

$$\boldsymbol{s}^A_{\xi\eta} = \begin{pmatrix} -\frac{l}{2} \\ 0 \end{pmatrix}$$

Transformation to the global coordinate frame gives

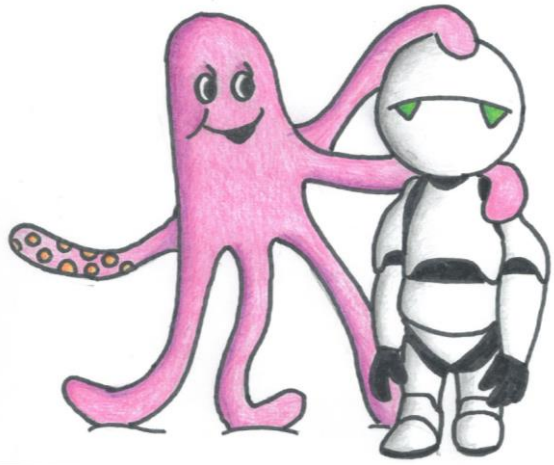
$$\boldsymbol{s}^A_{xy} = \boldsymbol{A}\boldsymbol{s}^A_{\xi\eta} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} -\frac{l}{2} \\ 0 \end{pmatrix} = \begin{pmatrix} -\frac{l}{2}\cos\theta \\ -\frac{l}{2}\sin\theta \end{pmatrix} \rightarrow \breve{\boldsymbol{s}}^A_{xy} = \begin{pmatrix} \frac{l}{2}\sin\theta \\ -\frac{l}{2}\cos\theta \end{pmatrix}$$

Using equation 45, we obtain the Jacobian matrix

$$\begin{bmatrix} \boldsymbol{I} & \breve{\boldsymbol{s}}^P_j \end{bmatrix} = \begin{bmatrix} 1 & 0 & \sin\theta\frac{l}{2} \\ 0 & 1 & -\cos\theta\frac{l}{2} \end{bmatrix}$$

It is observed that the two Jacobians are equal

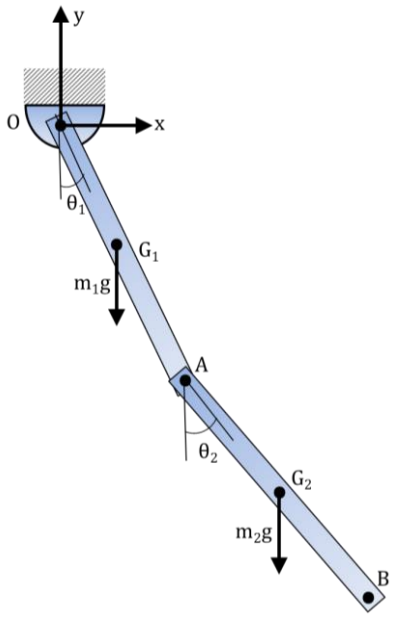
7. Mechanism design: open and closed kinematic chains



Carla the Kraken has by Marvin the paranoid android been characterized as My last and only friend, and the only being who understands me and doesn't make me depressed at all

A chain of rigid bodies, which only is attached to the world frame in one end is referred to as an open chain mechanism, while a chain of rigid bodies reattached to the world frame in multiple points is denoted a closed chain mechanism. Crank-slider and crank-rocker mechanisms are common examples of closed chains. In the following calculated example, the double pendulum, one of the simplest physical systems exhibiting chaotic behavior as there is no repeated pattern in the solution to the equations of motion, will be considered.

Calculated example 7.1: The double pendulum



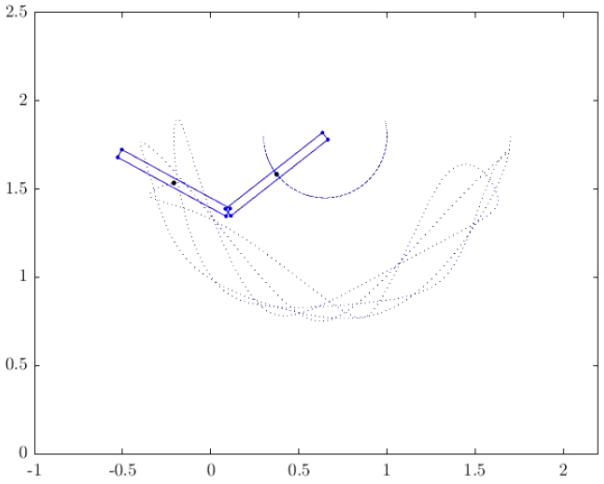


Figure 19 A double pendulum constituted by two long and slender rods

The four constraint equations required to form two pin-joints are given by

$x_O^{(1)} = 0 \qquad y_O^{(1)} = 0 \qquad x_A^{(1)} = x_B^{(2)} \qquad y_A^{(1)} = y_B^{(2)}$

