University of Bergen

Thorstein R Rykkje

Master thesis in Applied and Computational Mathematics, Department of Mathematics, Spring 2018

Lie Groups and the Principle of Virtual Work Applied to Systems of Linked Rigid Bodies

A generic algorithm for the equations of motion using the moving frame method in engineering

Author:
Thorstein R Rykkje

Supervisor:
Antonella Zanna Munthe-Kaas

Supervisor:
Thomas J. Impelluso

1 Preface and Acknowledgement

This thesis is the culmination of all of my experience and knowledge up to this point in time. I started my education in mechanics, by getting an Advanced Craft Certificate as a Construction Machine Mechanic. After working a few years I decided to move forward and took a bachelor degree in mechanical engineering, before applying to the masters program at UiB. To write this thesis I had to use more than my knowledge in mathematics and computation. I had to use my skills from engineering to plan and model the parts needed for the pendulum, I had to use my skills as a mechanic to produce some of the parts my self and finally assemble the construction. Thus I think that this thesis reflects me and my knowledge quite well.

I want to thank my supervisors Antonella and Thomas for their patience, time and invaluable guidance in the making of this thesis.

Furthermore, I would like to thank the three bachelor students Daniel Leinebø, Erlend Sande Bergaas and Andreas Skjelde that assisted me in the building and measuring of the pendulum as well as the engineers in the workshop at HVL that gave valuable advice and helped produce the parts needed for the pendulum.

I want to give a special thanks to friends, family and especially my girlfriend Siri, who has stood by me and not given up on me even though my head has been thoroughly buried in my thesis this past year.

2 Abstract

This thesis explores the possibility of a generic algorithm for systems of linked rigid bodies using the moving frame method (MFM) in engineering developed by H. Murakami and T.Impelluso. The project entails the construction of a generic algorithm for the equations of motion and the validation of the equations generated by said algorithm. The validation is done by comparing the equations of motion generated by the algorithm to equations evaluated manually. Furthermore, the resulting behavior from integration is compared to those from the Hamilton canonical equations of motion. Finally, a real-life model is built to see if the theory holds in reality. Naturally, when taking the step to bring in reality, friction and air resistance will have to be taken into account. Therefore, the equations of motion were supplemented with friction models constructed to match the movement of the real-life model.

Contents

List of Figures

3 Introduction

The moving frame method in engineering developed by H. Murakami and T.J. Impelluso modernizes the mathematics and pedagogy of rigid body mechanics [7], [8]. It does this by leveraging Lie Group Theory, Cartan's notion of moving frames and a new notation from geometrical physics [5], [6]. Next, it establishes a restriction on the variation of the angular velocity. Together with the principle of virtual work, the method gives a structured way of modelling systems of linked rigid bodies. While some may question the need to revisit the mathematical underpinnings of an established discipline (with concomitant tried and tested coding implementations), we are now at a dawn of Artificial Intelligence: machines not only think and communicate, they will learn. Thus, many of the legacy software used in industry must be rewritten and in more modular form. The moving frame method [1], with its streamlining of the equations of motion, empowers robust and agile coding.

As a prelude to a project at Western Norway Univeristy of Applied sciences(HVL) to create an engineering program in order to analyze motion, loads and bending on dynamical systems, this thesis seeks to explore the possibilities of a generic algorithm for the equations of motion for systems of linked rigid bodies, as well as validate the models constructed using this algorithm. The validation is done by comparing the resulting equations from the algorithm to equations evaluated by hand, as well as comparing the resulting behaviour to that of the Hamilton canonical equations of motion. Further more, The author has cooperated with a bachelor group at HVL in order to build a real-life model to compare the calculations with.

While the model's prime objective for this thesis is to be used to validate the result, it also stands on its own as an example on how such a project could be implemented in teaching. Now that the model itself is built, the process of predicting the behaviour could and should be implemented in advanced courses in dynamics to enhance the understanding and motivation of students. This has culminated into an article, an abstract has been sent and accepted by IMECE 2018 [11] with publication pending.

4 Theoretic background

The theoretic background of this thesis relies heavily on the research of H. Muracami and T.J Impelluso, and borrows from their pending text book in dynamics [1] to construct the equations of motion.

4.1 Notational convention

All rotation matrices, connection matrices and vectors in this thesis are relative to something. A rotation matrix describes a rotation from one vector basis to another, position vectors points to the position of one entity relative to another. Too keep track of all relative quantities the notational convention $((\alpha+1)/\alpha)$ is used. This is read as the $\alpha+1$ basis or body "as seen" from basis or body α. This notation will stand as a superscript, as in rotation matrices $R^{((\alpha+1)/\alpha)}$. With the special case when the basis or vector observed is relative to the inertial frame, in this case the inertial is not explicitly stated and the notation is as this: $R^{(\alpha)}$.

4.1.1 Some symbols

Following are some basic symbols used in the thesis.

$\theta^{(\alpha)}$ Angle vector of the of body alpha

$\boldsymbol{\omega}^{(\alpha)}$ Angular velocity vector of body α

$\omega_i^{(\alpha)}$ Angular velocity component i in the angular velocity vector of body α

$r^{(\alpha)}$ Position vector of the centre of mass of the α body

$r_i^{(\alpha)}$ Position vector component i of the centre of mass of the α body

$e^{(\alpha)}$ Vector basis in $\mathbb{R}^3$ corresponding to the orientation of the α body

$q^{(\alpha)}$ Essential generalized coordinate of body alpha associated to the freedom of the system

I Identity matrix

4.1.2 Hat operator

The hat operator [10, p.363] acts on a vector in $\mathbb{R}^3$ and returns the skew-symmetric matrix representation of said vector. In this thesis, it is most commonly used as $\widehat{\omega}$, which is a skew-symmetric matrix that is isomorphic to the angular velocity vector of the basis.

$$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\end{bmatrix} = \boldsymbol{\omega} \Rightarrow \widehat{\omega} = \begin{bmatrix}0 & -\omega_3 & \omega_2\\ \omega_3 & 0 & -\omega_1\\ -\omega_2 & \omega_1 & 0\end{bmatrix} \tag{1}$$

The hat operator can be used as a substitute for the cross product, as such the tangential velocity of a point can be found by:

$$\omega \times r = \begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\end{bmatrix} \times \begin{bmatrix}r_1\\ r_2\\ r_3\end{bmatrix} = \begin{bmatrix}0 & -\omega_3 & \omega_2\\ \omega_3 & 0 & -\omega_1\\ -\omega_2 & \omega_1 & 0\end{bmatrix} \begin{bmatrix}r_1\\ r_2\\ r_3\end{bmatrix} = \widehat{\omega} r \tag{2}$$

The hat operator has several identities with the following being used in the thesis:

$$[\widehat{x}, \widehat{y}] = \widehat{x}\widehat{y} - \widehat{y}\widehat{x} = \widehat{x \times y} \tag{3}$$

where $x, y \in \mathbb{R}^3$ and $[,]$ are the Lie bracket of the special orthogonal group.

4.2 Lie Groups

A lie group G is a smooth manifold with a product $\cdot$ such that:

$\rightarrow$ $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ Associativity $\forall x, y, z \in G$

$\rightarrow$ $\exists e \in G$ $s.t$ $x \cdot e = e \cdot x = x$ Identity element $\forall x \in G$

$\rightarrow$ $\exists x^{-1} \in G$ $s.t$ $x \cdot x^{-1} = x^{-1} \cdot x = e$ Inverse element $\forall x \in G$

And the maps $x \cdot y$ *and* x^{-1} are smooth

A lie algebra $\mathfrak{g}$ is a vector space endowed with the bracket operation that satisfy the following conditions:

$[A, B] = -[B, A]$ Skew symmetry

$[\alpha A + \beta B, C] = \alpha[A, C] + \beta[B, C]$ Bilinearity

$[A, [B, C]] + [B, [C, A]] + [C, [A, B]] = 0$ Jacobi Identity

A lie algebra $\mathfrak{g}$ that arise in conjunction with a Lie Group G, is the linear space of all tangents to G at the point e on the manifold, $\mathfrak{g} = T_e G$.

4.3 Lie Groups Applied to rigid bodies

4.3.1 Special Orthogonal Group

The Lie group central to this thesis is the Special Orthogonal Group applied to 3D space SO(3), and its Lie algebra. SO(3) is the group of rotational matrices $R(t) \in SO(3)$, that is matrices that are orthogonal and has a determinant equal to 1.

Applying the special orthogonal group to rigid bodies is the act of attaching vector bases to rigid bodies and using the group elements to relate them. Using the special orthogonal group this way makes the analysis of complicated system easier by enabling the user to split the systems into smaller manageable parts. [1, ch1-4]

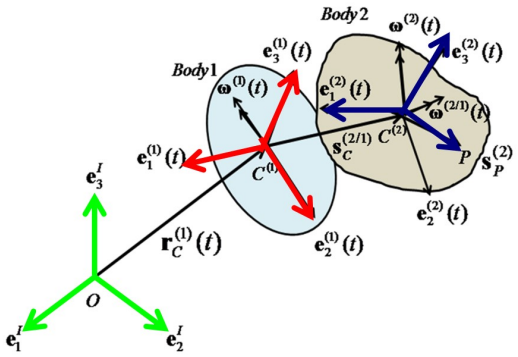


Figure 1: Two body related bases and an Inertial basis, [1, ch12]

Two rigid bodies, with body attached bases as well as an inertial base of reference.

$$\mathbf{e^{(1)}(t)} = (e_1^{(1)}(t), e_2^{(1)}(t), e_3^{(1)}(t)) \tag{4}$$

$$\mathbf{e^{(2)}(t)} = (e_1^{(2)}(t), e_2^{(2)}(t), e_3^{(2)}(t)) \tag{5}$$

$$\mathbf{e^{I}} = (e_1^I, e_2^I, e_3^I) \equiv e^{(1)}(0) \tag{6}$$

The second basis is given relative to the first and the first basis relative to the inertial. They are connected by rotational a matrices $R^{(\alpha)}(t) \in SO(3)$

$$\mathbf{e^{(2)}(t)} = \mathbf{e^{(1)}(t)} R^{(2/1)}(t) \tag{7}$$

$$\mathbf{e^{(1)}(t)} = \mathbf{e^{I}} R^{(1)}(t) \tag{8}$$

The rate of change of the first basis with respect to time can be found relative to the inertial reference frame.

$$\dot{\mathbf{e}}^{\mathbf{(1)}}(\mathbf{t}) = \mathbf{e^{I}} \dot{R}^{(1)}(t) \tag{9}$$

Stating the rate of change of the basis with the basis itself gives the Lie algebra of the first basis $\widehat{\omega} \in \mathfrak{g}$.

$$\dot{\mathbf{e}}^{\mathbf{(1)}}(\mathbf{t}) = \mathbf{e^{(1)}(t)} {R^{(1)}}^{T}(t) \dot{R}^{(1)}(t) \equiv \mathbf{e}^{(1)}(t) \widehat{\omega}^{(1)}(t) \tag{10}$$

with:

$$\widehat{\omega}^{(1)}(t) \in T_I \mathcal{M} \equiv \mathfrak{g} \tag{11}$$

The $\widehat{\omega}$ is a skew-symmetric matrix that is isomorphic to the angular velocity vector of the basis. They are connected trough the hat operator, it acts upon a vector and transforms it to an equivalent skew symmetric matrix.

Continuing, the orientation of the second basis is given relative to the first:

$$\mathbf{e^{(2)}(t)} = \mathbf{e^{(1)}(t)} R^{(2/1)}(t) \tag{12}$$

the rate of change of the second basis

$$\dot{\mathbf{e}}^{\mathbf{(2)}}(\mathbf{t}) = \dot{\mathbf{e}}^{\mathbf{(1)}}(\mathbf{t}) R^{2/1}(t) + \mathbf{e^{(1)}} \dot{R}^{(2/1)}(t) \tag{13}$$

Relating the rate of change of basis 2 to itself and inserting the rate of the first:

$$\dot{\mathbf{e}}^{\mathbf{(2)}}(\mathbf{t}) = \mathbf{e^{(2)}(t)} ({R^{2/1}}^{T}(t) \widehat{\omega}^{(1)}(t) R^{2/1}(t) + {R^{2/1}}^{T}(t) \dot{R}^{(2/1)}(t)) \tag{14}$$

with:

$${R^{2/1}}^{T}(t) \widehat{\omega}^{(1)}(t) R^{2/1}(t) + {R^{2/1}}^{T}(t) \dot{R}^{(2/1)}(t) \equiv \widehat{\omega}^{(2)}(t) \in T_I \mathcal{M} \equiv \mathfrak{g} \tag{15}$$

The rate of change of the second basis given in its own space is:

$$\dot{\mathbf{e}}^{\mathbf{(2)}}(\mathbf{t}) = \mathbf{e^{(2)}(t)} \widehat{\omega}^{(2)}(t) \tag{16}$$

4.3.2 Simplification of the Adjoint operation

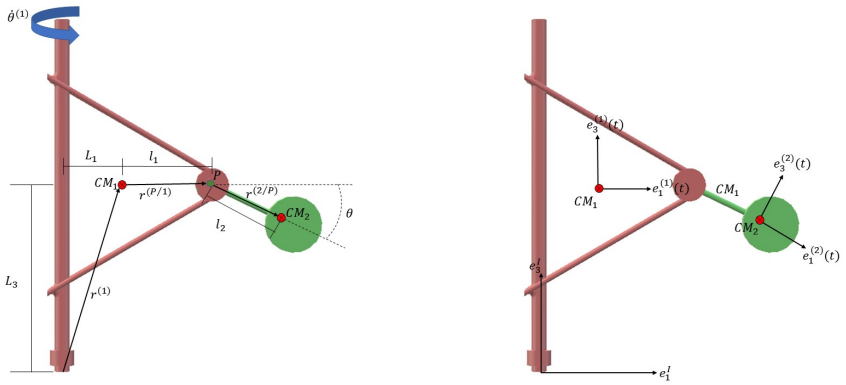
The adjoint operation can be simplified when using the special orthogonal group in 3D. The special case comes from the correspondence between the angular velocity matrix and the angular velocity vector. By stating the angular velocity as a vector, the adjoint operation can be replaced by a group element.

$$R^{(\alpha)^T}\widehat{\omega}^{(\alpha-1)}R^{(\alpha)} = \widehat{R^{(\alpha)^T}\omega^{(\alpha-1)}} \tag{17}$$

this simplifies the evaluation of the absolute angular velocities for many consecutive rotating bodies:

$$\boldsymbol{\omega}^{(\alpha)} = \mathbf{e}^{(\mathbf{1})}(\mathbf{t})\omega^{(1)} + \mathbf{e}^{(\mathbf{2})}(\mathbf{t})\omega^{(2/1)} + \cdots + \mathbf{e}^{(\alpha)}(\mathbf{t})\omega^{(\alpha/\alpha-1)} \tag{18}$$

Practical example SO(3)



(a) Pendulum with position vectors (b) Pendulum with the body attached vector bases

Figure 2: Pendulum where the red Column rotates around the inertial third direction, and the green arm about the local second direction

In this example, the goal is to find an expression for the acceleration of the centre of mass of the green body in figure 2. The first step is to define two body attached vector bases. The first is rotated around the third inertial

axis and related to the inertial with a rotational matrix.

$$e^{(1)}(t) = e^I R^{(1)}(t) = e^I \begin{bmatrix} \cos(\theta^{(1)}) & -\sin(\theta^{(1)}) & 0 \\ \sin(\theta^{(1)}) & \cos(\theta^{(1)}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{19}$$

where the inertial is defined as

$$e^I \equiv e^{(1)}(0) \tag{20}$$

and the second basis is rotated relative to the first with a rotational matrix $R^{(2/1)}$ that rotates around the local second axis.

$$e^{(2)}(t) = e^{(1)}(t) R^{(2/1)}(t) = e^{(1)}(t) \begin{bmatrix} \cos(\theta^{(2)}) & 0 & \sin(\theta^{(2)}) \\ 0 & 1 & 0 \\ -\sin(\theta^{(2)}) & 0 & \cos(\theta^{(2)}) \end{bmatrix} \tag{21}$$

Rate of change of the first basis with respect to time

(22)

$$\dot{e}^{(1)}(t) = e^I \dot{R}^{(1)}(t) = e^{(1)}(t) {R^{(1)}}^T(t) \dot{R}^{(1)}(t) = e^{(1)}(t) \begin{bmatrix} 0 & -\dot{\theta}^{(1)} & 0 \\ \dot{\theta}^{(1)} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \equiv e^{(1)}(t) \widehat{\omega}^{(1)}(t)$$

Rate of change of the second basis with respect to time

(23)

$$\dot{e}^{(2)}(t) = \dot{e}^{(1)}(t) R^{(2/1)}(t) + e^{(1)}(t) \dot{R}^{(2/1)}(t) = e^{(1)}(t) \widehat{\omega}^{(1)}(t) R^{(2/1)}(t) + e^{(1)}(t) \dot{R}^{(2/1)}(t) =$$

$$e^{(2)}(t) \left({R^{(2/1)}}^T(t) \widehat{\omega}^{(1)}(t) R^{(2/1)}(t) + {R^{(2/1)}}^T(t) \dot{R}^{(2/1)}(t) \right) =$$

$$e^{(2)}(t) \begin{bmatrix} 0 & \dot{\theta}^{(1)} sin(\theta^{(2)}) & \dot{\theta}^{(2)} \\ -\dot{\theta}^{(1)} sin(\theta^{(2)}) & 0 & -\dot{\theta}^{(1)} cos(\theta^{(2)}) \\ -\dot{\theta}^2 & \dot{\theta}^{(1)} cos(\theta^{(2)}) & 0 \end{bmatrix} \equiv e^{(2)}(t) \widehat{\omega}^{(2)}(t)$$

The position of the centre of mass of the pendulum can be found by stating the position vectors in figure 2a spanning from the inertial frame of reference to the points in the respective vector basis and adding them together.

$$P_{cm_2}(t) = \mathbf{e^{(1)}}(\mathbf{t}) r^{(P)} + \mathbf{e^{(2)}}(\mathbf{t}) r^{(2/P)} \tag{24}$$

where $r^{(P)} = r^{(1)} + r^{(P/1)}$.

The tangential velocity is found by differentiating (24) with respect to time.

$$\frac{dP_{cm_2}(t)}{dt} = v_{cm_2}(t) = \dot{\mathbf{e}}^{(\mathbf{1})}(\mathbf{t}) r^{(P)} + \dot{\mathbf{e}}^{(\mathbf{2})}(\mathbf{t}) r^{(2/P)} = \tag{25}$$
$$\mathbf{e^{(1)}}(\mathbf{t}) \widehat{\omega}^{(1)}(t) r^{(P)} + \mathbf{e^{(2)}}(\mathbf{t}) \widehat{\omega}^{(2)}(t) r^{(2/P)}$$

The acceleration is found by differentiating (25) with respect to time.

$$\begin{aligned}\frac{dv_{cm_2}(t)}{dt} = a_{cm_2} = \mathbf{e^{(1)}}(\mathbf{t})\left(\widehat{\omega}^{(1)}(t)\widehat{\omega}^{(1)}(t)r^{(P)} + \dot{\widehat{\omega}}^{(1)}(t)r^{(P)}\right) + \\ \mathbf{e^{(2)}}(\mathbf{t})\left(\widehat{\omega}^{(2)}(t)\widehat{\omega}^{(2)}(t)r^{(2/P)} + \dot{\widehat{\omega}}^{(2)}(t)r^{(2/P)}\right)\end{aligned} \tag{26}$$

Using the relation between the bases (21) to relate all the components in the first frame

$$\begin{aligned}a_{cm_2} =& \mathbf{e^{(1)}}(\mathbf{t})\left(\widehat{\omega}^{(1)}(t)\widehat{\omega}^{(1)}(t)r^{(P)} + \dot{\widehat{\omega}}^{(1)}(t)r^{(P)} + \right. \\ & \left. R^{(2/1)}\left(\widehat{\omega}^{(2)}(t)\widehat{\omega}^{(2)}(t)r^{(2/P)} + \dot{\widehat{\omega}}^{(2)}(t)r^{(2/P)}\right)\right)\end{aligned} \tag{27}$$

4.3.3 Special Euclidean Group

The Special Euclidean group expands upon the Special orthogonal group to encompass the position of the origin of the basis as well as the orientation. The bases are denoted with a vector basis and position of the origin $(\mathbf{e}^{(\alpha)}, \mathbf{r}^{(\alpha)})$. In SE(3) group elements $E^{(\alpha)}(t)$ are 4×4 matrices called connection matrices. They are block matrices that hold a rotational matrix in the upper left and a position vector in the upper right. Where the rotational matrix hold the relative orientation between frames and the vector hold the relative position of the origin. [1, ch13]

(28)
$$(e^{((\alpha+1)/\alpha)}, r^{((\alpha+1)/\alpha)}) = (e^{(\alpha)}, r^{(\alpha)})E^{((\alpha+1)/\alpha)} = (e^{(\alpha)}, r^{(\alpha)}) \begin{bmatrix} R^{((\alpha+1)/\alpha)} & r^{((\alpha+1)/\alpha)} \\ 0^T & 1 \end{bmatrix}$$

where:

- $R^{(\alpha)} \in SO(3)$
- $0^T = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$
- $r^{(\alpha)} = \begin{bmatrix} r_1^{(\alpha)} \\ r_2^{(\alpha)} \\ r_3^{(\alpha)} \end{bmatrix} \in \mathbb{R}^3$

SE(3) is closed under multiplication:

(29)
$$x \cdot y = z \in SE(n) \quad \rightarrow \forall x, y \in SE(n)$$

the Lie bracket of SE(3) is defined as:

(30)
$$[x, y] = xy - yx$$

The inverse element of the special Euclidean group in 3D has a known form and is found the following way:.