Expressing those in terms of angle and coordinates of the center of gravity for both rods, the following two expressions four obtained:

$$-\sin\theta_1 \frac{l_1}{2} + x_1 = 0 \qquad \cos\theta_1 \frac{l_1}{2} + y_1 = 0$$

$$x_1 + \sin\theta_1 \frac{l_1}{2} + \sin\theta_2 \frac{l_2}{2} - x_2 = 0 \qquad y_1 - \cos\theta_1 \frac{l_1}{2} - \cos\theta_2 \frac{l_2}{2} - y_2 = 0$$

On matrix form, this yields

$$\boldsymbol{\Phi} = \begin{pmatrix} -\sin\theta_1 \frac{l_1}{2} + x_1 \\ \cos\theta_1 \frac{l_1}{2} + y_1 \\ x_1 + \sin\theta_1 \frac{l_1}{2} + \sin\theta_2 \frac{l_2}{2} - x_2 \\ y_1 - \cos\theta_1 \frac{l_1}{2} - \cos\theta_2 \frac{l_2}{2} - y_2 \end{pmatrix} = \mathbf{0} \qquad \dot{\boldsymbol{\Phi}} = \begin{pmatrix} \dot{x}_1 - \cos\theta_1 (\dot{\theta}_1) \frac{l_1}{2} \\ \dot{y}_1 - \sin\theta_1 (\dot{\theta}_1) \frac{l_1}{2} \\ \dot{x}_1 - \dot{x}_2 + \cos\theta_1 (\dot{\theta}_1) \frac{l_1}{2} + \cos\theta_2 (\dot{\theta}_2) \frac{l_2}{2} \\ \dot{y}_1 - y_2 + \sin\theta_1 (\dot{\theta}_1) \frac{l_1}{2} + \sin\theta_2 (\dot{\theta}_2) \frac{l_2}{2} \end{pmatrix} = \mathbf{0}$$

$$\ddot{\boldsymbol{\Phi}} = \begin{pmatrix} \ddot{x}_1 - \cos\theta_1 \ddot{\theta}_1 \frac{l_1}{2} + \sin\theta_1 (\dot{\theta}_1)^2 \frac{l_1}{2} \\ \ddot{y}_1 - \sin\theta_1 \ddot{\theta}_1 \frac{l_1}{2} - \cos\theta_1 (\dot{\theta}_1)^2 \frac{l_1}{2} \\ \ddot{x}_1 - \ddot{x}_2 - \sin\theta_1 (\dot{\theta}_1)^2 \frac{l_1}{2} + \cos\theta_1 \ddot{\theta}_1 \frac{l_1}{2} - \sin\theta_2 (\dot{\theta}_2)^2 \frac{l_1}{2} + \cos\theta_2 \ddot{\theta}_2 \frac{l_2}{2} \\ \ddot{y}_1 - \ddot{y}_2 + \cos\theta_1 (\dot{\theta}_1)^2 \frac{l_1}{2} + \sin\theta_1 \ddot{\theta}_1 \frac{l_1}{2} + \cos\theta_2 (\dot{\theta}_2)^2 \frac{l_1}{2} + \sin\theta_2 \ddot{\theta}_2 \frac{l_2}{2} \end{pmatrix} = \mathbf{0}$$

These expressions can be rewritten to obtain the Jacobian and it's time derivative.

$$\dot{\boldsymbol{\Phi}} = \begin{pmatrix} \dot{x}_1 - \cos\theta_1(\dot{\theta}_1)\frac{l_1}{2} \\ \dot{y}_1 - \sin\theta_1(\dot{\theta}_1)\frac{l_1}{2} \\ \dot{x}_1 - \dot{x}_2 + \cos\theta_1(\dot{\theta}_1)\frac{l_1}{2} + \cos\theta_2(\dot{\theta}_2)\frac{l_2}{2} \\ \dot{y}_1 - y_2 + \sin\theta_1(\dot{\theta}_1)\frac{l_1}{2} + \sin\theta_2(\dot{\theta}_2)\frac{l_2}{2} \end{pmatrix}$$

$$\underbrace{\begin{bmatrix} 1 & 0 & -\cos\theta_1\frac{l_1}{2} & 0 & 0 & 0 \\ 0 & 1 & -\sin\theta_1\frac{l_1}{2} & 0 & 0 & 0 \\ 1 & 0 & \cos\theta_1\frac{l_1}{2} & -1 & 0 & \cos\theta_2\frac{l_2}{2} \\ 0 & 1 & \sin\theta_1\frac{l_1}{2} & 0 & -1 & \sin\theta_2\frac{l_2}{2} \end{bmatrix}}_{\boldsymbol{D}\text{ (Jacobian)}} \underbrace{\begin{Bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{\theta}_1 \\ \dot{x}_2 \\ \dot{y}_2 \\ \dot{\theta}_2 \end{Bmatrix}}_{\dot{\boldsymbol{r}}} = \mathbf{0}$$

$$\ddot{\boldsymbol{\Phi}} = \begin{pmatrix} \ddot{x}_1 - \cos\theta_1\ddot{\theta}_1\frac{l_1}{2} + \sin\theta_1(\dot{\theta}_1)^2\frac{l_1}{2} \\ \ddot{y}_1 - \sin\theta_1\ddot{\theta}_1\frac{l_1}{2} - \cos\theta_1(\dot{\theta}_1)^2\frac{l_1}{2} \\ \ddot{x}_1 - \ddot{x}_2 - \sin\theta_1(\dot{\theta}_1)^2\frac{l_1}{2} + \cos\theta_1\ddot{\theta}_1\frac{l_1}{2} - \sin\theta_2(\dot{\theta}_2)^2\frac{l_1}{2} + \cos\theta_2\ddot{\theta}_2\frac{l_2}{2} \\ \ddot{y}_1 - \ddot{y}_2 + \cos\theta_1(\dot{\theta}_1)^2\frac{l_1}{2} + \sin\theta_1\ddot{\theta}_1\frac{l_1}{2} + \cos\theta_2(\dot{\theta}_2)^2\frac{l_1}{2} + \sin\theta_2\ddot{\theta}_2\frac{l_2}{2} \end{pmatrix}$$

$$= \begin{bmatrix} 0 & 0 & \sin\theta_1\dot{\theta}_1\frac{l_1}{2} & 0 & 0 & 0 \\ 0 & 0 & -\cos\theta_1\dot{\theta}_1\frac{l_1}{2} & 0 & 0 & 0 \\ 0 & 0 & -\sin\theta_1\dot{\theta}_1\frac{l_1}{2} & 0 & 0 & -\sin\theta_2\dot{\theta}_2\frac{l_1}{2} \\ 0 & 0 & \cos\theta_1\dot{\theta}_1\frac{l_1}{2} & 0 & 0 & \cos\theta_2\dot{\theta}_2\frac{l_1}{2} \end{bmatrix} \begin{Bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{\theta}_1 \\ \dot{x}_2 \\ \dot{y}_2 \\ \dot{\theta}_2 \end{Bmatrix} + \begin{bmatrix} 1 & 0 & -\cos\theta_1\frac{l_1}{2} & 0 & 0 & 0 \\ 0 & 1 & -\sin\theta_1\frac{l_1}{2} & 0 & 0 & 0 \\ 1 & 0 & \cos\theta_1\frac{l_1}{2} & -1 & 0 & \cos\theta_2\frac{l_2}{2} \\ 0 & 1 & \sin\theta_1\frac{l_1}{2} & 0 & -1 & \sin\theta_2\frac{l_2}{2} \end{bmatrix} \begin{Bmatrix} \ddot{x}_1 \\ \ddot{y}_1 \\ \ddot{\theta}_1 \\ \ddot{x}_2 \\ \ddot{y}_2 \\ \ddot{\theta}_2 \end{Bmatrix}$$

$$\dot{\boldsymbol{D}} = \begin{bmatrix} 0 & 0 & \sin\theta_1\dot{\theta}_1\frac{l_1}{2} & 0 & 0 & 0 \\ 0 & 0 & -\cos\theta_1\dot{\theta}_1\frac{l_1}{2} & 0 & 0 & 0 \\ 0 & 0 & -\sin\theta_1\dot{\theta}_1\frac{l_1}{2} & 0 & 0 & -\sin\theta_2\dot{\theta}_2\frac{l_1}{2} \\ 0 & 0 & \cos\theta_1\dot{\theta}_1\frac{l_1}{2} & 0 & 0 & \cos\theta_2\dot{\theta}_2\frac{l_1}{2} \end{bmatrix}$$