Let $E^{\alpha} \in SE(3)$ be the connection matrix, and Multiplying the Connection matrix with its inverse ${E^{(\alpha)}}^{-1} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$:

(31)
$$\begin{aligned} E^{(\alpha)}{E^{(\alpha)}}^{-1} &= \underset{4\times4}{I} \\ \begin{bmatrix} R^{(\alpha)} & r^{(\alpha)} \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} A & B \\ C & D \end{bmatrix} &= \begin{bmatrix} I & 0 \\ 0^T & 1 \end{bmatrix} \\ \begin{bmatrix} R^{(\alpha)}A + r^{(\alpha)}C & R^{(\alpha)}B + r^{(\alpha)}D \\ C & D \end{bmatrix} &= \begin{bmatrix} I & 0 \\ 0^T & 1 \end{bmatrix} \end{aligned}$$

This gives the elements of the inverse:

- $C = 0^T$
- $D = 1$
- $R^{(\alpha)}A = I \Rightarrow A = R^{(\alpha)^T}$
- $R^{(\alpha)}B + r^{(\alpha)}D = 0 \Rightarrow B = -R^{(\alpha)^T}r^{(\alpha)}$

and thus the inverse element of SE(3) must be:

$$E^{(\alpha)^{-1}} = \begin{bmatrix} R^{(\alpha)^T} & -R^{(\alpha)^T}r^{(\alpha)} \\ 0^T & 1 \end{bmatrix} \tag{32}$$

4.3.4 Time Rate of Change of SE(3)

The time rate of change of the Special Euclidean group follows the same pattern as the Special Orthogonal group. Find the rate of change with respect to time and relate it to its own basis with the inverse element.

$$\frac{d}{dt}(e^{(\alpha)}(t), r_c^{(\alpha)}(t)) = (e^I, 0)\dot{E}^{(\alpha)}(t) = (e^{(\alpha)}(t), r_c^{(\alpha)}(t))E^{(\alpha)^{-1}}(t)\dot{E}^{(\alpha)}(t) \tag{33}$$

with $E^{(\alpha)^{-1}}(t)\dot{E}^{(\alpha)}(t) \equiv \Omega^{(\alpha)}(t) \in \mathfrak{g}$

$$\begin{aligned} &\frac{d}{dt}\left(e^{(\alpha)}(t), r_c^{(\alpha)}(t)\right) = (e^{(\alpha)}(t), r^{(\alpha)}(t))\Omega^{(\alpha)}(t) = \\ &(e^{(\alpha)}(t), r_c^{(\alpha)}(t)) \begin{bmatrix} \widehat{\omega}^{(\alpha)}(t) & \widehat{\omega}^{(\alpha)}(t)r^{(\alpha)}(t) + \dot{r}^{(\alpha)}(t) \\ 0^T & 0 \end{bmatrix} \end{aligned} \tag{34}$$

The rate of change of the $(\alpha + 1)$ basis

$$\begin{aligned} &\frac{d}{dt}(e^{(\alpha+1)}(t), r^{(\alpha+1)}(t)) = \frac{d}{dt}(e^{(\alpha)}(t), r^{(\alpha)}(t))E^{(\alpha+1/\alpha)}(t) + (e^{(\alpha)}(t), r^{(\alpha)}(t))\dot{E}^{(\alpha+1/\alpha)}(t) = \\ &(e^{(2)}(t), r^{(2/1)}(t))\left(E^{(\alpha+1/\alpha)^{-1}}(t)\Omega^{(\alpha)}(t)E^{(\alpha+1/\alpha)}(t) + E^{(\alpha+1/\alpha)^{-1}}(t)\dot{E}^{(\alpha+1/\alpha)}(t)\right) \end{aligned} \tag{35}$$

with:

- $E^{(\alpha+1/\alpha)^{-1}}(t)\dot{E}^{(\alpha+1/\alpha)}(t) \equiv \Omega^{(\alpha+1/\alpha)}(t) \in \mathfrak{g}$
- $(E^{(\alpha+1/\alpha)^{-1}}(t)\Omega^{(\alpha)}(t)E^{(\alpha+1/\alpha)}(t) + \Omega^{(\alpha+1/\alpha)}(t)) \equiv \Omega^{(\alpha+1)} \in \mathfrak{g}$

The rate of change of the $\alpha + 1$ basis becomes

(36)
$$\frac{d}{dt}(e^{(\alpha+1)}(t), r^{(\alpha+1)}(t)) = (e^{(\alpha+1)}(t), r^{(\alpha+1)}(t))\Omega^{(\alpha+1)}(t) =$$
$$(e^{(\alpha+1)}, r^{\alpha+1)}) \begin{bmatrix} \widehat{\omega}^{(\alpha+1)} & {R^{(\alpha+1/\alpha)}}^T(\widehat{\omega}^{(\alpha)} r^{(\alpha+1)} + \dot{r}^{(\alpha)}) + \widehat{\omega}^{(\alpha+1/\alpha)} r^{(\alpha+1/\alpha)} + \dot{r}^{(\alpha+1/\alpha)} \\ 0^T & 0 \end{bmatrix}$$

with

- $r^{(\alpha+1)} = r^{(\alpha+1/\alpha)} + r^{(\alpha)}$
- $\widehat{\omega}^{(\alpha+1)} = {R^{(\alpha+1/\alpha)}}^T \widehat{\omega}^{(\alpha)} R^{(\alpha+1/\alpha)} + \widehat{\omega}^{(\alpha+1/\alpha)}$

Example SE(3)

In this example the goal is to find an expression for the tangential velocity and angular velocity of both bodies in figure 2. To achieve this, the first step is to find the expression for the position of the centre of mass of both bodies.

Positions

Start by defining the vector bases needed and the position of their origins. The first vector basis is found relative to an inertial basis by multiplying two connection matrices together. The first holding the rotational matrix of the red column (19) and the second holding the vector from the lower bearing to the centre of mass of the column.

(37)
$$\begin{aligned}(e^{(1)}(t), r_c^{(1)}(t)) =& (e^I, 0) \begin{bmatrix} R^{(1)}(t) & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} I & r^{(1)} \\ 0^T & 1 \end{bmatrix} = \\ & (e^I, 0) \begin{bmatrix} R^{(1)}(t) & R^{(1)}(t) r^{(1)} \\ 0^T & 1 \end{bmatrix} = (e^I, 0) E^{(1)}(t)\end{aligned}$$

with the inertial basis and its origin defined as

(38)
$$(e^I, 0) \equiv (e^{(1)}(0), 0)$$

Furthermore the second basis is defined relative to the first by three connection matrices. The first translates from the first centre of mass to the point

P, second rotates around the local second axis (21) and third translates from P to the second centre of mass.

(39)
$$(e^{(2)}(t), r_c^{(2)}(t)) = (e^{(1)}(t), r_c^{(1)}(t)) \begin{bmatrix} I & r^{(P/1)} \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} R^{(2/1)}(t) & 0^T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I & r^{(2/P)} \\ 0^T & 1 \end{bmatrix} =$$
$$(e^{(1)}(t), r_c^{(1)}(t)) \begin{bmatrix} R^{(2/1)}(t) & r^{(P/1)} + R^{(2/1)}(t) r^{(2/P)} \\ 0^T & 1 \end{bmatrix} = (e^{(1)}(t), r_c^{(1)}(t)) E^{(2/1)}(t)$$

Velocities

By differentiating the first basis with respect to time, and relating the time derivative to the first basis, the tangential and angular velocity of the red column is found.

(40)
$$\frac{d}{dt}(e^{(1)}(t), r_c^{(1)}(t)) = (e^I, 0)\dot{E}^{(1)}(t) = (e^{(1)}(t), r_c^{(1)}(t)) E^{(1)}(t)^{-1} \dot{E}^{(1)}(t) = (e^{(1)}(t), r_c^{(1)}(t)) \Omega^{(1)}(t)$$

with:

(41)
$$\Omega^{(1)}(t) \equiv \begin{bmatrix} \widehat{\omega}^{(1)}(t) & \widehat{\omega}^{(1)}(t) r^{(1)} \\ 0^T & 0 \end{bmatrix}$$

The angular velocity matrix of the first basis can now be extracted from the upper left and the tangential velocity of the first centre of mass from the velocity vector in the upper right of $\Omega^{(1)}(t)$. Similarly the velocity of the second body can be found by differentiating the basis of the second body with respect to time and relate it to its own basis.

(42)
$$\frac{d}{dt}(e^{(2)}(t), r_c^{(2)}(t)) = \frac{d}{dt}(e^{(1)}(t), r_c^{(1)}(t)) E^{(2/1)}(t) + (e^{(1)}(t), r_c^{(1)}(t)) \dot{E}^{(2/1)}(t) =$$
$$(e^{(2)}(t), r_c^{(2)}(t)) \left(E^{(2/1)}(t)^{-1} \Omega^{(1)}(t) E^{(2/1)}(t) + E^{(2/1)}(t)^{-1} \dot{E}^{(2/1)}(t) \right) =$$
$$(e^{(2)}(t), r_c^{(2)}(t)) \Omega^{(2)}(t)$$

with:

(43)
$$\Omega^{(2)}(t) \equiv \begin{bmatrix} \widehat{\omega}^{(2)}(t) & R^{(2/1)^T}(t) \widehat{\omega}^{(1)}(t) r^{(2)}(t) + \widehat{\omega}^{(2/1)}(t) r^{(2/P)} \\ 0^T & 0 \end{bmatrix}$$

where:

- $\widehat{\omega}^{(2)}(t) = {R^{(2/1)}}^T(t)\widehat{\omega}^{(1)}(t)R^{(2/1)}(t) + \widehat{\omega}^{(2/1)}(t)$
- $\widehat{\omega}^{(2/1)}(t) = {R^{(2/1)}}^T(t)\dot{R}^{(2/1)}(t)$
- $r^{(2)}(t) = r^{(1)} + r^{(P/1)} + R^{(2/1)}(t)r^{(2/P)}$

The angular velocity matrix of the second basis can now be extracted from the upper left and the tangential velocity of the second centre of mass from the upper right.

4.4 Calculus of Variation

4.4.1 Variation of SE(3)

Taking the variation of a moving frame follows the same pattern as the time derivative. [1, ch13]

$$\begin{aligned}(\delta e^{(\alpha)}, \delta r^{(\alpha)}) &= (e^I, 0)\delta E^{(\alpha)} \\ (\delta e^{(\alpha)}, \delta r^{(\alpha)}) &= (e^{(\alpha)}, r^{(\alpha)}){E^{(\alpha)}}^{-1}\delta E^{(\alpha)} \\ (\delta e^{(\alpha)}, \delta r^{(\alpha)}) &= (e^{(\alpha)}, r^{(\alpha)})\delta\Pi^{(\alpha)}\end{aligned} \tag{44}$$

where:

$$\delta\Pi^{(\alpha)} \equiv \begin{bmatrix} {R^{(\alpha)}}^T\delta R^{(\alpha)} & {R^{(\alpha)}}^T\delta r^{(\alpha)} \\ 0^T & 0 \end{bmatrix} \tag{45}$$

and:

$$\widehat{\delta\pi^{(\alpha)}(t)} \equiv {R^{(\alpha)}}^T\delta R^{(\alpha)} \tag{46}$$

the variation of the frame becomes:

$$\delta\Pi^{(\alpha)} = \begin{bmatrix} \widehat{\delta\pi^{(\alpha)}(t)} & {R^{(\alpha)}}^T\delta r^{(\alpha)} \\ 0^T & 0 \end{bmatrix} \tag{47}$$

4.4.2 Restriction of $\delta\Omega^{(\alpha)}(t)$

Enforcing the commutativity of the δ and the $\frac{d}{dt}$ operators give the restriction of the virtual time rate of the connection matrix $\Omega^{(\alpha)}$ by equating the time

derivative of the variation with the variation of the time derivative of the frame connection matrix.

$$\frac{d}{dt}(\delta e^{(\alpha)}, \delta r^{(\alpha)}) = \delta(\dot{e}^{(\alpha)}, \dot{r}^{(\alpha)}) \tag{48}$$

Evaluating the left hand side gives

$$\begin{aligned} &\frac{d}{dt}(\delta e^{(\alpha)}, \delta r^{(\alpha)}) = \frac{d}{dt}\left\{(e^{(\alpha)}, r^{(\alpha)})\delta\Pi^{(\alpha)}\right\} = \\ &(e^{(\alpha)}, r^{(\alpha)})\left\{\Omega^{(\alpha)}(t)\delta\Pi^{(\alpha)}(t) + \frac{d}{dt}\delta\Pi^{(\alpha)}(t)\right\} \end{aligned} \tag{49}$$

The right hand side

$$\begin{aligned} &\delta(\dot{e}^{(\alpha)}, \dot{r}^{(\alpha)}) = \delta\left\{(e^{(\alpha)}, r^{(\alpha)})\Omega^{(\alpha)}(t)\right\} = \\ &(e^{(\alpha)}, r^{(\alpha)})\left\{\delta\Pi^{(\alpha)}(t)\Omega^{(\alpha)}(t) + \delta\Omega^{(\alpha)}(t)\right\} \end{aligned} \tag{50}$$

inserting (49) and (50) into (48).

$$\Omega^{(\alpha)}(t)\delta\Pi^{(\alpha)}(t) + \frac{d}{dt}\delta\Pi^{(\alpha)}(t) = \delta\Pi^{(\alpha)}(t)\Omega^{(\alpha)}(t) + \delta\Omega^{(\alpha)}(t) \tag{51}$$

using the Lie bracket (30) gives the restriction of the virtual time rate of the frame connection matrix.

$$\delta\Omega^{(\alpha)}(t) = \frac{d}{dt}\delta\Pi^{(\alpha)}(t) + \left[\Omega^{(\alpha)}(t), \delta\Pi^{(\alpha)}(t)\right] \tag{52}$$

Expanding (52)

$$\begin{aligned} &\begin{bmatrix} \delta\widehat{\omega}^{(\alpha)} & \delta\dot{r}_c^{(\alpha)}(t) \\ 0^T & 0 \end{bmatrix} = \frac{d}{dt}\begin{bmatrix} \widehat{\delta\pi^{(\alpha)}(t)} & (R^{(\alpha)}(t))^T\delta r_c^{(\alpha)}(t) \\ 0^T & 0 \end{bmatrix} + \\ &\left[\begin{bmatrix} \widehat{\omega}^{(\alpha)} & (R^{(\alpha)}(t))^T\dot{r}_c^{(\alpha)} \\ 0^T & 0 \end{bmatrix}, \begin{bmatrix} \widehat{\delta\pi^{(\alpha)}(t)} & (R^{(\alpha)}(t))^T\delta r_c^{(\alpha)}(t) \\ 0^T & 0 \end{bmatrix}\right] \end{aligned} \tag{53}$$

Focusing on the upper left sub matrix of (52)

$$\delta\widehat{\omega}^{(\alpha)} = \frac{d}{dt}(\widehat{\delta\pi}^{(\alpha)}) + \left[\widehat{\omega}^{(\alpha)}, \widehat{\delta\pi}^{(\alpha)}\right] \tag{54}$$

By the Lie bracket (30) of the special orthogonal group and the identity 3 where the Lie bracket of two components of the lie algebra becomes the skew symmetric matrix of the cross product of the two as vectors (3).

$$\left[\widehat{\omega}^{(\alpha)}, \widehat{\delta\pi}^{(\alpha)}\right] = \widehat{\omega^{(\alpha)} \times \pi}^{(\alpha)} = \widehat{\widehat{\omega}^{(\alpha)}\pi^{(\alpha)}} \tag{55}$$

(54) becomes

$$\delta\widehat{\omega}_{(\alpha)} = \frac{d}{dt}(\widehat{\delta\pi}^{(\alpha)}(t)) + \widehat{\widehat{\omega}^{(\alpha)}\pi^{(\alpha)}} \tag{56}$$

in vector form

$$\delta\omega_{(\alpha)} = \frac{d}{dt}(\delta\pi^{(\alpha)}(t)) + \widehat{\omega}^{(\alpha)}(t)\pi^{(\alpha)}(t) \tag{57}$$

The upper right vector gives the trivial relation

$$\begin{gathered} \lim_{\epsilon\to 0} \frac{\partial^2 r_c^{(\alpha)}}{\partial t \partial \epsilon} = \lim_{\epsilon\to 0} \frac{\partial^2 r_c^{(\alpha)}}{\partial \epsilon \partial t} \\ \frac{d}{dt}\delta r_c^{(\alpha)}(t) = \delta \dot{r}_c^{(\alpha)}(t) \end{gathered} \tag{58}$$

(57) and (58) together in matrix form

$$\begin{Bmatrix} \delta\dot{r}_c^{(\alpha)}(t) \\ \delta\omega^{(\alpha)}(t) \end{Bmatrix} = \frac{d}{dt}\begin{Bmatrix} \delta r_c^{(\alpha)}(t) \\ \delta\pi^{(\alpha)}(t) \end{Bmatrix} + \begin{bmatrix} 0_{3\times 3} & 0_{3\times 3} \\ 0_{3\times 3} & \widehat{\omega}^{(\alpha)}(t) \end{bmatrix} \begin{Bmatrix} \delta r_c^{(\alpha)}(t) \\ \delta\pi^{(\alpha)}(t) \end{Bmatrix} \tag{59}$$

Using the definitions in (4.5) to generalize (59) to a system of n bodies

$$\delta\dot{X}(t) = \frac{d}{dt}\delta X(t) + D(t)\delta X(t) \tag{60}$$

4.5 Some definitions

Before continuing with the principle of virtual work and the equations of motion, some definitions must be made. For a system of linked rigid bodies, the coordinates for each body can be written as a $n \times 1$ vector. Following are the needed definitions for the equations of motion for a system of linked rigid bodies.

The system generalized coordinates:

$$X(t) = \begin{bmatrix} r_c^{(1)}(t) \\ \theta^{(1)}(t) \\ r_c^{(2)}(t) \\ \theta^{(2)}(t) \\ \vdots \\ r_c^{(n)}(t) \\ \theta^{(n)}(t) \end{bmatrix} \tag{61}$$

The systems generalized velocities:

$$\dot{X}(t) = \begin{bmatrix} \dot{r}_c^{(1)}(t) \\ \omega^{(1)}(t) \\ \dot{r}_c^{(2)}(t) \\ \omega^{(2)}(t) \\ \vdots \\ \dot{r}_c^{(n)}(t) \\ \omega^{(n)}(t) \end{bmatrix} \tag{62}$$

The system generalized virtual displacements:

$$\delta X(t) = \left[\frac{\partial}{\partial \epsilon} X(t, \epsilon)\Big|_{(\epsilon=0)}\right] = \begin{bmatrix} \delta r_c^{(1)}(t) \\ \delta \pi^{(1)}(t) \\ \delta r_c^{(2)}(t) \\ \delta \pi^{(2)}(t) \\ \vdots \\ \delta r_c^{(n)}(t) \\ \delta \pi^{(n)}(t) \end{bmatrix} \tag{63}$$

The system generalized virtual velocities:

$$\delta \dot{X}(t) = \left[\frac{\partial}{\partial \epsilon} \dot{X}(t, \epsilon)\Big|_{(\epsilon=0)}\right] = \begin{bmatrix} \delta \dot{r}_c^{(1)}(t) \\ \delta \omega^{(1)}(t) \\ \delta \dot{r}_c^{(2)}(t) \\ \delta \omega^{(2)}(t) \\ \vdots \\ \delta \dot{r}_c^{(n)}(t) \\ \delta \omega^{(n)}(t) \end{bmatrix} \tag{64}$$

The system essential generalized coordinates, with n^* being the degrees of freedom of the system:

$$\{q(t)\} = \begin{Bmatrix} q^{(1)}(t) \\ q^{(2)}(t) \\ \vdots \\ q^{(n^*)}(t) \end{Bmatrix} \tag{65}$$

The system essential generalized velocities:

$$\dot{q}(t) = \begin{bmatrix} \dot{q}^{(1)}(t) \\ \dot{q}^{(2)}(t) \\ \vdots \\ \dot{q}^{(n^*)}(t) \end{bmatrix} \tag{66}$$

The system essential generalized virtual displacements:

$$\delta q(t) = \begin{bmatrix} \delta q^{(1)}(t) \\ \delta q^{(2)}(t) \\ \vdots \\ \delta q^{(n^*)}(t) \end{bmatrix} \tag{67}$$

The system angular velocity matrix:

$$[D(t)] = \begin{bmatrix} 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & \dots & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & \widehat{\omega^{(1)}(t)} & 0_{3\times3} & 0_{3\times3} & \dots & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & \dots & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & \widehat{\omega^{(2)}(t)} & \dots & 0_{3\times3} & 0_{3\times3} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & \dots & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & \dots & 0_{3\times3} & \widehat{\omega^{(n)}(t)} \end{bmatrix} \tag{68}$$

The system mass matrix:

$$[M] = \begin{bmatrix} m^{(1)}I & \underset{3\times3}{0} & \underset{3\times3}{0} & \underset{3\times3}{0} & \dots & \underset{3\times3}{0} & \underset{3\times3}{0} \\ \underset{3\times3}{0} & J_c^{(1)} & \underset{3\times3}{0} & \underset{3\times3}{0} & \dots & \underset{3\times3}{0} & \underset{3\times3}{0} \\ \underset{3\times3}{0} & \underset{3\times3}{0} & m^{(2)}I & \underset{3\times3}{0} & \dots & \underset{3\times3}{0} & \underset{3\times3}{0} \\ \underset{3\times3}{0} & \underset{3\times3}{0} & \underset{3\times3}{0} & J_c^{(2)} & \dots & \underset{3\times3}{0} & \underset{3\times3}{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \underset{3\times3}{0} & \underset{3\times3}{0} & \underset{3\times3}{0} & \underset{3\times3}{0} & \dots & m^{(n)}I & \underset{3\times3}{0} \\ \underset{3\times3}{0} & \underset{3\times3}{0} & \underset{3\times3}{0} & \underset{3\times3}{0} & \dots & \underset{3\times3}{0} & J_c^{(n)} \end{bmatrix} \tag{69}$$

with I being the identity matrix:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{70}$$

4.5.1 B Matrix

For a system of n bodies the system generalized velocities and essential generalized velocities defined in (64), are linearly related through the $[B(t)]$ matrix. It is a $6n \times n^*$ matrix. Where n is the number of bodies in the system and $n^*$is degrees of freedom of the system.

$$\dot{X}(t) = B(t)\dot{q}(t) \tag{71}$$

The linear relation is a result of the linearity that comes from the chain rule in derivation and as such, the same matrix consequently relates the virtual generalized displacements and the virtual essential displacements.

$$\delta X(t) = B(t)\delta q(t) \tag{72}$$

Furthermore, to facilitate ease in the differentiating of the $[B(t)]$ matrix. The matrix must be constructed with the tangential velocity components stated in the inertial basis and the angular velocity components stated in the respective local basis. Doing this, simplifies the operation down to only differentiating the respective components, as the rate of change of the frames drops away. Demonstrating by using the α input to the matrix:

$$\dot{B}(t) = \begin{bmatrix} e^I \dot{v}^{(\alpha)} \\ e^{(\alpha)}(t)\left(\widehat{\omega}^{(\alpha)}\omega^{(\alpha)} + \dot{\omega}^{(\alpha)}\right) \end{bmatrix} = \begin{bmatrix} e^I \dot{v}^{(\alpha)} \\ e^{(\alpha)}(t)\dot{\omega}^{(\alpha)} \end{bmatrix} \tag{73}$$

with: $\widehat{\omega}^{(\alpha)}\omega^{(\alpha)} = 0$ as this is the cross product of two parallel vectors.

4.6 Principle of Virtual work

To find a general equation of motion for systems of linked rigid bodies, the principle of virtual work is used. This is to encompass the non-conservative forces and moments as well as the conservative ones. The action integral for the principle of virtual work are:

$$\int_{t_0}^{t_1} \delta K(t) + \delta W(t) dt = 0 \tag{74}$$

with $K(t)$ being the kinetic energy of the system and $W(t)$ being the work done on the system by external forces and moments.

4.6.1 Variation of the Kinetic Energy

The kinetic energy of one rigid body can be expressed in matrix form as

$$K^{(\alpha)}(t) = \frac{1}{2} \begin{pmatrix} \dot{x}_c^{(\alpha)}(t) \\ \omega^{(\alpha)}(t) \end{pmatrix}^T \begin{bmatrix} m^{(\alpha)} I & \underset{3\times3}{0} \\ \underset{3\times3}{0} & J_c^{(\alpha)} \end{bmatrix} \begin{pmatrix} \dot{x}_c^{(\alpha)}(t) \\ \omega^{(\alpha)}(t) \end{pmatrix} \tag{75}$$

Introducing ϵ into the functions of time, and taking the variation at $\epsilon = 0$

$$\begin{aligned} \delta K^{(\alpha)}(t) = & \frac{1}{2} \begin{pmatrix} \delta\dot{x}_c^{(\alpha)}(t) \\ \delta\omega^{(\alpha)}(t) \end{pmatrix}^T \begin{bmatrix} m^{(\alpha)} I & \underset{3\times3}{0} \\ \underset{3\times3}{0} & J_c^{(\alpha)} \end{bmatrix} \begin{pmatrix} \dot{x}_c^{(\alpha)}(t) \\ \omega^{(\alpha)}(t) \end{pmatrix} + \\ & \frac{1}{2} \begin{pmatrix} \dot{x}_c^{(\alpha)}(t) \\ \omega^{(\alpha)}(t) \end{pmatrix}^T \begin{bmatrix} m^{(\alpha)} I & \underset{3\times3}{0} \\ \underset{3\times3}{0} & J_c^{(\alpha)} \end{bmatrix} \begin{pmatrix} \delta\dot{x}_c^{(\alpha)}(t) \\ \delta\omega^{(\alpha)}(t) \end{pmatrix} \end{aligned} \tag{76}$$

Since the mass moment of inertia tensor is symmetrical, (76) can be simplified to:

$$K^{(\alpha)}(t) = \begin{pmatrix} \delta\dot{x}_c^{(\alpha)}(t) \\ \delta\omega^{(\alpha)}(t) \end{pmatrix}^T \begin{bmatrix} m^{(\alpha)} I & \underset{3\times3}{0} \\ \underset{3\times3}{0} & J_c^{(\alpha)} \end{bmatrix} \begin{pmatrix} \dot{x}_c^{(\alpha)}(t) \\ \omega^{(\alpha)}(t) \end{pmatrix} \tag{77}$$

using definitions at 4.5, (77) is generalized to n number of bodies.

$$\delta K(t) = \delta\dot{X}(t)^T M \dot{X}(t) \tag{78}$$

with the restriction to the virtual generalized velocities (60) and definition (64) the variation of the kinetic energy becomes:

$$\delta K(t) = \left(\frac{d}{dt}\delta X + D\delta X \right)^T M\dot{X}(t) \tag{79}$$

4.6.2 Virtual work

Work for a system of n bodies is defined as

$$W(t) = X(t)^T F \tag{80}$$

Where F is a $6n^* \times 1$ vector holding all applied forces and moments on the body.

Virtual work is:

$$\delta W(t) = \delta X(t)^T F \tag{81}$$

with (72)

$$\delta W(t) = [B(t)\delta q(t)]^T F \tag{82}$$

Distributing the transpose and using following definition

$$F^*(t) = [B(t)]^T F \tag{83}$$

the virtual work becomes:

$$\delta W(t) = \delta q(t)^T F^*(t) \tag{84}$$

4.6.3 Equation of motion

Inserting the expression for kinetic energy (79) and the virtual work (84) into the action integral (74) gives:

$$\int_{t_0}^{t_1} \left(\left(\frac{d}{dt}\delta X(t) + D(t)\delta X \right)^T M\dot{X}(t) + \delta q(t)^T F^*(t) \right) dt = 0 \tag{85}$$

Using integration by parts on the first term and the skew symmetry of the D matrix that gives$D^T = -D$

$$\int_{t_0}^{t_1} \left(\delta X^T(t) \left(\frac{d}{dt} \left(M\dot{X}(t) \right) + DM\dot{X}(t) \right) - \delta q(t)^T F^*(t) \right) dt = 0 \tag{86}$$

Then filling in for δX^T and $\dot{X}$ using definitions in 4.5.1

$$\int_{t_0}^{t_1} \delta q(t)^T \left(B^T MB\ddot{q} + B^T M\dot{B}\dot{q} + B^T DMB\dot{q} - F^*(t) \right) dt = 0 \tag{87}$$

with the definitions

- $M^*(t) \equiv B(t)^T M B(t) \longrightarrow$ Reduced mass matrix
- $N^*(t) \equiv B(t)^T \left(M\dot{B}(t) + D(t)MB(t) \right)$

the equation of motion for n-rigid bodies with both conservative and non-conservative forces and moments applied is extracted.

$$M^*(t)\ddot{q}(t) + N^*(t)\dot{q}(t) = F^* \tag{88}$$

4.7 The Hamilton Equation of Motion

The Hamilton equation of motion is derived following the procedure outlined in [2, ch8]. The Hamilton formulation seek to describe the motion in terms of first order differential equations. To do this the conjugate momenta is defined as (89).

$$p_i = \frac{\partial L(q_j, \dot{q}_j, t)}{\partial \dot{q}_i} \tag{89}$$

By applying this to the Lagrange equations below,

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0 \tag{90}$$

the time rate of change of the conjugate momenta is found to be (91)

$$\dot{p}_i = \frac{\partial L}{\partial q_i} \tag{91}$$

using (91) the differential of the Lagrange equation (90) can be written as

$$dL = \dot{p}_i dq_i + p_i d\dot{q}_i + \frac{\partial L}{\partial t} dt \tag{92}$$

by applying the Legendre transformation [2, ch8,p.335-336] on equation (92) the Hamilton (93) is found

$$H(q, p, t) = \dot{q}_i p_i - L(q, \dot{q}, t) \tag{93}$$

differentiating (93) yields

$$dH = \dot{q}_i dp_i - \dot{p}_i dq_i - \frac{\partial L}{\partial t} dt \tag{94}$$

By inverting the conjugate momenta equations (89), a change of variable matrix $Q(q, t)$ can be constructed such that:

$$\dot{q}_i = Q^{-1} p_i \tag{95}$$

By equating (94) with the differential of the Hamilton (96), and applying the change of variable matrix (95) to the resulting Hamiltonian the canonical equation of Hamilton can be extracted (97).

$$dH = \frac{\partial H}{\partial q_i} dq_i + \frac{\partial H}{\partial p_i} dp_i + \frac{\partial H}{\partial t} dt \tag{96}$$

$$\begin{aligned} \dot{q}_i &= \frac{\partial H}{\partial p_i} \\ -\dot{p}_i &= \frac{\partial H}{\partial q_i} \end{aligned} \tag{97}$$

5 Methods

5.1 Manual Evaluation of the Equations of Motion

Validating the resulting equation of motion that the algorithm produces is done by manually evaluating the same systems as the algorithm. Then by differentiating the resulting equations and controlling that the result is 0, it can be concluded that the resulting equation is correct. This is done for several systems, but only one system is shown explicitly in the thesis, the other test can be viewed in the appendix C.2. All matrix multiplication is done in Matlab by use of the symbolic toolbox.

Following system is the same as the constructed pendulum 7a and the one in figure 2.

5.1.1 Constructing the B matrix

Constructing the $[B(q,\dot{q})]$ matrix consists of finding the expressions for the angular velocity and tangential velocity of each centre of mass and stating them as a system of equations. Note that the tangential velocities must be stated in the inertial frame and the angular in the respective local frame (73).

Column

Starting with the velocities in the column, extracting the tangential velocity and angular velocity vector from the first velocity matrix in (41).

$$\dot{x}^{(1)} = R^{(1)}\widehat{\omega}^{(1)}r^{(1)} = -R^{(1)}\widehat{r}^{(1)}\omega^{(1)} = \begin{bmatrix} -R^{(1)}\widehat{r}^{(1)}\begin{pmatrix}0\\0\\1\end{pmatrix} & \begin{pmatrix}0\\0\\0\end{pmatrix} \end{bmatrix} \begin{bmatrix}\dot{\theta}^{(1)}\\ \dot{\theta}^{(2)}\end{bmatrix} \tag{98}$$

continuing with the angular velocity:

$$\omega^{(1)} = \begin{bmatrix}0\\0\\\dot{\theta}^{(1)}\end{bmatrix} = \begin{bmatrix}0 & 0\\0 & 0\\1 & 0\end{bmatrix} \begin{bmatrix}\dot{\theta}^{(1)}\\ \dot{\theta}^{(2)}\end{bmatrix} \tag{99}$$

Arm

Extracting the tangential and angular velocity from the second velocity matrix (43). Starting with the tangential velocity:

$$
\begin{aligned}
&\dot{x}^{(2)} = R^{(1)}\widehat{\omega}^{(1)}r^{(2)} + R^{(1)}R^{(2/1)}\widehat{\omega}^{(2/1)}r^{(2/P)} = \\
&- R^{(1)}\left(\widehat{r}^{(2)}\omega^{(1)} + R^{(2/1)}\widehat{r}^{(2/P)}\omega^{(2/1)}\right) = \\
&\begin{bmatrix} -R^{(1)}\widehat{r}^{(2)}\begin{pmatrix}0\\0\\1\end{pmatrix} & -R^{(1)}R^{(2/1)}\widehat{r}^{(2/P)}\begin{pmatrix}0\\1\\0\end{pmatrix} \end{bmatrix}\begin{bmatrix}\dot{\theta}^{(1)}\\ \dot{\theta}^{(2)}\end{bmatrix}
\end{aligned} \tag{100}
$$

Continuing with the angular velocity

(101)

$$
\omega^{(2)}(t) = \begin{bmatrix} -\dot{\theta}^{(1)}sin(\theta^{(2)}) \\ \dot{\theta}^{(2)} \\ \dot{\theta}^{(1)}cos(\theta^{(2)}) \end{bmatrix} = {R^{(2/1)}}^T\omega^{(1)} + \omega^{(2/1)} = \begin{bmatrix} {R^{(2/1)}}^T\begin{pmatrix}0\\0\\1\end{pmatrix} & \begin{pmatrix}0\\1\\0\end{pmatrix} \end{bmatrix}\begin{bmatrix}\dot{\theta}^{(1)}\\ \dot{\theta}^{(2)}\end{bmatrix}
$$

B matrix

Thus by collecting all components in one matrix the relation $[\dot{X}(t)] = [B(t)]\{\dot{q}\}$ is found:

$$
\begin{bmatrix}\dot{x}^{(1)}\\ \omega^{(1)}\\ \dot{x}^{(2)}\\ \omega^{(2)}\end{bmatrix} = \begin{bmatrix} -R^{(1)}\widehat{r}^{(1)}\begin{pmatrix}0\\0\\1\end{pmatrix} & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ -R^{(1)}\widehat{r}^{(2)}\begin{pmatrix}0\\0\\1\end{pmatrix} & -R^{(1)}R^{(2/1)}\widehat{r}^{(2/P)}\begin{pmatrix}0\\1\\0\end{pmatrix} \\ {R^{(2/1)}}^T\begin{pmatrix}0\\0\\1\end{pmatrix} & 0 \\ & 1 \\ & 0 \end{bmatrix}\begin{bmatrix}\dot{\theta}^{(1)}\\ \dot{\theta}^{(2)}\end{bmatrix} \tag{102}
$$

5.1.2 $\dot{\mathbf{B}}(\mathbf{t})$ matrix

Differentiating the $[B(t)]$ matrix with respect to time is shown in (73) to be the time derivative of the components, thus the $[\dot{B}(t)]$ becomes:

$$[\dot{B}(t)] = \begin{bmatrix} -\dot{R}^{(1)}\widehat{r}^{(1)} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ -\left[\dot{R}^{(1)}\widehat{r}^{(2)} - R^{(1)}\dot{R}^{\widehat{(2/1)}r^{(2/P)}}\right] \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} & -\dot{R}^{(2)}\widehat{r}^{(2/P)} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \\ \dot{R}^{(2/1)^T} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \end{bmatrix} \tag{103}$$

5.1.3 Mass matrix

$$M = \begin{bmatrix} m^{(1)} \underset{3\times3}{I} & \underset{3\times3}{0} & \underset{3\times3}{0} & \underset{3\times3}{0} \\ \underset{3\times3}{0} & J^{(1)} & \underset{3\times3}{0} & \underset{3\times3}{0} \\ \underset{3\times3}{0} & \underset{3\times3}{0} & m^{(2)} \underset{3\times3}{I} & \underset{3\times3}{0} \\ \underset{3\times3}{0} & \underset{3\times3}{0} & \underset{3\times3}{0} & J^{(2)} \end{bmatrix} \tag{104}$$

5.1.4 D(t) matrix

$$D(t) = \begin{bmatrix} \underset{3\times3}{0} & \underset{3\times3}{0} & \underset{3\times3}{0} & \underset{3\times3}{0} \\ \underset{3\times3}{0} & \widehat{\omega}^{(1)}(t) & \underset{3\times3}{0} & \underset{3\times3}{0} \\ \underset{3\times3}{0} & \underset{3\times3}{0} & \underset{3\times3}{0} & \underset{3\times3}{0} \\ \underset{3\times3}{0} & \underset{3\times3}{0} & \underset{3\times3}{0} & \widehat{\omega}^{(2)}(t) \end{bmatrix} \tag{105}$$

5.1.5 Force Vector

The force vector holds the external forces, for example gravity and driving moments.

$$F(t) = \begin{bmatrix} 0 \\ 0 \\ -gm^{(1)} \\ 0 \\ 0 \\ F^{(1)}(t) \\ 0 \\ 0 \\ -gm^{(2)} \\ 0 \\ F^{(2)}(t) \\ 0 \end{bmatrix} \tag{106}$$

with $F^{(1)}(t)$ and $F^{(2)}(t)$ being applied moments on the bodies.

5.1.6 Equation of motion for the Pendulum

Using the definitions 4.6.3 and (83) to construct $M^*(t)$ and $N^*(t)$, the equations of motion can be written as:

$$\ddot{q} = M^*(t)^{-1} \left(F^*(t) - N^*(t)\dot{q}\right) \tag{107}$$

By evaluating this equation in Matlab it can be differentiated with the output from the algorithm, if the result is 0 then the equations from the algorithm can be said to be correct.

$$\ddot{q}_{Manual} - \ddot{q}_{Algorithm} = 0 \tag{108}$$

5.2 Canonical Equations of Motion

To construct the canonical equations of Hamilton the Lagrangian $[L(q, \dot{q}, t) = K(q, \dot{q}) - V(q)]$ of the system must be found. In this section of the paper the equations are differentiated with respect to the generalized coordinates. The kinetic energy of the system is the sum of the energy of the tangential velocities due to rotation and the angular velocities of the centre of masses

of the bodies. To facilitate the change of variable between the generalized essential velocity and the conjugate momenta (95), the generalized form of equation (75) for a system of rigid n bodies is used (109).

$$K(t) = \frac{1}{2}\dot{X}(q,\dot{q})^T M \dot{X}(q,\dot{q}) = \frac{1}{2}(B(q)\dot{q})^T M B(q)\dot{q} \tag{109}$$

Consequently the main work in setting up the equation for the kinetic energy is the construction of the $[B(t)]$ matrix 4.5.1. It is constructed in the previous section (102), from the velocity vectors of the bodies that where found in 4.3.4. Note that the specific matrix multiplication and derivation is done in Matlab C.5.9.

Potential energy

Before formulating the Lagrangian and subsequently the Hamiltonian, the expression for the energy due to conservative forces must be found. Potential energy is formulating by the height of the centre of masses above the lowest point in the inertial third direction. The first centre of mass does not move in this direction and are therefore not a part of this equation. As such the potential energy of the system is given as:

$$V(\theta^{(2)}) = \left(r^{(2/P)} - \sin(\theta^{(2)})r^{(2/P)}\right) m^{(2)} g \tag{110}$$

5.2.1 The Hamiltonian

The Lagrangian is now the difference in the sum of the kinetic energies and potential energy, with the $[M]$ matrix being identical to (104).

(111)

$$L(\theta^{(1)},\theta^{(2)},\dot{\theta^{(1)}},\dot{\theta^{(2)}}) = \frac{1}{2}\left(B(\theta^{(1)},\theta^{(2)})\begin{bmatrix}\dot{\theta}^{(1)}\\ \dot{\theta}^{(2)}\end{bmatrix}\right)^T M \left(B(\theta^{(1)},\theta^{(2)})\begin{bmatrix}\dot{\theta}^{(1)}\\ \dot{\theta}^{(2)}\end{bmatrix}\right) - V(\theta^{(2)})$$

Following the steps given in [2, ch8,p 338] to formulate the Hamiltonian from the Lagrangian the conjugate momenta is defined as

$$\begin{aligned} p^{(1)} &= \frac{\partial L(\theta^{(1)},\theta^{(2)},\dot{\theta^{(1)}},\dot{\theta^{(2)}})}{\partial \dot{\theta}^{(1)}} \\ p^{(2)} &= \frac{\partial L(\theta^{(1)},\theta^{(2)},\dot{\theta^{(1)}},\dot{\theta^{(2)}})}{\partial \dot{\theta}^{(2)}} \end{aligned} \tag{112}$$

Using (111) and (112) the Hamiltonian is formulated:

$$H = \begin{bmatrix}\dot{\theta}^{(1)}\\ \dot{\theta}^{(2)}\end{bmatrix} \cdot \begin{bmatrix}p^{(1)}\\ p^{(2)}\end{bmatrix} - L(\theta^{(1)},\theta^{(2)},\dot{\theta^{(1)}},\dot{\theta^{(2)}}) \tag{113}$$

Continuing the expressions for the canonical momentas are evaluated and written as a system of equations:

$$\begin{bmatrix} p^{(1)} \\ p^{(2)} \end{bmatrix} = Q(\theta^{(1)}, \theta^{(2)}) \begin{bmatrix} \dot{\theta}^{(1)} \\ \dot{\theta}^{(2)} \end{bmatrix} \tag{114}$$

By inverting (114) an expression for the angular velocities are found as a function of the angles and the canonical momentas.

$$\begin{bmatrix} \dot{\theta}^{(1)} \\ \dot{\theta}^{(2)} \end{bmatrix} = \begin{bmatrix} p^{(1)} \\ p^{(2)} \end{bmatrix} Q(\theta^{(1)}, \theta^{(2)})^{-1} \tag{115}$$

Applying (115) to the Hamiltonian (113) eliminates the $\dot{\theta}$'s from the expression and makes it a function solely of the angles and canonical momentas.

$$H = \begin{bmatrix} p^{(1)} \\ p^{(2)} \end{bmatrix} Q(\theta^{(1)}, \theta^{(2)})^{-1} \cdot \begin{bmatrix} p^{(1)} \\ p^{(2)} \end{bmatrix} - L(\theta^{(1)}, \theta^{(2)}, p^{(1)}, p^{(2)}) \tag{116}$$

Using the Hamiltonian (116) the canonical equations of motion (97) are found.

$$\begin{aligned} \dot{\theta}^{(1)} &= \frac{\partial H(\theta^{(1)}, \theta^{(2)}, p^{(1)}, p^{(2)})}{\partial p^{(1)}} \\ \dot{\theta}^{(2)} &= \frac{\partial H(\theta^{(1)}, \theta^{(2)}, p^{(1)}, p^{(2)})}{\partial p^{(2)}} \\ \dot{p}^{(1)} &= -\frac{\partial H(\theta^{(1)}, \theta^{(2)}, p^{(1)}, p^{(2)})}{\partial \theta^{(1)}} \\ \dot{p}^{(2)} &= -\frac{\partial H(\theta^{(1)}, \theta^{(2)}, p^{(1)}, p^{(2)})}{\partial \theta^{(2)}} \end{aligned} \tag{117}$$

The above equations are evaluated in Matlab and written to a function file to facilitate the use of the midpoint rule [4, ch1] for solving the differential equations.

5.3 Design and Construction

The model is split up into two main parts, the frame and the pendulum. The pendulum is the dynamic system that the calculation will be applied to, and the frame is the framework that hold the pendulum safely in place as it moves.

5.3.1 Frame

When the pendulum moves, it generates substantial dynamic reaction forces. Thus, a strong, wide and rigid frame is needed to hold it in place. A hexagonal shape was chosen for the frame 3a, as it would provide the needed stability and strength. Furthermore, the hexagon will provide a natural boundary in which the pendulum can move safely.

For the frame 18 m of square 15 mm wide and 1.5 mm thick stainless steel was used. The bars were cut to create the frame, 6 units of 500 mm length for the height, and 12 units of 600 mm length cut with 30-degree angle at the ends for the hexagon at the top and bottom. For the supports at the top and bottom 6 units of 400 mm length was used, as well as 2 hexagonal plates with 300 mm radius and 2 mm thickness was cut with a plasma cutter.