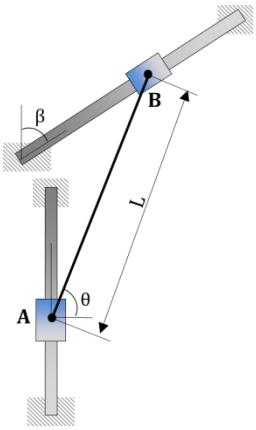
Code Example 7.1: The double pendulum

```
PlotCoGs= "on";      PlotCoGTrcs="on";    PlotBdbs= "off";
Bdb="off";
Bdy2BdyCts="off";    Bdy2WldCts="off";    SolverType="Var";
PltLim=[-1,2.2,0,2.5];
%Contact Parameters
PltFct=1;            %Plot each time step
nt=1600;             %Number of time steps
tlim=6;              %Time limit
%Generate geometry, first rod
Pts_Geb1=[0,0;0,0.05;0.7,0.05;0.7,0];
TriLns_Geb1=[1,2,3; 1,3,4];
IniCnd=[1,1.8,0,0,0,0];
GenGebBdy(IniCnd,2,1/12*2*0.7^2,Pts_Geb1,TriLns_Geb1)
% IniCnd=[1,1.8,0,0,0,0];
% GenLnsBdy(IniCnd,2,1/12*2*0.7^2,0.7);
%Generate geometry, second rod rod
Pts_Geb1=[0,0;0,0.05;0.7,0.05;0.7,0];
TriLns_Geb1=[1,2,3; 1,3,4];
IniCnd=[1.7,1.8,0,0,0,0];
GenGebBdy(IniCnd,2,1/12*2*0.7^2,Pts_Geb1,TriLns_Geb1)
% IniCnd=[1.7,1.8,0,0,0,0];
% GenLnsBdy(IniCnd,2,1/12*2*0.7^2,0.7);
%Generate Joints
Bdy2WldPinJnt(BdyNum-1,[-0.35,0])
Bdy2BdyPinJnt(BdyNum,[-0.35,0],BdyNum-1,[0.35,0])
```

8. Function list

[To be updated]

Problems



Computational MBD, problem 1

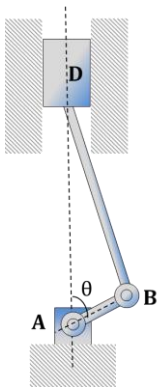
Collar B moves downwards with a constant velocity v_B=1.2 m/s in a system with parameters L=0.4 m and β=50 deg. At the instant shown when θ=60 deg, determine the angular velocity of AB, the velocity of A and the velocity of the CoG of AB using the vector algebra and rotation matrix in this section

Using classical vector algebra is considered cheating.

Ans:

ω_{AB}=2.654$\frac{rad}{s}$(CW), v_A =1.302 $\frac{m}{s}$

v_x^G = 1.457 $\frac{m}{s}$, v_y^G = 0.572 $\frac{m}{s}$



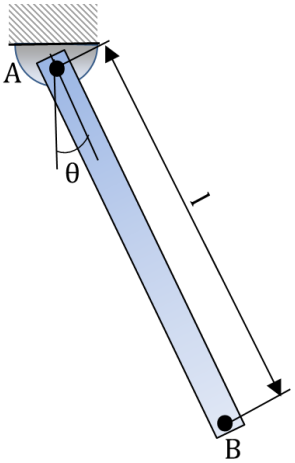
Computational MBD, problem 2

In the engine shown, l_{BD}=270 mm and r_{AB}=110 mm. AB rotates with a constant clockwise speed of 2000 rpm. Determine the velocity of the piston and the angular velocity of the connecting rod when θ=90 deg

Using classical vector algebra is considered cheating

Ans:

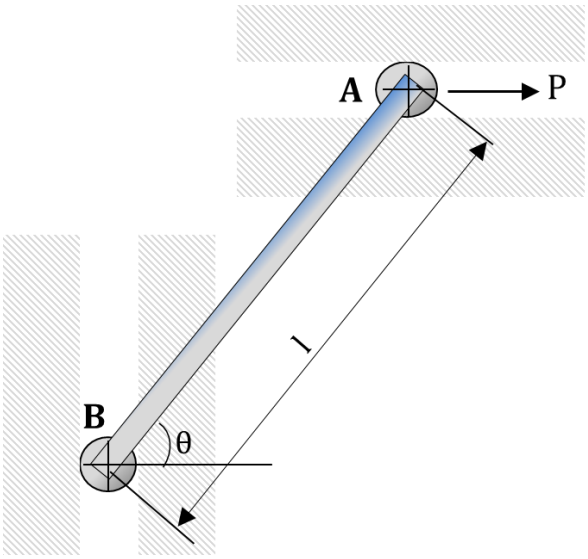
ω_{BD}=0, v_D=23.038 m/s



Computational MBD, problem 3

A rod of length l and with mass m is hinged in point A as shown in the figure. If released from rest with initial angle $\theta = \theta_0$ it constitutes a physical pendulum. Determine:

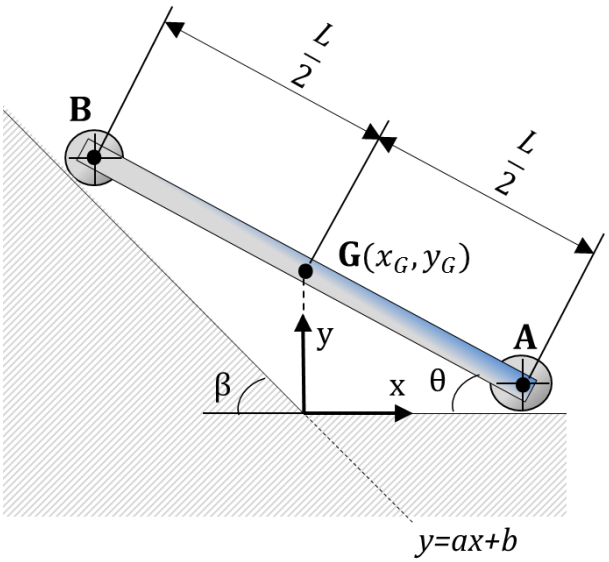
a) The required initial conditions for x_G and y_G
b) The equation of motion of the system for $\theta << 1$.
c) The period of the oscillating motion for $\theta << 1$
d) Implement the constraint system of equations (example 6.1) in Matlab in your own script. Compare the solution obtained by numerical integration with the analytical solution and the Kraken code.



Computational MBD, problem 4

For the uniform rod with mass *m* and length *l* subjected to the constant load *P*, determine:

a) The free-body diagram, the kinetic diagram and the force vector **h**
b) The Newton-Euler equations on matrix form in terms of the angle θ
c) The kinematic constraints in centroid coordinates
d) The system of equations required to solve for accelerations and reactions
e) Implement these equations with a numerical integrator using Matlab and compare with the analytical result



Computational MBD, problem 5

The uniform rod shown of length L and mass m is released from rest in the position shown.

a) The free-body diagram, the kinetic diagram and the force vector **h**
b) The Newton-Euler equations on matrix form in terms of the angle θ
c) The kinematic constraints in centroid coordinates
d) The system of equations required to solve for accelerations and reactions
e) Implement and solve the equations using Matlab

Computational MBD, problem 6

For the double pendulum, the considered mechanism can be converted into a force controlled crank-slider mechanism (reciprocating machine) by adding a single kinematic constraint. Determine:

a) The required constraint equation
b) The additional matrix equations needed to analyse the system
c) Appropriate boundary conditions
d) Implement the system of equations in Matlab
e) Modify the system of equations to include friction between the piston and the surrounding cylinder

Computational MBD, problem 6

The double pendulum can be converted to a triple pendulum by adding an additional body and two additional kinematic constraints. Determine:

a) The required constraint equations
b) The additional matrix equations needed to analyse the system
c) Appropriate boundary conditions
d) Implement the system of equations in Matlab

References

[1] P.E. Nikravesh, *Planar multibody dynamics*, 1st ed.: CRC Press, 2008.
[2] P.E. Nikravesh, *Computer-aided analysis of mechanical systems*, 1st ed.: Prentice Hall, 1989.
[3] S. Goyal, E.N. Pinson, F.W. Sinden, "Simulation of dynamics of interacting rigid bodies including friction Part I and II", in Engineering with Computers, vol. 10, 1994
[4] Y.F. Liu, Z.M. Zhang, X.H. Hu, W.J. Zhang, Experimental comparison of five friction models on the same test-bed of the micro stick-slip motion system, Mechanical Sciences, Mech. Sci. 6, 15-28, 2015
[5] N.H. Ostergaard and S. Danjou, On Numerical Simulation of the Dynamics of Bottles in Conveyor Systems, Journal of Applied Packaging Research, 2017
[6] N.H. Ostergaard and S. Danjou, Application of Computational MBD for Simulation of Wrao Packaging Performance, Journal of Applied Packaging Research, 2018
[7] Computational RBD in Package Engineering, Visualizations of obtained results – look them up on YouTube
[8] CarlaTheKraken Matlab source code https://drive.google.com/file/d/1-d4KqNSCOpH6VUZvRl2VnrhhVrbkliBv/view
[9] CarlaTheKraken on YouTube