The frame was welded together by the author using a mig welding apparatus with compact tread. An angle grinder was used to smooth any sharp edges.

5.3.2 Pendulum

The two parts of the pendulum, the column and arm 5a is made by using steel pipes and rods. The column is a 25×2×500mm steel pipe, with 4 10mm steel rods extended out in the radial direction to hold the arm support. The support for the arm is composed of an axle and two cylindrical supports to hold it. The axle is connected to the arm with two radial ball bearings. The column is connected to the frame with 3 bearings. One thrust ball bearing located at the bottom, to hold the weight of the system as it rotates. At the top and bottom there are radial ball bearings for stability 4.

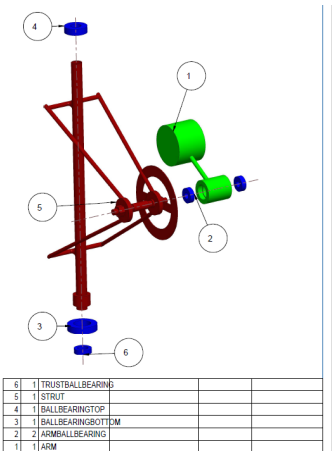
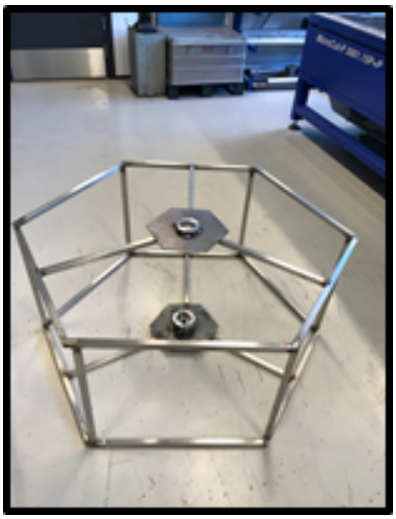


Figure 4: Pendulum with bearings as blue objects, this is an outcrop of figure 29

(a) The Frame for the pendulum



(b) The Frame for the pendulum, with bearing supports

Figure 3: The Frame before assembly with the Strut and Arm

The arm is made up of three parts. A massive 70mm long rod with radius of 95mm that hold most of the mass in the arm, a 312mm long rod with radius of 10mm and a cylindrical components that house the bearings and axle, that connects the arm to the column.

Most of the parts where produced at HVL using a Computer Numerical Controled(CNC) turning machine [3, ch7.3], after the author had designed and created the necessary part drawings. The axle and main mass of the pendulum was produced by the author in a semi-manual turning machine. The pendulum was assembled when all parts where done with the assistance of the bachelor team at HVL.

(a) The arm

(b) The column, with the arm attached

Figure 5: Arm and Column assembly

5.4 Measurements

Three separate behaviours of the pendulum is measured. Two to construct the friction functions, and one main measurement to compare with the end result. First the RPM of the column with the arm fixed, to be used to model the friction in the Column. Second measurement is of the arm with the column fixed, to be used to model the friction in the Arm, and last with both bodies free to move.

5.4.1 Column

Before the measurement is taken the position of the arm is locked in place relative to the column. The column is then accelerated to approximately 100RPM. It is then allowed to spin freely and decelerate at a natural pace while the angular velocity is measured constantly using a RPM measuring tool 6. The RPM measurement tool is filmed using a mobile device with a camera. The RPM and time of each measurement are read of the films after completion. This was repeated 4 times.

(a) Overview of the measurements

(b) Measuring the angle

Figure 7: Measuring the angle of the Arm and overview

5.4.2 Arm

The arm is given an initial angle and released with the column kept at a fixed position 7. A mobile device with a camera filmed the movement and the angle of each of the top point and time of each swing was measured until it came to rest. This was repeated 4 times.

Figure 6: Measuring the RPM of the Column.

5.4.3 Main Measurement

Measuring the pendulum as it was moving, was accomplished by three cameras. One where filming the Arm 7b, number two was filming the column from above with a protractor and an indicator 6 and the third was giving an overview to match the timing of the two other measurements 7a. Reading the measurements of the films where done by opening the video files in Microsoft picture, choosing the function 'save pictures' that creates a series of pictures of a film and the stepping trough the films frame by frame. To get the timing

of the measurement of the RPM and the angle correctly, the overview film was used. The swing of the pendulum where observed and the time point for each end of the swings where noted. By then matching the point in time where the Arm drops for the first time, with the video of the Column the angular velocity measurements where timed correctly.

Angle of the Arm

The angle of the arm is measured using a protractor and a camera, both fitted to the strut side of the Pendulum 7a. The angle and time is measured for each endpoint of a swing for the arm.

Angular velocity of the column

Calculating the angular velocity of the column became necessary due to the fact that the RPM measuring tool only updates its measurement each 0.7 second and thus is too slow to pick up the rapid change in angular velocity that the pendulum is subject to. Therefore, using the film of the column and time points from the overview film, the change in angle and the time used was measured from slightly before and after each time point. Then by dividing the change in angle with the time, the angular velocity of the column where calculated for each swing of the Arm.

5.5 Energy Calculation

Computing the energy of the system is done by summing the kinetic and potential energy. The kinetic energy is calculated using the following equation:

$$K(t) = \frac{1}{2}\dot{X}(t)^T M \dot{X}(t) \tag{118}$$

where

$$\dot{X} = B(t)\dot{q} \tag{119}$$

$\dot{X}$ is given by the function velocities.mat C.7.2. The equations are created by the function MFMNumEquMaker C.4.1 and saved to a Matlab file by the function VelocitySaver C.5.1. Potential energy is calculated for each body by the equation, using the positions of the centre of mass in the inertial 3 direction.

$$V = \sum_{i=1}^{\alpha} \left(r^{(i)}(t) e_3^I - \min |r^{(i)}(t) e_3^I| \right) m^{(i)} g \tag{120}$$

Where $r^{(i)}(t)e_3^I$ is the position of the i'th centre of mass in the third inertial direction, $\min|r^{(i)}(t)e_3^I|$ is the lowest point the i'th centre of mass can reach in the third inertial direction and $m^{(i)}$ is the mass of the i'th body. Using equation (118) and (120) the function EAnalysis C.7.1 calculates the energy for each time step defined by the solution matrix from integration. Potential energy for the Arm, when the column was held in place was calculated straight forward:

$$V_{(arm)} = (r^{(2/p)} - sin(\theta^{(2)})r^{(2/p)})m^{(2)}g \tag{121}$$

By calculating the initial energy using the initial conditions the energy error can be calculated by:

$$E_{err} = E_{tot} - E_{init} \tag{122}$$

5.6 Friction Models

5.7 Dissipative functions

To account for friction in the bearings and air resistance, two dissipation functions is needed. One for the column and one for the arm. The functions is assumed to be on the form of Rayleigh's dissipation functions and such being polynomials of order 2 in the essential generalized velocities $\mathcal{F}(\dot{q}^{(i)})_i = O(\dot{q}^{(i)^2})$. Thus the friction force functions become:

$$F(\dot{q}^{(i)})_i = \frac{d\mathcal{F}_i}{d\dot{q}^{(i)}} = O(\dot{q}^{(i)}) \tag{123}$$

Both models is added to the F vector, as moments that act in the opposite direction of the local angular velocity for each body.

5.7.1 Friction in the Arm

Using the measured angle, the energy and energy change between each oscillation is calculated using (121). Furthermore, by dividing the difference in angle for each oscillation and the time used, the average angular velocity of each swing of the Arm is calculated. The change in energy is then plotted against the average angular velocity of each oscillation 13b. By observing the resulting plot, there is an indication that the dissipative function might

behave as a first order polynomial. Thus two dissipative functions is tested using the data and the Matlab function lsqnonlin.

$$\mathcal{F}(\dot{\theta}^{(2)})_{arm_1} = -C_1 {\dot{\theta}^{(2)}}^2 \tag{124}$$

$$\mathcal{F}(\dot{\theta}^{(2)})_{arm_2} = -C_2 |\dot{\theta}^{(2)}| \tag{125}$$

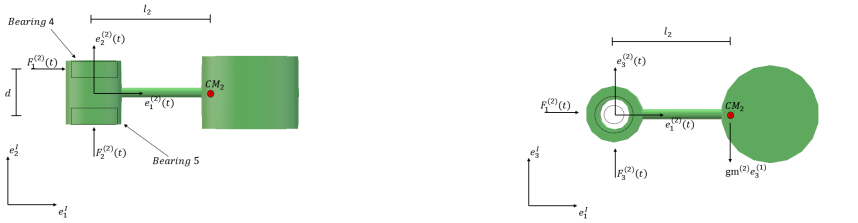
Where C_i are constants and the absolute value of $\dot{\theta}^{(2)}$ in (125) is due to ensure that the energy change due to friction is always negative, no mater the direction of the angular velocity of the arm. The resulting dissipation is plotted in 14, and resulting behaviour due to the two functions in 16 and 15. By observing the resulting behaviour, it is clear that although the second order function fits to begin with but does not stop. The first order function stops, but does not follow the curve at the start. By manually testing for different values of C_1 and C_2, it is established that the friction function behaves as a polynomial at high values in the angle, and more as a constant at lower angles. As such a third friction function was tested:

$$\frac{d\mathcal{F}_{Arm3}}{d\dot{\theta}^{(2)}} = F(\dot{\theta}^{(2)})_{Arm3} = \left(C_3 {\dot{\theta}^{(2)}}^2 + C_4\right) n(\dot{\theta}^{(2)}) \tag{126}$$

where $n(\dot{\theta}^{(2)})$ is a directional vector pointing in the opposite direction of the angular velocity. Resulting plots 17, and adjusting C_4 to make the pendulum stop within the time frame 18. The form of the friction force function (126) is easily recognized as the reaction force of a pendulum where the angular acceleration has been neglected. Therefore a fourth test is performed, where the friction force is the reaction force at the bearings times a constant and a directional vector. Both the fourth and third tests are sent to an optimizing function and the results plotted 21 19, the argument for also testing the third being that it would be interesting to see if a simplified friction function could be used. As the reaction force function would grow to immense sizes as the amount of bodies in a system is increased.

Reaction Force in the Arm

Modelling the friction with the reaction forces times a friction constant follows the philosophy outlined in [3, ch2.1 p.44], note that as these are bearings the resulting direction of the reaction force is of no consequence. To evaluate the equations for the reaction force on the Arm bearings, both forces and



(a) Arm, seen from above

(b) Arm, seen from the front

Figure 8: The Arm with vector basis and reaction forces

moments on the centre of mass must be found. This is done by formulation Newtons equation and Eulers equation [1, ch7]. Starting with Newtons equation for the forces using equation (27).

$$\sum F_{c_2}^{(2)}(t) = L\dot{(t)} = m^{(2)} a_{cm_2}(t) \tag{127}$$

Continuing with Euler's equation for the moments, using the equation for the angular velocity of the Arm (23).

$$\sum M_c = \dot{H}^{(2)}(t) = e^{(1)}(t) R^{(2/1)} \left(\widehat{\omega}^{(2)}(t) J^{(2)} \omega^{(2)}(t) + J^{(2)} \dot{\omega}^{(2)}(t) \right) \tag{128}$$

By taking the sum of the forces and evaluating for the reaction forces in bearing 4:

$$F_{R_4}^{(2)}(t) = F_c^{(2)}/2 + \begin{bmatrix} 0 \\ 0 \\ m^{(2)} g/2 \end{bmatrix} + \begin{bmatrix} -\dot{H}_3^{(2)}/d \\ 0 \\ \dot{H}_1^{(2)}/d \end{bmatrix} + \begin{bmatrix} F_{c_2}^{(2)} l_2/d \\ 0 \\ 0 \end{bmatrix} \tag{129}$$

and in bearing 5:

$$F_{R_5}^{(2)}(t) = F_c^{(2)}/2 + \begin{bmatrix} 0 \\ 0 \\ m^{(2)} g/2 \end{bmatrix} + \begin{bmatrix} \dot{H}_3^{(2)}/d \\ 0 \\ -\dot{H}_1^{(2)}/d \end{bmatrix} - \begin{bmatrix} F_{c_2}^{(2)} l_2/d \\ 0 \\ 0 \end{bmatrix} \tag{130}$$

By taking the norm of the reaction forces, summing up and multiplying with a friction constant and a directional vector the friction function is constructed.

$$F_{arm_4} = C_{10} \left(\| F_{R_4}^{(2)}(t) \| + \| F_{R_5}^{(2)}(t) \| \right) n(\dot{\theta}^{(2)}) \tag{131}$$

5.7.2 Friction in the Column

It is assumed that the friction in the column behaves as the friction in the Arm, thus two tests are performed. One with the full reaction force on the bearings and one simplified version (132) 23 24. There are three bearings on the column, bearing 1 and 2 takes forces in the xy plane, and bearing 3 takes forces in the third inertial direction.

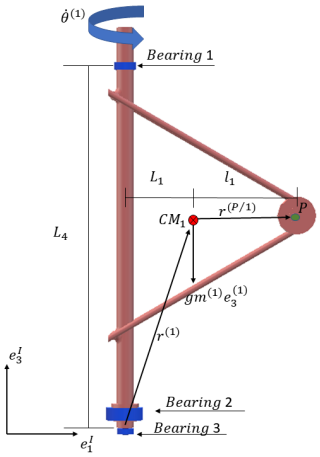


Figure 9: Column with bearings as blue objects.

(132)
$$F_{base_{simp}} = \left(C_5 \dot{\theta^{(1)}}^2 + C_6 + C_7 \left(L_1 + l_1 + cos(\theta^{(2)}) l_2\right) \dot{\theta^{(2)}}^2\right) n(\dot{\theta}^{(1)})$$

Reaction Force in the Column

Forces due to acceleration on the first centre of mass:

(133)
$$F_c^{(1)} = m^{(1)} e^{(1)}(t) \left(\widehat{\omega}^{(1)}(t)\widehat{\omega}^{(1)}(t) r^{(1)} + \dot{\widehat{\omega}}^{(1)}(t) r^{(1)}\right)$$

Moments due to the rotation of the first centre of mass

(134)
$$\sum M_c^{(1)} = \dot{H}^{(1)} = e^{(1)}(t \left(\widehat{\omega}^{(1)}(t) J^{(1)} \omega^{(1)}(t) + J^{(1)} \dot{\omega}^{(1)}(t)\right))$$

Bearing 1

(135)
$$F_{R_1}^{(1)}(t) = F_{c_1}/2 + F_{c_2}/2 + \begin{bmatrix} \dot{H}_2^{(1)}/L_4 \\ -\dot{H}_1^{(1)}/L_4 \\ 0 \end{bmatrix} + \begin{bmatrix} m^{(1)} g L_1/L_4 \\ m^{(1)} g L_2/L_4 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -\dot{H}^{(1)}/L_4 \\ 0 \end{bmatrix} + \begin{bmatrix} \left(F_{c_3}^{(2)} + m^{(2)} g\right) L_1/L_4 \\ 0 \\ 0 \end{bmatrix}$$

Bearing 2

$$F_{R_2}^{(1)}(t) = F_{c_1}/2 + F_{c_2}/2 + \begin{bmatrix} -\dot{H}_2^{(1)}/L_4 \\ \dot{H}_1^{(1)}/L_4 \\ 0 \end{bmatrix} + \begin{bmatrix} -m^{(1)}gL_1/L_4 \\ -m^{(1)}gL_2/L_4 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \dot{H}^{(1)}/L_4 \\ 0 \end{bmatrix} + \begin{bmatrix} -\left(F_{c_3}^{(2)} + m^{(2)}g\right) L_1/L_4 \\ 0 \\ 0 \end{bmatrix} \tag{136}$$

Bearing 3

$$F_{R_3}^{(1)}(t) = \left(m^{(1)} + m^{(2)}\right) g + F_{c_3}^{(2)} \tag{137}$$

By taking the norm of all the reaction forces and multiplying with a friction constant and a directional vector, the friction function for the column is constructed. Note that Bearing 3 has a different constant that 1 and 2, as it is a different bearing than 1 and 2.

$$F_{Base} = \left(\left(\|F_{R_1}^{(1)}(t)\| + \|F_{R_2}^{(1)}(t)\|\right) C_8 + C_9 \|F_{R_3}^{(1)}(t)\|\right) n(\dot{\theta}^{(1)}) \tag{138}$$

Both friction functions where optimized by adjusting the constants in Matlab using lsqnonlin and ode45.

5.8 Testing the Algorithm

Testing the algorithm is done by running it several times, with increasing amounts of bodies. The system tested is a simplified chain. With each link being one body and modelled as a hollow cylinder. The first body is free to rotate about the third inertial axis, the second around the local second axis, the third body rotates around the local third axis and so on. Mass is calculated using the density of steel and the volume of the models. The algorithm uses the function simplify to shorten the equations, this function works well when the equations are sufficiently short. But as the system grows it will max-out the memory on the computer and fail. Thus the measurements where done twice, one with simplify and one without. The following measurements where done on the performance of the algorithm:

- The time the algorithm uses to construct the equations of motion
- The size of the resulting func file

- How long it takes to run the func file for one time step

Time usage is measured using the tic/toc function in Matlab. Testing script can be viewed in appendix C.1.

6 Results

6.1 Algorithm for the Principle of Virtual Work

Before starting to construct the algorithm, it is important to note that it will only be able to handle movement due to rotations, as such the relative position vectors cannot change. Further more, there will be no other restrictions on the movement other that those naturally occurring due to applied forces and moments. If the algorithm is to be used for modelling impacts or systems with other geographical restrictions, such as an object hitting the ground or the piston in an engine, the restriction must be formulated as an applied force and cannot be enforced in the equation of motion itself.

6.1.1 B matrix

By analysing the equation of motion (87) it is evident that the main component is the $[B(t)]$ matrix. Of all the components of the equation it is the most complex and hard to compute. The matrix itself can be divided into two main categories, the angular velocity and tangential. Finding a pattern in how these velocities expand as more bodies are added, is integral to the construction of the algorithm.

Angular velocity

Using the simplified form of the Adjoint operation 18, and keeping in mind that the angular velocity must be stated in the local vector basis, the form of the α velocity is:

(139)

$$\dot{e}^{(\alpha)} = e^{(\alpha)}\omega^{(\alpha)} = e^{(\alpha)}(R^{(\alpha/1)^T}\omega^{(1)} + R^{(\alpha/2)^T}\omega^{(2/1)} + \cdots + R^{(\alpha-1/\alpha-2)^T}\omega^{(\alpha-1/\alpha-2)} + \omega^{(\alpha/\alpha-1)})$$

By using the above equation and expanding one body at the time:

(140)

$$\begin{aligned}
\dot{e}^{(1)} &= e^{(1)}\omega^{(1)} \\
\dot{e}^{(2)} &= e^{(2)}\omega^{(2)} = e^{(2)}(R^{(2/1)^T}\omega^{(1)} + \omega^{(2/1)}) \\
\dot{e}^{(3)} &= e^{(3)}\omega^{(3)} = e^{(3)}(R^{(3/2)^T}R^{(2/1)^T}\omega^{(1)} + R^{(3/2)^T}\omega^{(2/1)} + \omega^{(3/2)})
\end{aligned}$$

As can be seen, the angular velocity takes the previous input to the $[B(t)]$ matrix and multiplies it with the local rotational matrix transposed. Treating

the local angular velocity vectors as a special case and inputting them first, the algorithm for the angular velocity input to the $[B(t)]$ matrix becomes:

$$B_{i,i} = \omega^{(i/i-1)}/\dot{\theta}^{(i)} \tag{141}$$

and

$$B_{i,j} = R^{(i/i-1)^T} B_{i-1,j} \tag{142}$$

where i in (142) goes from $2 \rightarrow \alpha$ and j from $1 \rightarrow i$

Tangential velocity

Starting with the local velocity matrix, which coincidently also is the first input of the tangential velocities, it is shown that the local tangential velocity is always the last angular velocity matrix times position vector $r^{(\alpha/p_r^{(\alpha)})}$ spanning from the last point of rotation to the last centre of mass:

$$\Omega^{(\alpha/\alpha-1)}(t) = E^{(\alpha/\alpha-1)^{-1}} \dot{E}^{(\alpha/\alpha-1)} = \begin{bmatrix} \widehat{\omega}^{(\alpha/\alpha-1)} & \widehat{\omega}^{(\alpha/\alpha-1)} r^{(\alpha/p_r^{(\alpha)})} \\ 0^T & 0 \end{bmatrix} \tag{143}$$

Since the tangential velocity must be given in the inertial vector basis, the $B(i,i)$ input to the matrix becomes:

$$B_{i,i} = R^{(i)} \widehat{\omega}^{(i/i-1)} r^{(i/p_r^{(i)})}/\dot{\theta}^{(i)} \tag{144}$$

Continuing, to find the other inputs the Adjoint operation of the Special Euclidean group is consulted. Note that the position vectors in the connection matrices contain a rotation, it is not of importance now but it will be when constructing the time derivative of $[B(t)]$.

(145)

$$Ad_{E^{(\alpha/\alpha-1)}}(\Omega^{(\alpha-1)}) = E^{(\alpha/\alpha-1)^{-1}} \Omega^{(\alpha-1)} E^{(\alpha/\alpha-1)} =$$
$$\begin{bmatrix} R^{(\alpha/\alpha-1)^T} & -R^{(\alpha/\alpha-1)^T} r^{(\alpha/\alpha-1)} \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} \widehat{\omega}^{(\alpha-1)} & R^{(\alpha-1)^T} \dot{x}^{(\alpha-1)} \\ 0^T & 0 \end{bmatrix} \begin{bmatrix} R^{(\alpha/\alpha-1)} & r^{(\alpha/\alpha-1)} \\ 0^T & 1 \end{bmatrix}$$

Extracting the tangential velocity component and relating it to the inertial.

$$\dot{x}^{(\alpha)} = \dot{x}^{(\alpha-1)} + R^{(\alpha-1)} \widehat{\omega}^{(\alpha-1)} r^{(\alpha/\alpha-1)} \tag{146}$$

Expanding the last term:

$$\begin{aligned} &R^{(\alpha-1)} \left(R^{(\alpha/\alpha-1)^T} \widehat{\omega}^{(\alpha-1)} R^{(\alpha/\alpha-1)} + \widehat{\omega}^{(\alpha/\alpha-1)} \right) r^{(\alpha/\alpha-1)} = \\ &R^{(\alpha-1)} \widehat{\omega}^{(\alpha-1)} R^{(\alpha/\alpha-1)} r^{(\alpha/\alpha-1)} + R^{(\alpha)} \widehat{\omega}^{(\alpha/\alpha-1)} r^{(\alpha/\alpha-1)} \end{aligned} \tag{147}$$

Thus the algorithm for the tangential velocities first for $i = [2, 3, ..., \alpha]$ and $j = [1, 2, ..., i-2]$, with the input j=i-1 and j=i treated as special cases.

$$B_{i,j} = B_{i-1,j} + R^{(j)}\widehat{\omega}^{(j/j-1)}R^{(i/j)}r^{(i/i-1)}/\dot{\theta}^{(j)} \tag{148}$$

and for $j = i - 1$

$$B_{i,j} = B_{i-1,j} + R^{(i)}\widehat{\omega}^{(i/j)}r^{(i/j)}/\dot{\theta}^{(i)} \tag{149}$$

The algorithm is implemented in Matlab and can be viewed in the appendix C.4.3.

6.1.2 Time derivative of B

The angular accelerations

The angular acceleration in vector form is:

$$\ddot{e}^{(\alpha)} = \dot{e}^{(\alpha)}\omega^{(\alpha)} + e^{(\alpha)}\dot{\omega}^{(\alpha)} = e^{(\alpha)}\widehat{\omega}^{(\alpha)}\omega^{(\alpha)} + e^{(\alpha)}\dot{\omega}^{(\alpha)} \tag{150}$$

where $e^{(\alpha)}\widehat{\omega}^{(\alpha)}\omega^{(\alpha)} = 0$ since this term is just the cross product of two parallel vectors. This shows that differentiating the angular velocity part of the $[B(t)]$ matrix is just the time derivative of the components.

$$\ddot{e}^{(\alpha)} = e^{(\alpha)}\dot{\omega}^{(\alpha)} \tag{151}$$

It is important to note that the ω's that are in the $[B(t)]$ matrix are all just vectors with two zero components and one 1. Therefore the (i, i) input to the $\dot{B}$ will always be zero, as this input is strictly the local angular velocity vector $\omega^{(i/i-1)}$ (141). Further more, this simplifies the rest of the time derivatives, as now the derivative of the previous input is known. Differentiating (142) with respect to time gives:

$$\dot{B}_{i,j} = \dot{R}^{(i/i-1)^T}B_{i-1,j} + R^{(i/i-1)^T}\dot{B}_{i-1,j} \tag{152}$$

The tangential accelerations

All the linear velocities in $[B(t)]$ is stated relative to the inertial frame, therfore the time derivative simplifies to the components of B. Starting by evaluating the special case giving the (i, i) input (144). With both $\widehat{\omega}^{(i/i-1)}$ and $r^{(i/p_r^{(i)})}$ being constants, the time derivative becomes:

$$\dot{B}_{i,i} = \dot{R}^{(i)}\widehat{\omega}^{(i/i-1)}r^{(i/p_r^{(i)})} \tag{153}$$

This takes care of the first input to the $\dot{B}$ matrix, and simplifies the next step as the previous time derivative of the input to the B is always known. Continuing with the special case for $(i, i-1)$ and remembering that $r^{(i/i-1)} = r^{(p_r^{(i)}/i-1)} + R^{(i/i-1)} r^{(i/p_r^{(i)})}$

$$\dot{B}_{i,j} = \dot{B}_{i-1,j} + \dot{R}^{(i)} \widehat{\omega}^{(i/j)} r^{(i/j)} + R^{(i)} \widehat{\omega}^{(i/j)} \dot{R}^{(i/i-1)} r^{(i/p_r^{(i)})} \tag{154}$$

and the general input:

$$\begin{aligned} \dot{B}_{i,j} =& \dot{B}_{i-1,j} + \dot{R}^{(j)} \widehat{\omega}^{(j/j-1)} R^{(i/j)} r^{(i/i-1)} + R^{(j)} \widehat{\omega}^{(j/j-1)} \dot{R}^{(i/j)} r^{(i/i-1)} + \\ & R^{(j)} \widehat{\omega}^{(j/j-1)} R^{(i/j)} \dot{R}^{(i/i-1)} r^{(i/p_r^{(i)})} \end{aligned} \tag{155}$$

The algorithm is implemented in Matlab and can be viewed in the appendix C.4.3.

6.1.3 The Algorithm

To construct the equations of motion, the algorithm takes the physical size of the system as input. Using these, it constructs the necessary matrices and vectors, multiply them together and writes the equations to a separate file called func.mat. This function can then again be used by any generic integration function to solve the equations of motion. The algorithm is called MFMNumEquMaker. Below is a quick summary of how the function MFMNumEquMaker constructs the equations of motion. The functions are written in Matlab and can be viewed in the appendix C.

Input

Example of how the input must look like and how the function MFMNumEquMaker is called can be viewed in the appendix C.3.1

PreVList List of vectors pointing from the last centre of mass to the next point of rotation

Axis Axis of rotation for each body

PostVList List of vectors pointing from the last point of rotation to the next centre of mass

NFrames Number of bodies

Mass Vector with the mass of each body

Jlist Matrix with the mass moment of inertia tensor for each body

g Gravity constant

MFMNumEquMaker

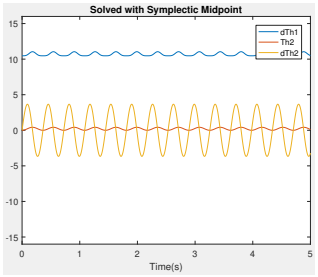
The algorithm starts by calling SymVMaker C.4.2 to create the symbols for the angles, angular velocity and moments. Using the output from SymVMaker, the $D\_B\_dB\_Maker$ C.4.3 and Fmaker C.4.4 is called to create the $[D(t)]$ matrix, $[B(t)]$ matrix, the $[dB(t)]$ matrix and the $F(t)$ vector.

Before continuing on the equation of motion the algorithm saves the velocity, the equations that gives the position of the centres of mass and the coordinates of the rotation points to separate files to facilitate the energy calculation and plotting of the system. To find the velocities, the algorithm multiplies the $[B(t)]$ matrix with the essential generalized coordinates, they are then saved to a file using VelocitySaver.mat C.5.1. The centre of mass functions and rotation point function are created and saved by PosVectorFunctionMaker.mat C.5.2. Examples of such files can be viewed in C.7.2, C.7.3 and C.7.4.

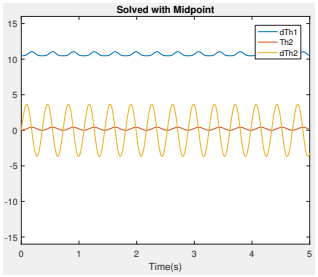
Continuing with the equation of motion, the only component missing is the mass matrix $[M]$, it is constructed by NumMassMatrixMaker.mat. NumMassMatrixMaker takes the mass and mass moment of inertia of the system and saves them to the correct positions in the $[M]$ matrix. Thus the M^*, N^* and F^* is calculated using (83) and the definitions given in (4.6.3). The M^* is then inverted using C.4.6 and the equation of motion is evaluated and simplified before being saved to a file by SaveNumEndEqu.mat C.4.7. An exapmle of one such file is given in the appendix C.4.8.

6.2 Hamilton equations of motion

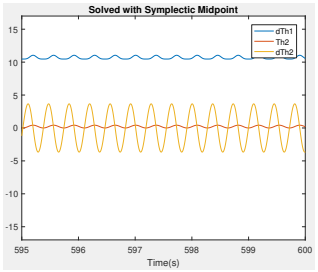
Following is the results from Hamiltons canonical equations of motion and the result from the same system solved with the algorithm. Presented are the angular velocity of the first body, and both angle and angular velocity of the second body. Angle of the first body is not shown as it is only a straight line, and does not add anything significant to the thesis.



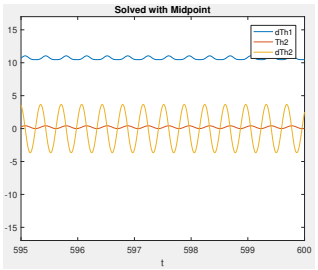
(a) Result from symplectic midpoint, start of time



(b) Result from midpoint, at the start of time



(c) Result from symplectic midpoint, at the end of time



(d) Result from midpoint, at the end of time

Figure 10: Resulting behaviour from Hamiltonian equations of motion and the algorithm

Presenting the energy error of the system, when solving with the midpoint rule. It shows how the energy changes overtime due to numerical instability.

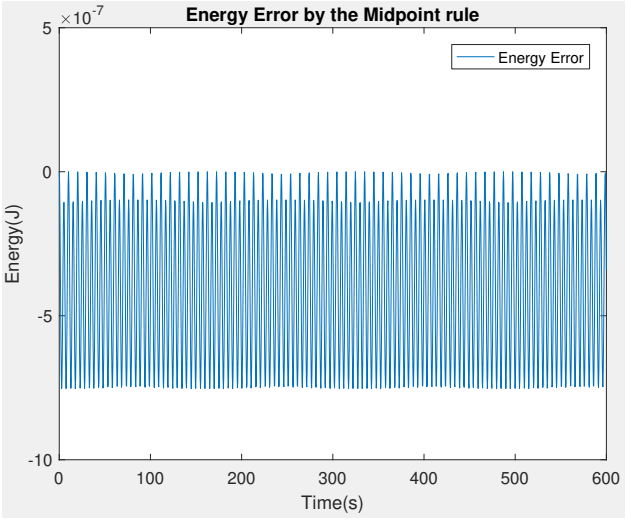


Figure 11: Energy error from midpoint rule

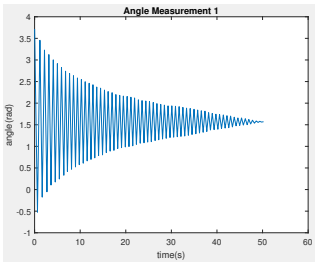
6.3 Measurements

The measurements taken of the pendulum are divided into two categories, the needed measurement for the modelling of the friction functions and the measurements of the main behaviour of the pendulum. Modelling the friction of the arm is done in steps, to see what functions that seems to fit the best. They are assumed to be of the form described by Lord Rayleigh [2, ch1,p.23], where the frictional force is proportional to the velocity.

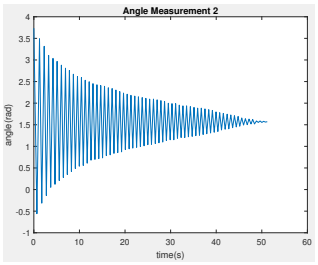
6.3.1 Measurements For Friction Models

Arm

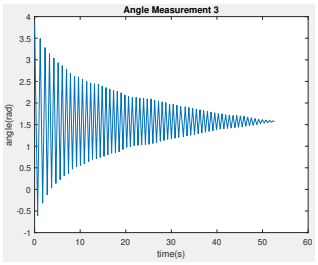
The column is held in place as the arm is raised and released. The angle is measured each time the pendulum reaches an end point 12. Measurement repeated 4 times.



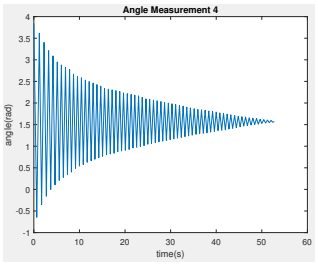
(a) Angle of the arm from first measurement



(b) Angle of the arm from second measurement



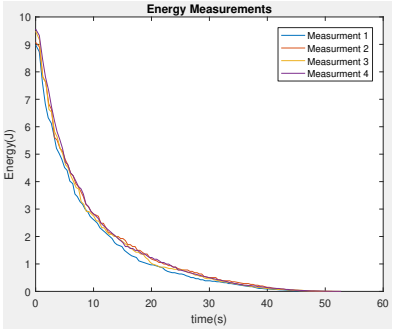
(c) Angle of the arm from third measurement



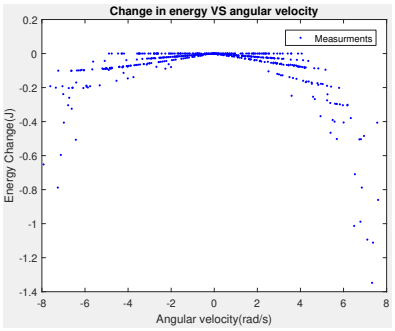
(d) Angle of the arm from fourth measurement

Figure 12: The four measurements of the angle, for use in the arm diffusion function

The potential energy is calculated using the measured angles in 12 and equation (121). With $r^{(2/P)} = sin(\theta^{(2)}(t))0.126m$ and $m^{(2)} = 4.6811kg$. Physical values given by Creo Parametric A.1. Using the difference in angle, and time used between the measurements, the average angular velocity is calculated and plotted against the change in energy between each oscillation 13.



(a) The measured energy in the arm over time



(b) Energy change in the arm plotted against angular velocity

Figure 13: The diffusion of energy in the Arm

By observing figure 13, two diffusion functions are made using lsqnonlin, plotting results together with measured diffusion 14.

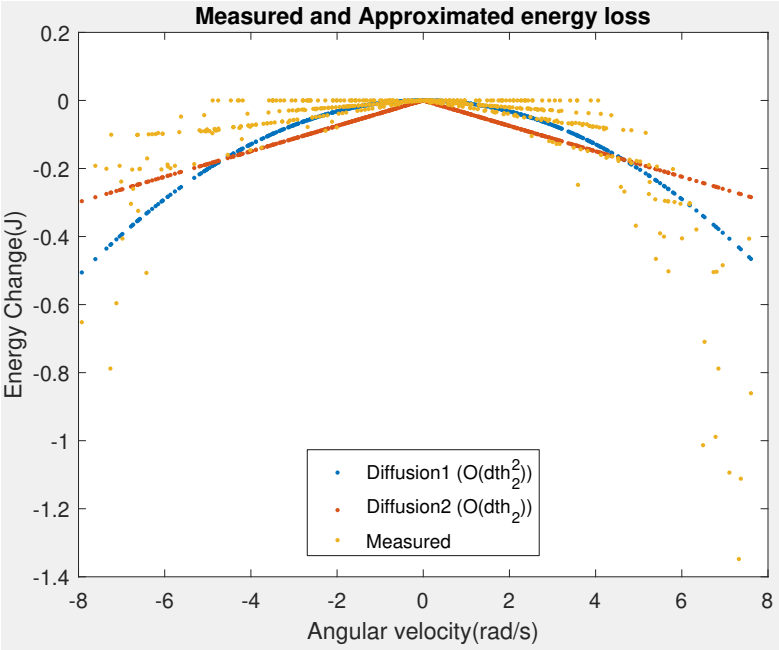


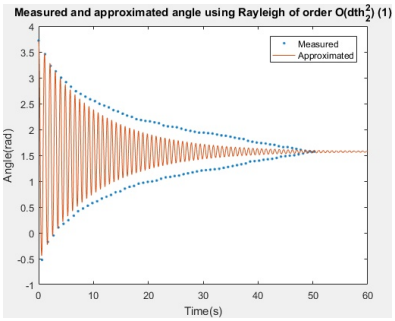
Figure 14: Energy change in the arm plotted against angular velocity, with first diffusion tests

Testing of friction models

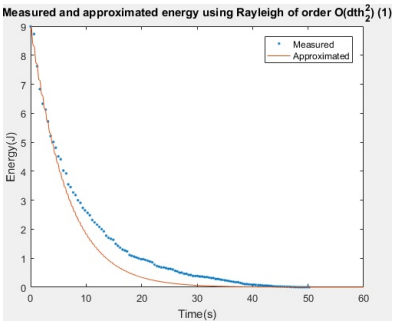
Presenting the resulting behaviour of the arm with different friction functions. In the following section the C_i is different constants and $n(\dot{\theta}^{(2)})$ is the direction vector dependent on the angular velocity.

First test

First friction test with $F_{Arm_1} = -C_1\dot{\theta}^{(2)}$



(a) The measured and approximated angle in the arm with $F_{Arm_1} = -C_1\dot{\theta}^{(2)}$ over time

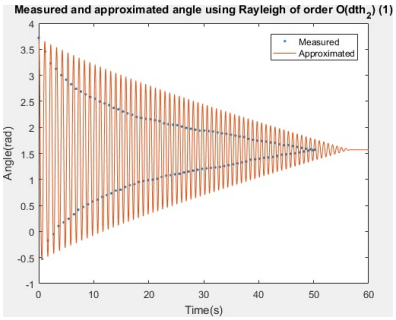


(b) The measured and approximated energy in the arm with $F_{Arm_1} = -C_1\dot{\theta}^{(2)}$ over time

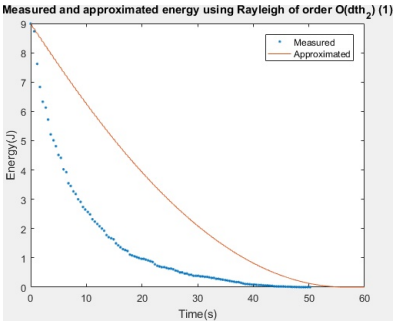
Figure 15: The results after testing the friction force $F_{Arm_1} = -C_1\dot{\theta}^{(2)}$

Second Test

Second friction test with friction function $F_{Arm_2} = -C_2 n(\dot{\theta}^{(2)})$.



(a) The measured and approximated angle in the arm with $F_{Arm_2} = -C_2 n(\dot{\theta}^{(2)})$ over time

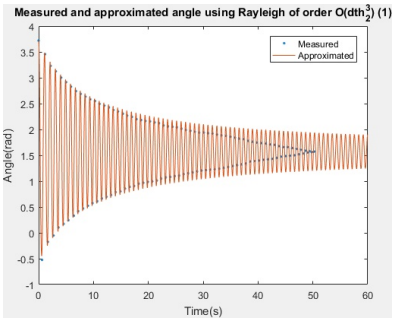


(b) The measured and approximated energy in the arm with $F_{Arm_2} = -C_2 n(\dot{\theta}^{(2)})$ over time

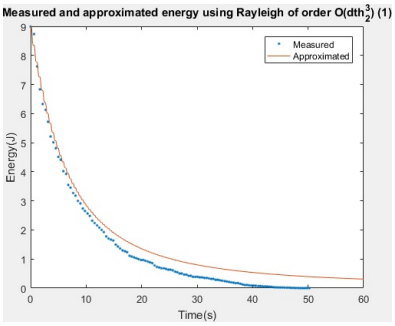
Figure 16: The results after testing with the friction force as a constant

Third test

Third friction test with $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)})$



(a) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)}$ over time

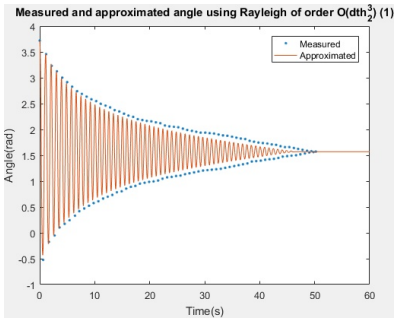


(b) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)}$ over time

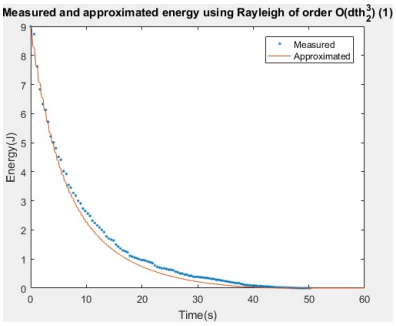
Figure 17: The results after testing the friction force $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)})$

Fourth test

Fourth friction test with $F = \left(C_3\dot{\theta}_2^2 + (C_4 - 0.02)\right) n(\dot{\theta}^{(2)})$



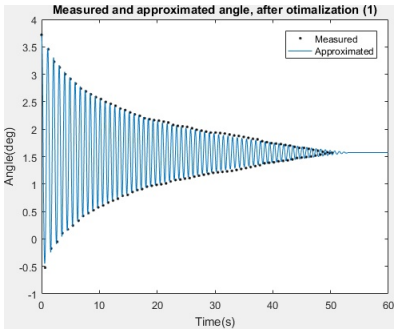
(a) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4 - 0.02)\right) n(\dot{\theta}^{(2)})$ over time



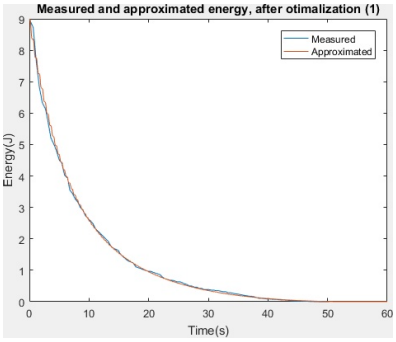
(b) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4 - 0.02)\right) n(\dot{\theta}^{(2)})$ over time

Figure 18: The results after testing the friction force $F = \left(C_3\dot{\theta}_2^2 + (C_4 - 0.02)\right) n(\dot{\theta}^{(2)})$

Friction function test after optimization by lsqnonlin with friction force $F = \left(C_3\dot{\theta}_2^2 + C_4\right) n(\dot{\theta}^{(2)})$



(a) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + C_4\right) n(\dot{\theta}^{(2)})$ over time after optimization

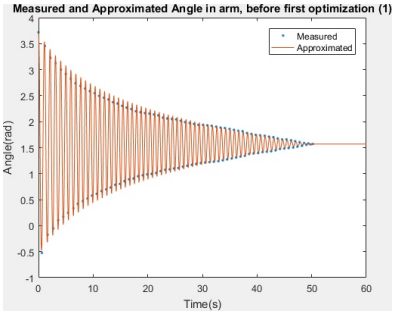


(b) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + C_4\right) n(\dot{\theta}^{(2)})$ over time after optimization

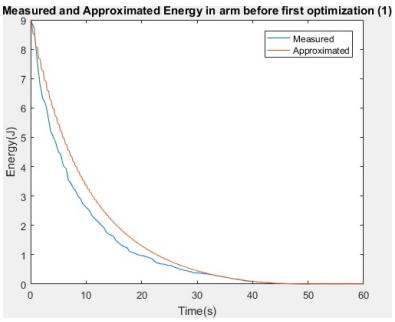
Figure 19: The results after testing the friction force $F = \left(C_3\dot{\theta}_2^2 + C_4\right) n(\dot{\theta}^{(2)})$ after optimization

Reaction force as friction

Friction function test with the equations for reaction forces on the bearings, before optimization.



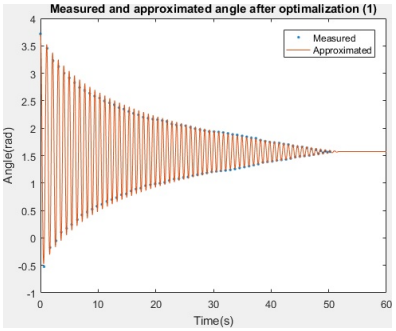
(a) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time before optimization



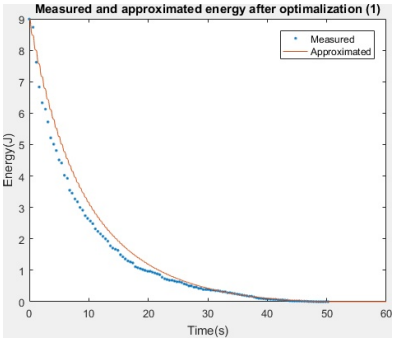
(b) The measured and approximated energy in the arm with friction force modelled on reaction forces on the bearings over time before optimization

Figure 20: The results after testing the friction modelled on the reaction forces on the bearings before optimization

Friction function test with the equations for reaction forces on the bearings, after optimization.



(a) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time after optimization



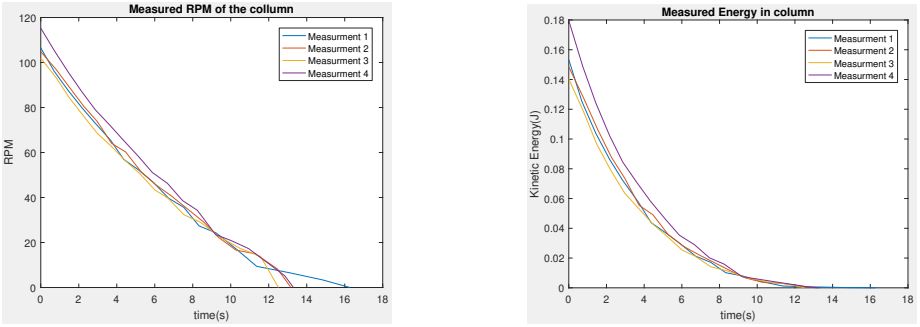
(b) The measured and approximated energy in the arm with friction force modelled on reaction forces on the bearings over time after optimization

Figure 21: The results after testing the friction modelled on the reaction forces on the bearings after optimization

The Column

Continuing on the column, it is assumed that the friction force on the column is of the same form as the ones on the arm. This is done on the basis that the bearings that are used to hold the column are of the same type as the ones on the arm, albeit a couple of sizes bigger.

The column is accelerated to approximately 100RPM 4 times and released. While it is decelerating freely the RPM is measured at time intervals of approximately 0.7 seconds 22a. Using the Measured RPM, the energy of the column is calculated using (118).

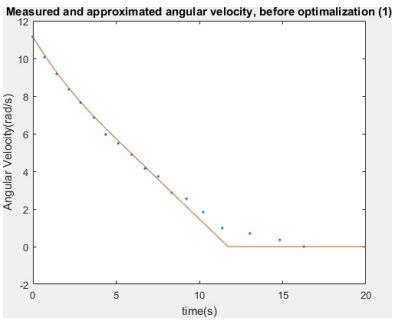


(a) The measured RPM in the column (b) The resulting energy in the column

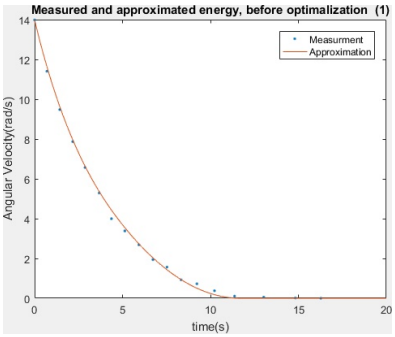
Figure 22: Measured data from the Column

Friction force

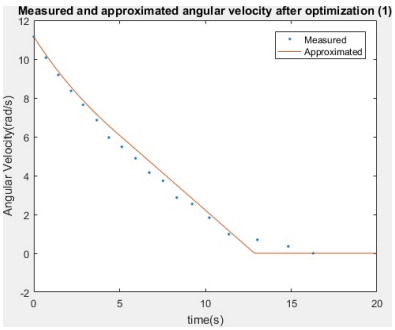
Two test are performed and both are optimized, one with the full force equation the bearings and one simplified with $F_{Column} = \left(C_7 \dot{\theta}^{(1)^2} + C_8\right) n(\dot{\theta}^{(1)})$. The resulting behaviour is plotted with the measured values. Only the first comparison is presented here, the rest can be viewed in the appendix B.2.



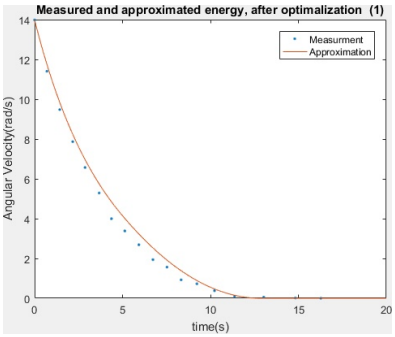
(a) The measured and approximated angular velocity in the column with friction force modelled on reaction forces on the bearings over time before optimization



(b) The measured and approximated energy in the column with friction force modelled on reaction forces on the bearings over time before optimization

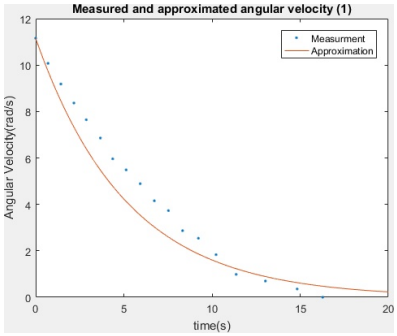


(c) The measured and approximated angular velocity in the column with friction force modelled on reaction forces on the bearings after optimization

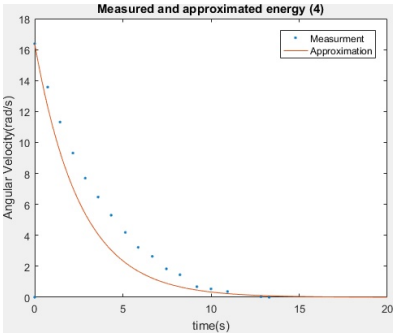


(d) The measured and approximated energy in the column with friction force modelled on reaction forces on the bearings over time after optimization

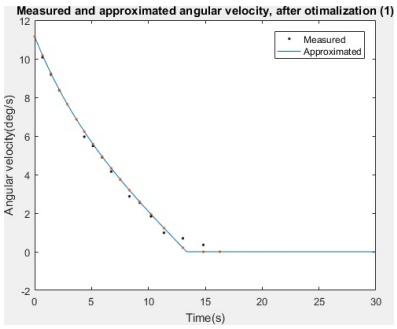
Figure 23: The results after testing the friction modelled on the reaction forces on the bearings before and after optimization



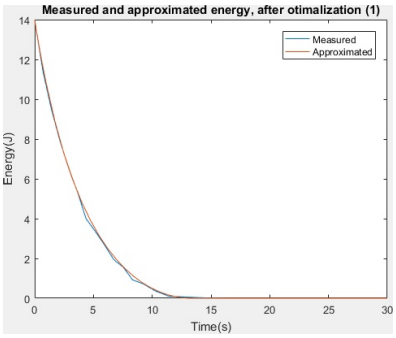
(a) The measured and approximated angular velocity in the column with friction $F_{Column} = \left(C_7\dot{\theta}^{(1)^2} + C_8\right) n(\dot{\theta}^{(1)})$ before optimization



(b) The measured and approximated energy in the column with friction $F_{Column} = \left(C_7\dot{\theta}^{(1)^2} + C_8\right) n(\dot{\theta}^{(1)})$ before optimization



(c) The measured and approximated angular velocity in the column with friction $F_{Column} = \left(C_7\dot{\theta}^{(1)^2} + C_8\right) n(\dot{\theta}^{(1)})$ after optimization

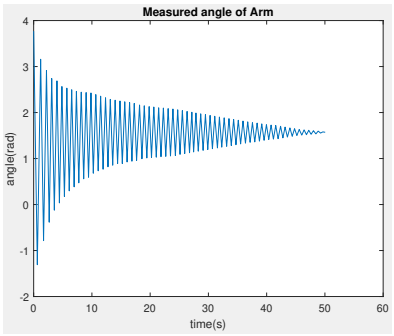


(d) The measured and approximated energy in the column with friction $F_{Column} = \left(C_7\dot{\theta}^{(1)^2} + C_8\right) n(\dot{\theta}^{(1)})$ after optimization

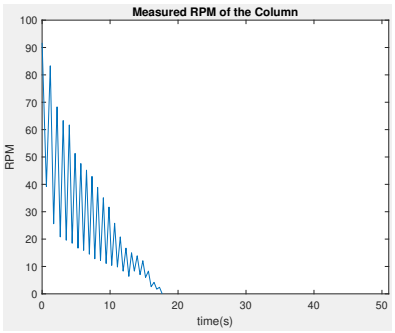
Figure 24: The result after testing the friction $F_{Column} = \left(C_7\dot{\theta}^{(1)^2} + C_8\right) n(\dot{\theta}^{(1)})$, before and after optimization

6.3.2 Main Behaviour Measurements

The Pendulum is accelerated with the arm held in place. When the measured RPM of the column reach approximately 100 RPM, the arm is released. At that point the angular velocity of the Column where 91.7 RPM, and the angle of the arm were 3.6477 rad. Measurements are taken at each top point of the arm 25. Using the angular velocity of the Column and the angle of the Arm, the energy of the Pendulum is calculated using (75) and the height of the arm above the lowest point it can reach in the inertial third direction.

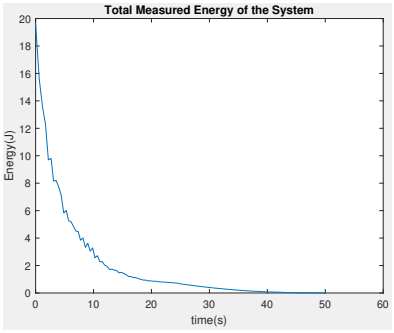


(a) The angle of the arm as the Pendulum moves

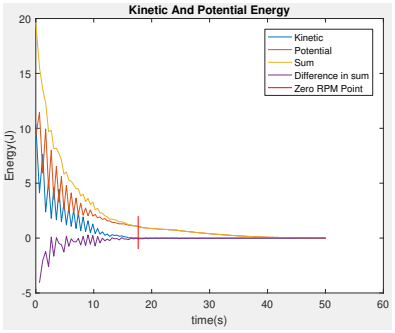


(b) The RPM of the Column as the Pendulum moves

Figure 25: The measured behaviour of the Pendulum



(a) The resulting measured energy of the Pendulum

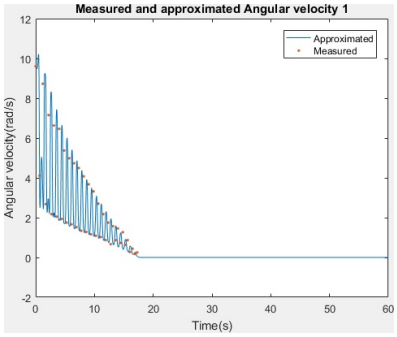


(b) The kinetic, potential, difference and total measured Energy of the Pendulum

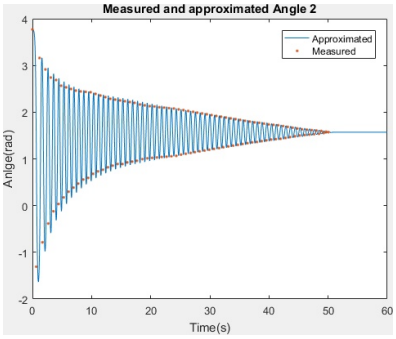
Figure 26: The resulting energy of the Pendulum

6.4 Comparing theory and reality

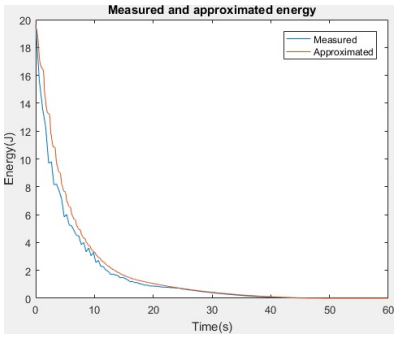
The resulting plots from the numerics and the real pendulum is presented, following is the results from when the friction is modelled on the reaction forces and the simplified version.



(a) The RPM of the column, approximated and measured after optimization

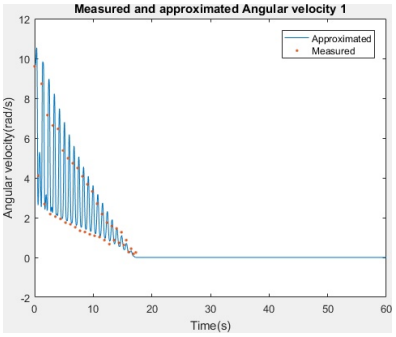


(b) The angle of the Arm, approximated and measured after optimization

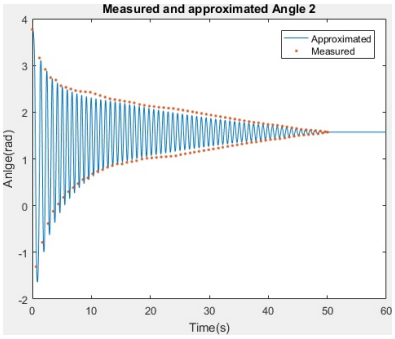


(c) The energy of the Pendulum, approximated and measured after optimization

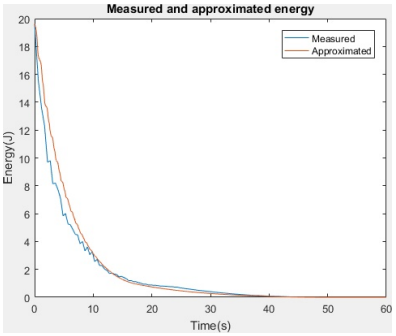
Figure 27: The measured and approximated behaviour of the Pendulum with friction force as reactions at bearings



(a) The RPM of the column, approximated and measured after optimization



(b) The angle of the Arm, approximated and measured after optimization



(c) The energy of the Pendulum, approximated and measured after optimization

Figure 28: The measured and approximated behaviour of the Pendulum, with simplified friction force

6.5 Testing the Algorithm

Testing of the algorithm is done by running it with increasing number of bodies. The time the algorithm uses, size of the resulting file and time it takes to call said file once is measured for each run. It was assumed that the simplifying the inverse of $[M^*]$ and the end equation, would shorten the equations. Therefore the algorithm is run two times for each number of bodies. One time with simplify and one without. This was done for bodies up to 3. For equations involving 4 or more bodies, simplify maxed out the memory of the computer (15GB) and stopped. As such the last tests where run without simplifying the inverse of $[M^*]$ and the end equation.

nr of bodies	Time used by algorithm	file size	time to call func	note
1	3.2sec	79 byte	0.001sec	with simplify
1	2.7sec	79 byte	0.002sec	without simplify
2	3.65sec	299 byte	0.002sec	with simplify
2	2.4sec	287 byte	0.002sec	without simplify
3	12.2sec	9,60 kB	0.05sec	with simplify
3	5.16sec	11,3 kB	0.07sec	without simplify
4	11.05sec	446 kB	2.3sec	without simplify
5	88.4sec	13,0 MB	14.4sec	without simplify
6	1597.3sec	298 MB	N/A	without simplify

In the last test Matlab was unable to call the func file. The operation was terminated with the following error: Out of memory

7 Evaluation

7.1 Validation

First step of the validation was to manually evaluate the equations of motion and comparing these with the equation given from the algorithm. This test was successful in showing that the pattern found in the $B(t)$ and $\dot{B}(t)$ is correct and that the algorithm found does reproduce the equations of motion for systems of linked rigid bodies. Furthermore, the Hamilton canonical equations of motion gives the same behaviour when applied to the pendulum with friction less movement. By observing the figures in figure 10, it is clear that the different equations of motion reproduce the same behaviour. Although at the end of the integration the frequency of the arm differs somewhat this is the result of inaccuracies in the integration and not a bug in the algorithm itself.

Lastly, there is the replication of the movement of the pendulum. By observing the figures in 27 and 28, it can be said that the equations of motion constructed by the algorithm does replicate the actual motion of the pendulum. Although the friction models does have a visible error from the measurements, it should be pointed out that the measurements themselves are rough at best. Furthermore, Rayleigh's diffusion function seems to be too simple to be of use in the modelling of systems of linked rigid bodies. Even in a simple system as the pendulum the centrifugal force and gravity are the biggest contributors, and thus the friction must be modelled thereafter.

7.2 Performance

By evaluating the performance of the algorithm shown in table 6.5, a clear limitation is apparent. For 4 bodies with rotational axes that are perpendicular to each other the end equation becomes so big that it takes 2.3 seconds to call the acceleration file. When integrating numerically this file will be called millions of times, and thus using several seconds to run one time step would slow the integration down significantly. For reference the midpoint rule integration method used to produce the figures in 10 uses 2 400 001 time-steps. That would mean a time consumption of a total 63 days to finish integrating, and that does not take into account the fixed point iteration used for each time-step. Therefore the algorithm as it is with constructing the needed parts as symbolic functions and then evaluating them to get the equations of motion is limited to 3 bodies.

7.3 Pendulum and friction models

By looking at the modelling of the pendulum as a project on its own, the measuring of the pendulum and subseqent modelling of the friction did work. The method used in measuring and modelling of the pendulum, are simple enough that a small group of students could perform them as a class project. It should be pointed out that it is possible to find the friction models by only using the main measurements. This is due to the friction in the bearings are a functions of both angular velocities, and thus must be optimized for the full behaviour. Furthermore, by only focusing on the main measurements the project would shorten in the sense that the students would not have to take as many different measurements and thus save a lot of time else used to stare at a video and write numbers in Microsoft xls.

8 Conclusion

8.1 Algorithm

This thesis has used the symbolic toolbox in Matlab to construct the equations of motion as a symbolic function, then by turning the equation into strings it has been saved to a separate file. While this facilitates the use of generic integration schemes like ode45 and the midpoint rule, it has shown itself to be highly ineffective as the size of the resulting file grows exponentially as the number of bodies in the system is increased.

Although the symbolic approach to solving for the equations of motion has shown itself to be too ineffective, the critical components needed has been identified. As shown in 6.1, the equations follow a predictable pattern and can thus be coded in a generic manner. While the algorithm itself is complicated, it only relies on a set of basic components to be able to run. That is the rotational matrix, the time derivative of the rotational matrix and the angular velocity matrix for each body. While there can be many bodies, for known rotational axis's there is only three distinct versions of the matrices. Thus it can be concluded that there is indeed a way to handle full 3D motion, with applied non-conservative forces and moments.

8.2 Pendulum and friction incorporated in teaching

As mentioned in the introduction, the constructed model stands on its own as an example on how a practical project could be incorporated in the teaching of mechanics to enhance learning and motivation among students. As shown in the method and results, even simple and rough measurements as the ones done during this project enables the modelling of a real world contraption. By having students in small groups performing the measurements to find the frictional constants and then modelling the behaviour of the pendulum will induce an enhanced sense of accomplishment and thus increase the likelihood of them remembering the theory.

For students, to see how the theory they are presented with can be applied directly to reality is highly motivational. Furthermore, an argument can be made that students see too much of a blackboard and too little reality during their studies in science and engineering in particular. By incorporating practical projects such as this in teaching, the effect of discussing phenomenon such as centrifugal force and the conservation of angular momentum will be higher when student see the effects in real life.

9 Future Work

9.1 Future of the algorithm

The next step of the algorithm will be to find alternate ways to model the systems using the found pattern in the equations of motion. With the critical components identified, they can be supplemented by pre scripted functions. Thus the equation of motion could be evaluated for each time step, making the need to save absurdly long strings obsolete. Although the efficiency of such an approach could be in question, it would make the modelling of bigger systems possible.

9.2 Enhancing the pendulum

Moving on with the model itself will be to install digital measuring devices onto both bodies. The natural choice is inductive censors equivalent to the ones used in the ABS systems in cars and trucks. Censors such as these will enable much more accurate measurements of the angular velocities of the bodies and thus actual accuracy of the mathematical models can be discussed.

Further more, the pendulum should be expanded with more bodies. With more accurate measurements the next step would be to model more chaotic systems, and the double pendulum being a well known chaotic system it is the natural next step. As such the addition of another body on the end of the pendulum is a good choice as a next expansion.

References

[1] Murakami,H. & Impelluso,T., *Moving Frame Method in Dynamics - A Geometrical Approach*: Pearson Publishing, N.J. Publication Pending.

[2] Herbert Goldstein, John L. Safko, Charles P. Poole Jr, *Classical Mechanics*: Pearson New International Edition, 2013. 3rd ed.

[3] Grote, K. & Antonsson, E.,K., *Springer Handbook of Mechanical Engineering* Springer-Verlag Berlin Heidelberg 2009. 1st ed.

[4] Iserles, A. (2008). *A First Course in the Numerical Analysis of Differential Equations (Cambridge Texts in Applied Mathematics).* Cambridge: Cambridge University Press. doi:10.1017/CBO9780511995569

[5] Murakami,H., *A moving frame method for multi-body dynamics* Proceedings of the ASME 2013 International Mechanical Engineering Congress & Exposition, paper IMECE2013-62833.

[6] Murakami,H., *A moving frame method for multi-body dynamics using SE(3)* Proceedings of the ASME 2015 International Mechanical Engineering Congress & Exposition, paper IMECE2015-51192.

[7] Impelluso,T., *Rigid body dynamics: A new philosophy, math and pedagogy* Proceedings of the ASME 2016 International Mechanical Engineering Congress & Exposition, paper IMECE2016-65970.

[8] Impelluso, T., *The moving frame method in dynamics: Reforming a curriculum and assessment* International Journal of Mechanical Engineering Education. 2017, August 30,.

[9] Murakami,H., Rios,O., Impelluso,T., *A Theoretical and Numerical Study of the Dzhanibekov and Tennis Racket Phenomena.* Journal of applied mechanics 2016; Volume 83, No. 11.

[10] Iserles, A., Munthe-Kaas, H., Nørsett, S., Zanna, A., *Lie-group methods.* Acta Numerica 2000, 9, 215-365.

[11] Rykkje, T., R., Leinebø, D., Bergaas,E.,S., Skjelde, A., Impelluso., T., *Inspiring learning: Assessment of friction in a real-world model using the moving frame method in dynamics.* Proceedings of the ASME 2018 International Mechanical Engineering & Exposition, Paper IMECE2018-86189. **Publication Pending**

A Data from Creo Parametric

A.1 Arm Data

Data on the Arm given from Creo Parametric.

```
VOLUME =  6.0014229e+05  MM^3
SURFACE AREA =  5.7600335e+04  MM^2
DENSITY =  7.8000000e-09 TONNE / MM^3
MASS =  4.6811099e-03 TONNE

CENTER OF GRAVITY with respect to ROTPNKT coordinate frame:
X   Y   Z     1.2675638e+02  0.0000000e+00  0.0000000e+00  MM

INERTIA at CENTER OF GRAVITY with respect to ROTPNKT
coordinate frame:(TONNE*MM^2)

INERTIA TENSOR:
Ixx Ixy Ixz  4.1857344e+00  0.0000000e+00  0.0000000e+00
Iyx Iyy Iyz  0.0000000e+00  1.9213110e+01  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  1.9778369e+01

PRINCIPAL MOMENTS OF INERTIA:  (TONNE * MM^2)
I1  I2  I3   4.1857344e+00  1.9213110e+01  1.9778369e+01

ROTATION MATRIX from ROTPNKT orientation to PRINCIPAL AXES:
1.00000        0.00000        0.00000
0.00000        1.00000        0.00000
0.00000        0.00000        1.00000

ROTATION ANGLES from ROTPNKT orientation to PRINCIPAL AXES (degrees):
angles about x  y  z   0.000          0.000          0.000

RADII OF GYRATION with respect to PRINCIPAL AXES:
R1  R2  R3 2.9902770e+01  6.4065529e+01  6.5001117e+01  MM
```

A.2 Base Data

Data on the Base, given by Creo Parametric.

```
VOLUME =  4.3134681e+05  MM^3
SURFACE AREA =  2.2608337e+05  MM^2
DENSITY =  7.8000000e-09 TONNE / MM^3
MASS =  3.3645051e-03 TONNE

CENTER OF GRAVITY with respect to CS1 coordinate frame:
X   Y   Z     1.1705442e+02  3.0102662e+02  1.9696113e+01  MM

INERTIA with respect to CS1 coordinate frame:  (TONNE * MM^2)

INERTIA TENSOR:
Ixx Ixy Ixz  3.8939946e+02 -1.2208736e+02 -1.5436827e+01
Iyx Iyy Iyz -1.2208736e+02  1.0913135e+02 -2.0542999e+01
Izx Izy Izz -1.5436827e+01 -2.0542999e+01  4.6662300e+02

INERTIA at CENTER OF GRAVITY with respect to CS1 coordinate frame:  (TONNE *

INERTIA TENSOR:
Ixx Ixy Ixz  8.3212803e+01 -3.5339833e+00 -7.6799028e+00
Iyx Iyy Iyz -3.5339833e+00  6.1726572e+01 -5.9466628e-01
Izx Izy Izz -7.6799028e+00 -5.9466628e-01  1.1564199e+02

PRINCIPAL MOMENTS OF INERTIA:  (TONNE * MM^2)
I1  I2  I3   6.1097854e+01  8.2114024e+01  1.1736949e+02

ROTATION MATRIX from CS1 orientation to PRINCIPAL AXES:
0.16941       -0.96074       -0.21971
0.98494        0.17287        0.00353
0.03459       -0.21700        0.97556

ROTATION ANGLES from CS1 orientation to PRINCIPAL AXES (degrees):
angles about x  y  z  -0.207        -12.692         80.000

RADII OF GYRATION with respect to PRINCIPAL AXES:
R1  R2  R3 1.3475732e+02  1.5622411e+02  1.8677426e+02  MM
```

A.3 Exploded model of the pendulum

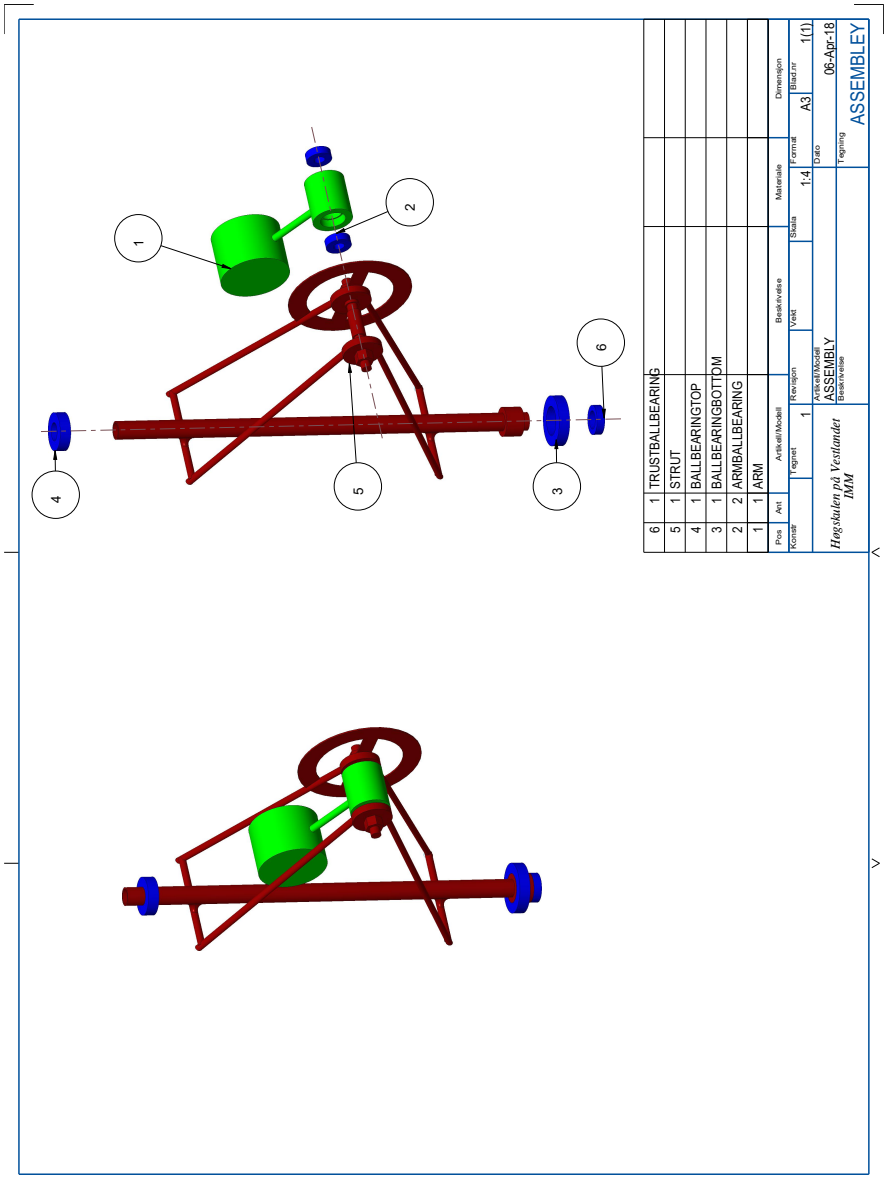


Figure 29: Exploded view of the pendulum

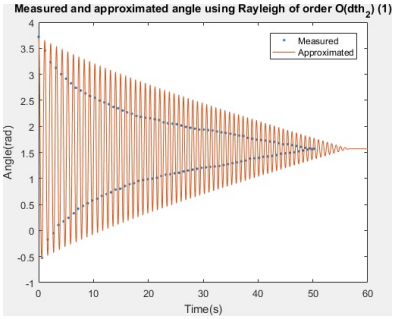
B Friction function trials

B.1 Arm

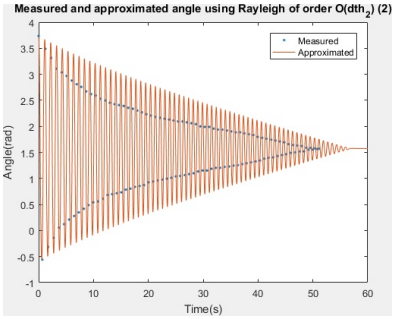
Presented below are the resulting behaviour of the numerical solution when different friction models are tested.

B.1.1 First test

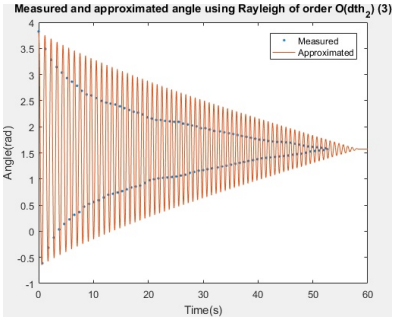
In the following section the C_i is different constants and $n(\dot{\theta}^{(2)})$ is the direction vector dependent on the angular velocity. First test with friction function $F_{Arm} = -C_1 n(\dot{\theta}^{(2)})$.



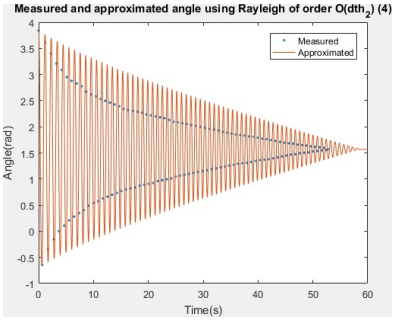
(a) The measured and approximated angle in the arm with $F_{Arm} = -C_1 n(\dot{\theta}^{(2)})$ nr. 1



(b) The measured and approximated angle in the arm with $F_{Arm} = -C_1 n(\dot{\theta}^{(2)})$ nr. 2

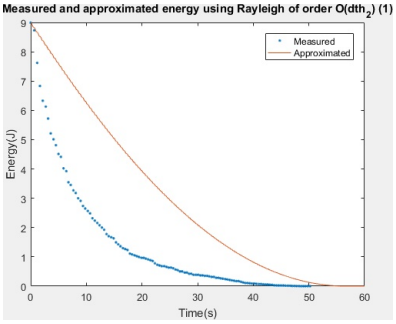


(c) The measured and approximated angle in the arm with $F_{Arm} = -C_1 n(\dot{\theta}^{(2)})$ nr. 3

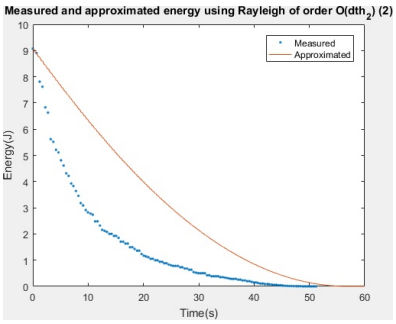


(d) The measured and approximated angle in the arm with $F_{Arm} = -C_1 n(\dot{\theta}^{(2)})$ nr. 4

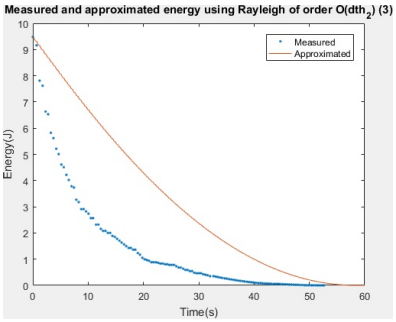
Figure 30: The resulting angle after testing with the friction force as a constant



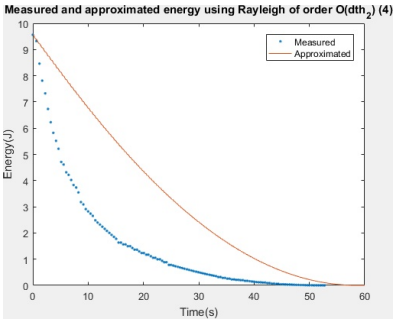
(a) The measured and approximated energy in the arm with $F_{Arm} = -C_1 n(\dot{\theta}^{(2)})$ nr. 1



(b) The measured and approximated energy in the arm with $F_{Arm} = -C_1 n(\dot{\theta}^{(2)})$ nr. 2



(c) The measured and approximated energy in the arm with $F_{Arm} = -C_1 n(\dot{\theta}^{(2)})$ nr. 3

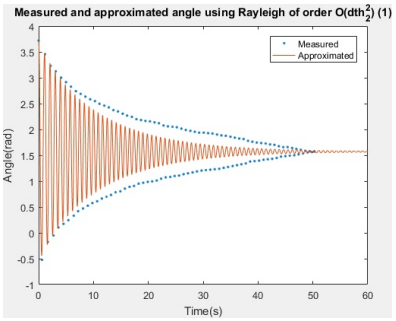


(d) The measured and approximated energy in the arm with $F_{Arm} = -C_1 n(\dot{\theta}^{(2)})$ nr. 4

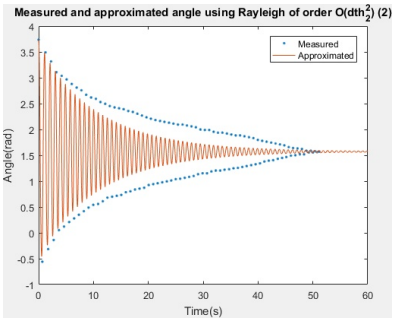
Figure 31: The resulting energy after testing with the friction force as a constant

B.1.2 Second test

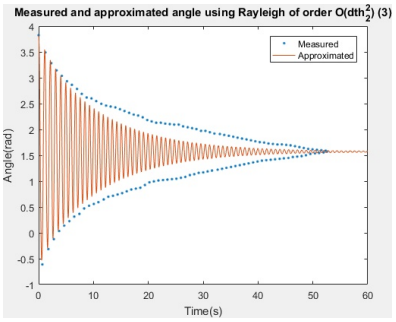
Second test with friction function $F_{Arm} = -C_1 \dot{\theta}^{(2)}$



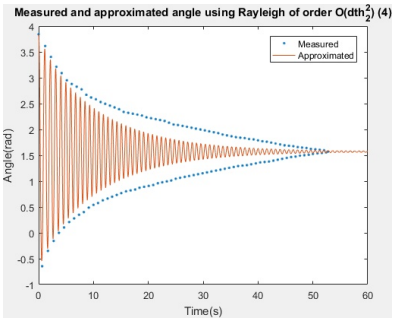
(a) The measured and approximated angle in the arm with $F_{Arm} = -C_2\dot{\theta}^{(2)}$ nr. 1



(b) The measured and approximated angle in the arm with $F_{Arm} = -C_2\dot{\theta}^{(2)}$ nr. 2



(c) The measured and approximated angle in the arm with $F_{Arm} = -C_2\dot{\theta}^{(2)}$ nr. 3

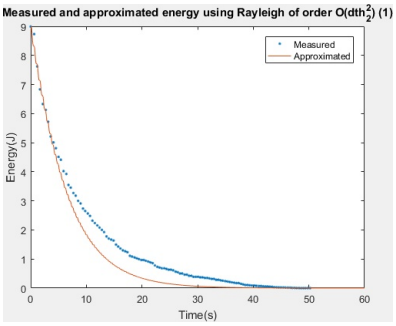


(d) The measured and approximated angle in the arm with $F_{Arm} = -C_2\dot{\theta}^{(2)}$ nr. 4

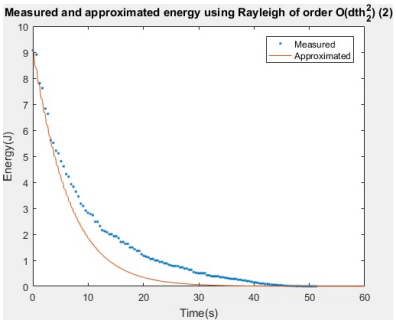
Figure 32: The resulting angle after testing with the friction force as $F_{Arm} = -C_2\dot{\theta}^{(2)}$

B.1.3 Third test

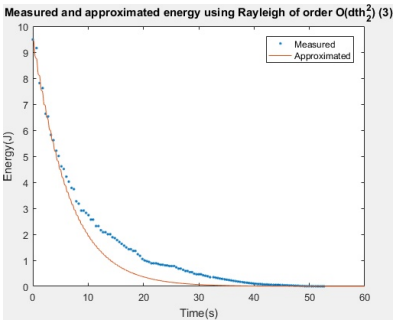
Third friction test with $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)}$



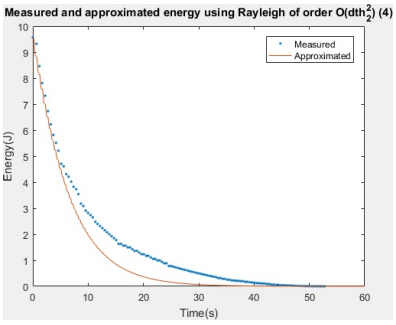
(a) The measured and approximated energy in the arm with $F_{Arm} = -C_1 n(\dot{\theta}^{(2)})$ nr. 1



(b) The measured and approximated energy in the arm with $F_{Arm} = -C_1 n(\dot{\theta}^{(2)})$ nr. 2



(c) The measured and approximated energy in the arm with $F_{Arm} = -C_1 n(\dot{\theta}^{(2)})$ nr. 3

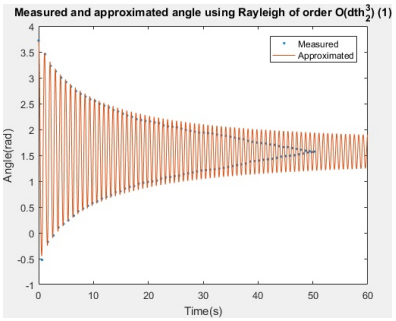


(d) The measured and approximated energy in the arm with $F_{Arm} = -C_1 n(\dot{\theta}^{(2)})$ nr. 4

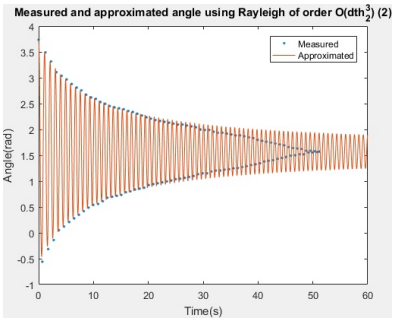
Figure 33: The resulting energy after testing with the friction force as $F_{Arm} = -C_2\dot{\theta}^{(2)}$

B.1.4 Fourth test

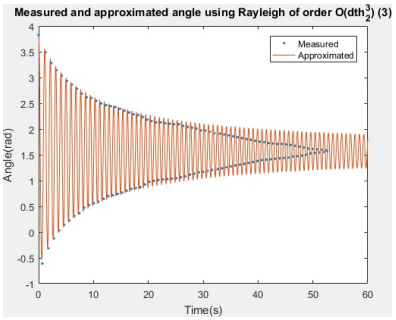
Fourth friction test with $F = \left(C_3\dot{\theta}_2^2 + (C_4 - 0.02)\right) n(\dot{\theta}^{(2)})$



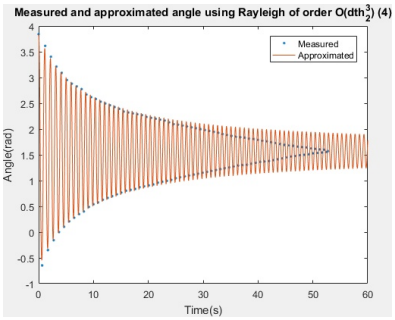
(a) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)}$ nr. 1



(b) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)}$ nr. 2



(c) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)}$ nr. 3

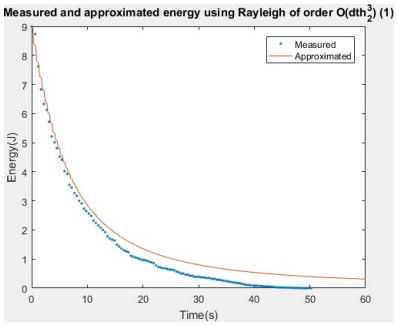


(d) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)}$ nr. 4

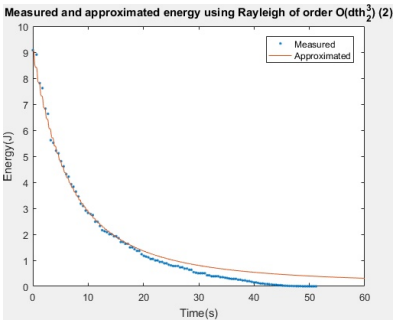
Figure 34: The resulting angles after testing the friction force $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)})$

B.1.5 Fourth test after optimization

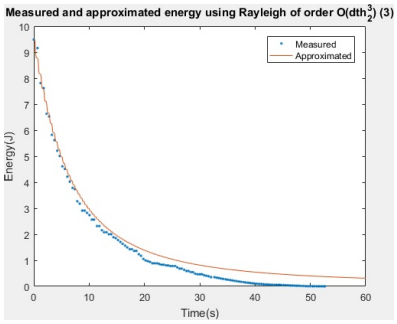
Fourth friction test with $F = \left(C_3\dot{\theta}_2^2 + (C_4)\right) n(\dot{\theta}^{(2)})$ after optimization



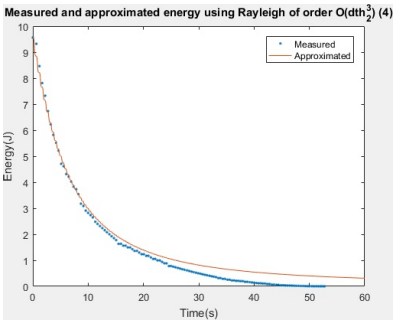
(a) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)}$ nr. 1



(b) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)}$ nr. 2



(c) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)}$ nr. 3

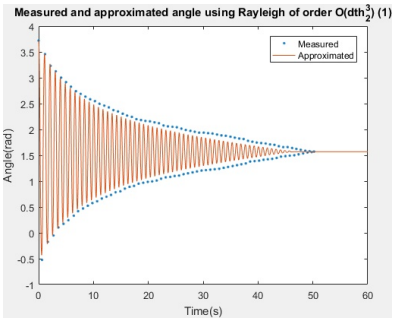


(d) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)}$ nr. 4

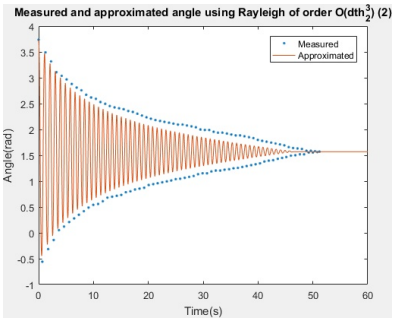
Figure 35: The resulting energy after testing with the friction force as $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)})$

B.1.6 Friction Force as reactions at bearings before optimizing

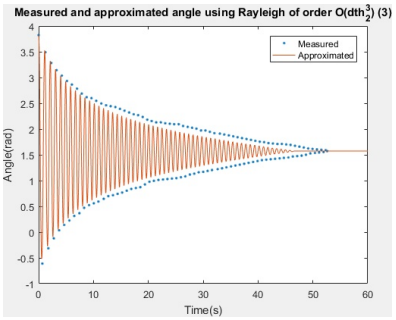
Friction function test with the equations for reaction forces on the bearings, before optimization.



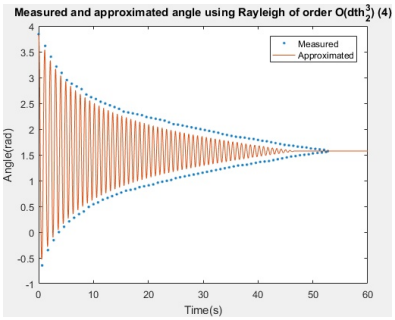
(a) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4 - 0.02)\right) n(\dot{\theta}^{(2)})$ nr. 1



(b) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4 - 0.02)\right) n(\dot{\theta}^{(2)})$ nr. 2



(c) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4 - 0.02)\right) n(\dot{\theta}^{(2)})$ nr. 3

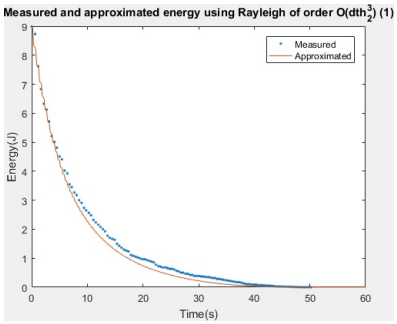


(d) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4 - 0.02)\right) n(\dot{\theta}^{(2)})$ nr. 4

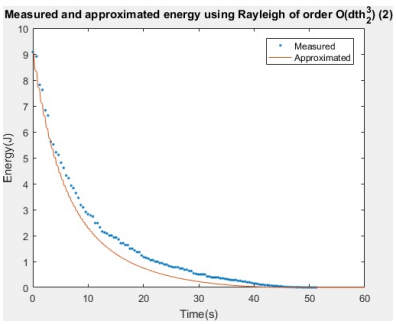
Figure 36: The resulting angles after testing the friction force $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)})$

B.1.7 Friction Force as reactions at bearings after optimizing

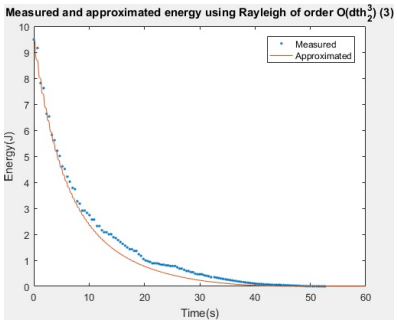
Friction function test with the equations for reaction forces on the bearings, before optimization.



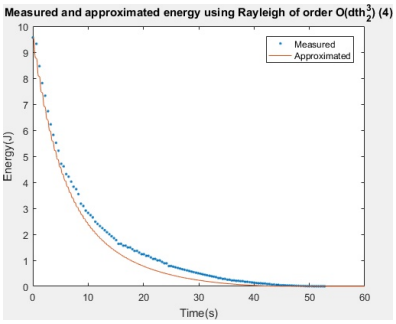
(a) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4 - 0.02)\right) n(\dot{\theta}^{(2)})$ nr 1



(b) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4 - 0.02)\right) n(\dot{\theta}^{(2)})$ nr 2

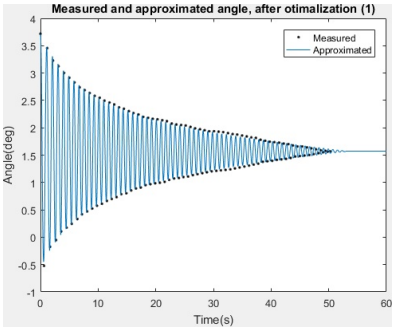


(c) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4 - 0.02)\right) n(\dot{\theta}^{(2)})$ nr 3

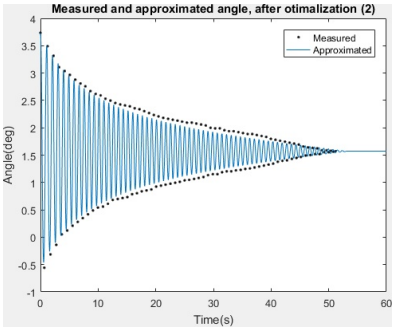


(d) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4 - 0.02)\right) n(\dot{\theta}^{(2)})$ nr 4

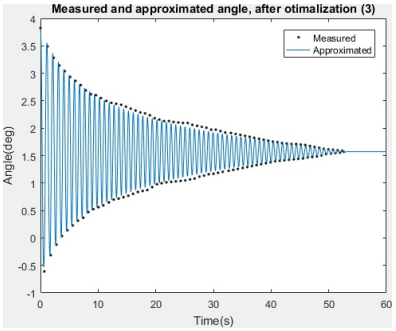
Figure 37: The resulting energy after testing the friction force $F = \left(C_3\dot{\theta}_2^2 + (C_4 - 0.02)\right) n(\dot{\theta}^{(2)})$



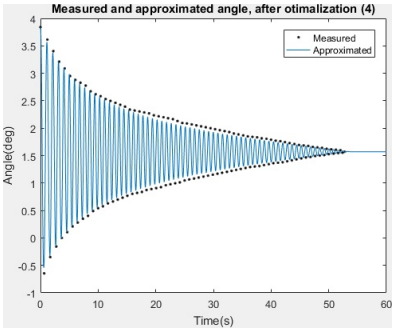
(a) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4)\right) n(\dot{\theta}^{(2)})$ over time after optimization 1



(b) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4)\right) n(\dot{\theta}^{(2)})$ over time after optimization 2

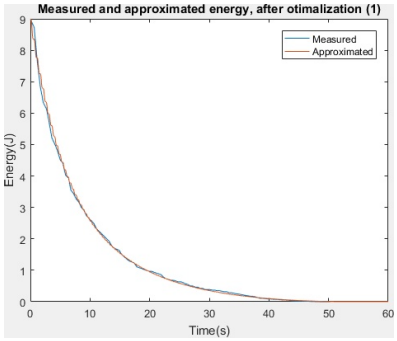


(c) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4)\right) n(\dot{\theta}^{(2)})$ over time after optimization 3

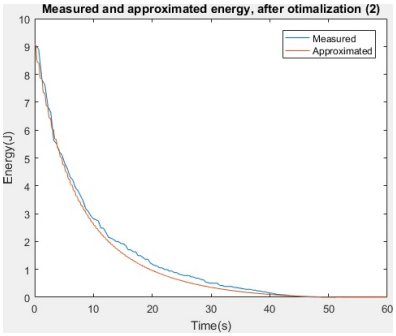


(d) The measured and approximated angle in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4)\right) n(\dot{\theta}^{(2)})$ over time after optimization 4

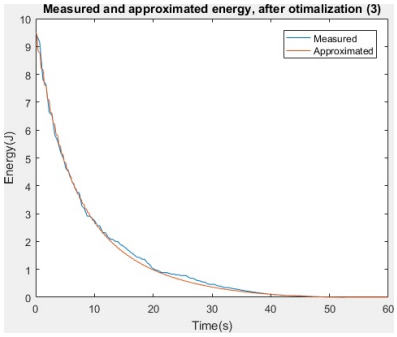
Figure 38: The resulting angles after testing the friction force $F = \left(C_3\dot{\theta}_2^2 + C_4)\right) n(\dot{\theta}^{(2)})$



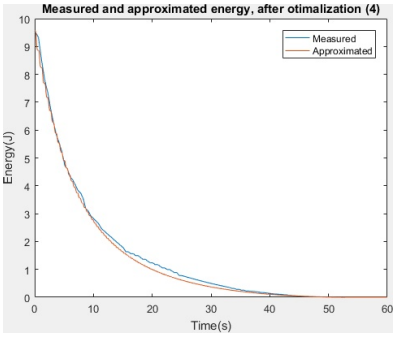
(a) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4)\right) n(\dot{\theta}^{(2)})$ over time after optimization 1



(b) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4)\right) n(\dot{\theta}^{(2)})$ over time after optimization 2

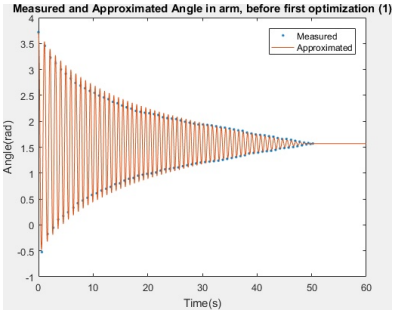


(c) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4)\right) n(\dot{\theta}^{(2)})$ over time after optimization 3

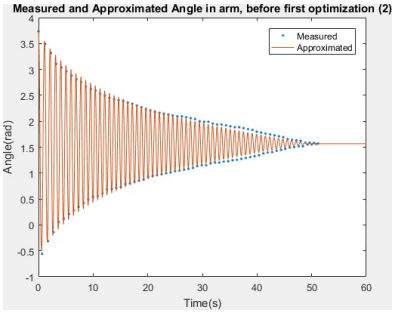


(d) The measured and approximated energy in the arm with $F = \left(C_3\dot{\theta}_2^2 + (C_4)\right) n(\dot{\theta}^{(2)})$ over time after optimization 4

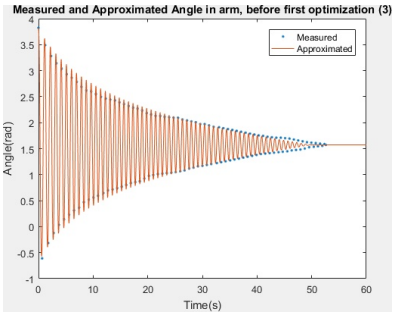
Figure 39: The resulting energy after testing the friction force $F = \left(C_3\dot{\theta}_2^2 + (C_4)\right) n(\dot{\theta}^{(2)})$ after optimization



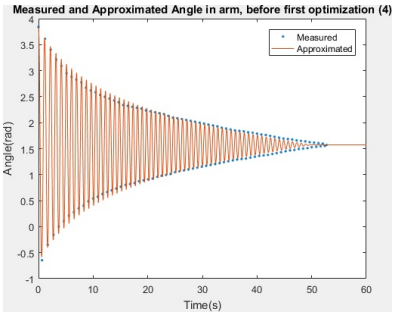
(a) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time before optimization 1



(b) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time before optimization 2

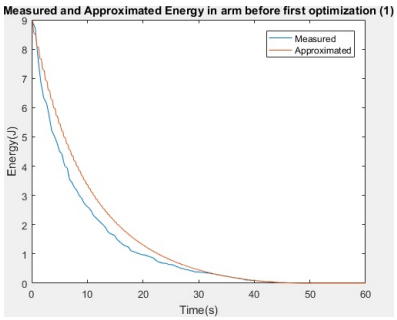


(c) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time before optimization 3

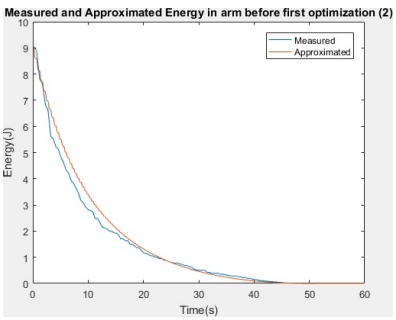


(d) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time before optimization 4

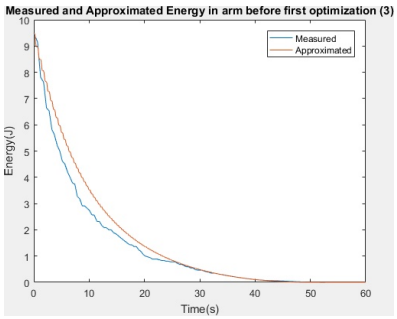
Figure 40: The resulting angle after testing the friction modelled on the reaction forces on the bearings before optimization



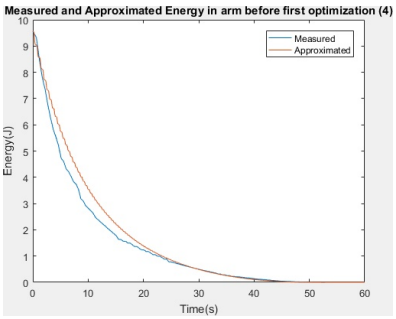
(a) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time before optimization 1



(b) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time before optimization 2

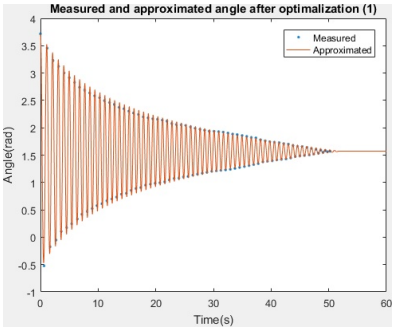


(c) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time before optimization 3

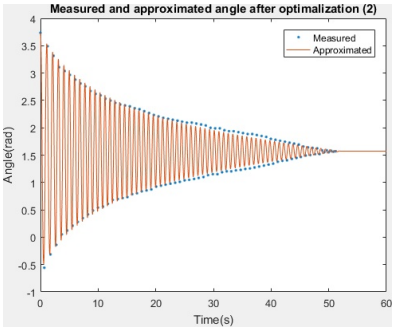


(d) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time before optimization 4

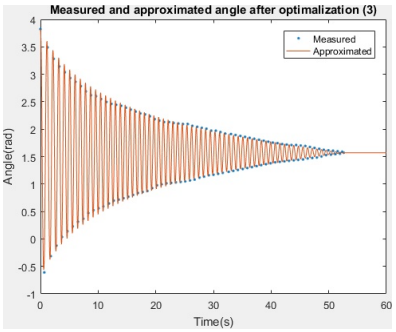
Figure 41: The resulting energy after testing the friction modelled on the reaction forces on the bearings before optimization



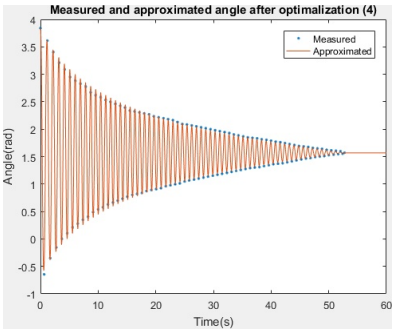
(a) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time after optimization 1



(b) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time after optimization 2

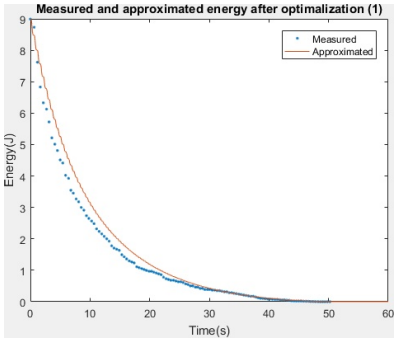


(c) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time after optimization 3

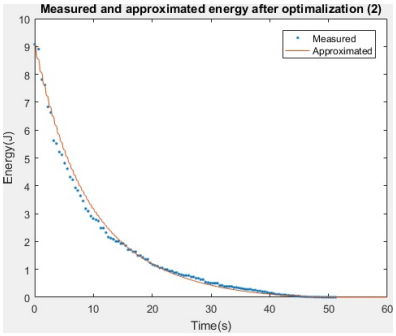


(d) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time after optimization 4

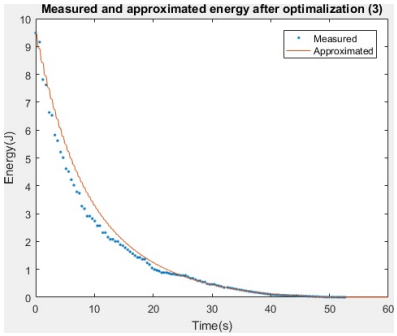
Figure 42: The resulting angles after testing the friction modelled on the reaction forces on the bearings after optimization



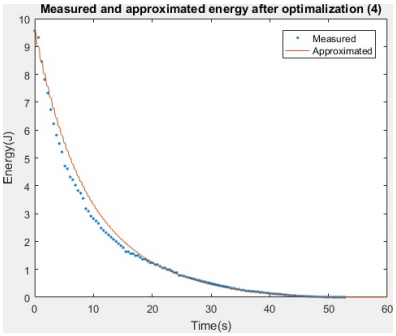
(a) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time after optimization 1



(b) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time after optimization 2



(c) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time after optimization 3

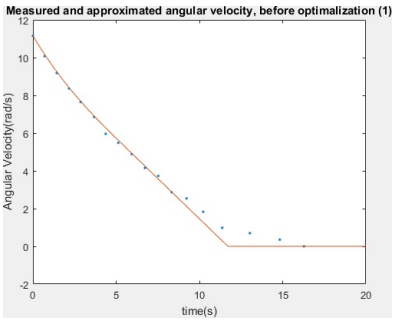


(d) The measured and approximated angle in the arm with friction force modelled on reaction forces on the bearings over time after optimization 4

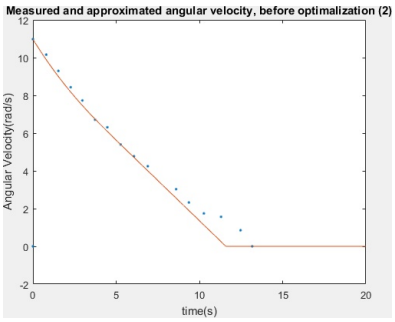
Figure 43: The resulting energy after testing the friction modelled on the reaction forces on the bearings after optimization

B.2 Column

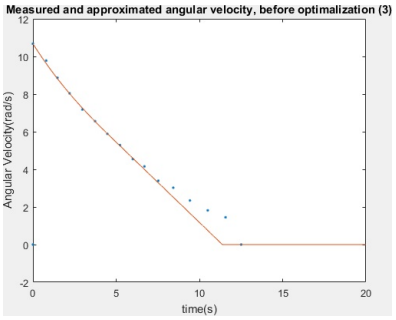
Two test are performed to approximate the bahaviour of the column, and both are optimized. The first with the full force equation the bearings and one simplified with $F_{Column} = \left(C_7 \dot{\theta}^{(1)^2} + C_8\right) n(\dot{\theta}^{(1)})$. The resulting behaviour is plotted with the measured values below.



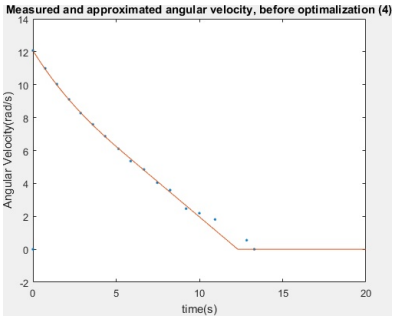
(a) The measured and approximated angular velocity in the column with friction force modelled on reaction forces on the bearings over time before optimization 1



(b) The measured and approximated angular velocity in the column with friction force modelled on reaction forces on the bearings over time before optimization 2

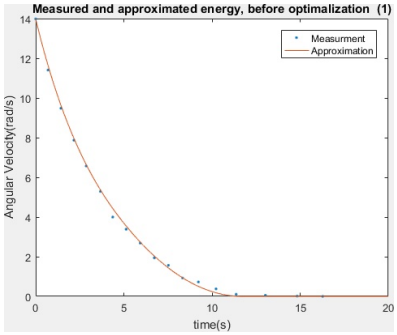


(c) The measured and approximated angular velocity in the column with friction force modelled on reaction forces on the bearings over time before optimization 3

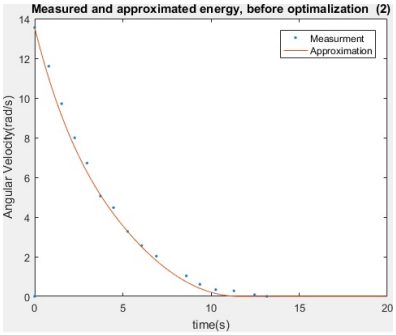


(d) The measured and approximated angular velocity in the column with friction force modelled on reaction forces on the bearings over time before optimization 4

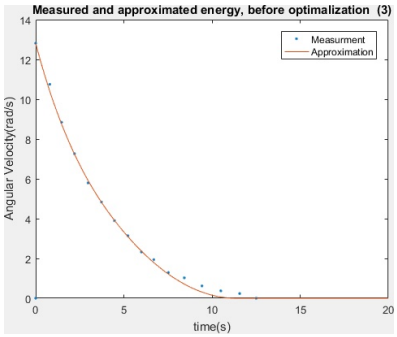
Figure 44: The resulting angular velocity after testing the friction modelled on the reaction forces on the bearings before optimization



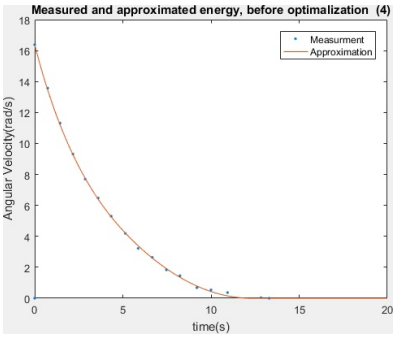
(a) The measured and approximated energy in the column with friction force modelled on reaction forces on the bearings over time before optimization 1



(b) The measured and approximated energy in the column with friction force modelled on reaction forces on the bearings over time before optimization 2

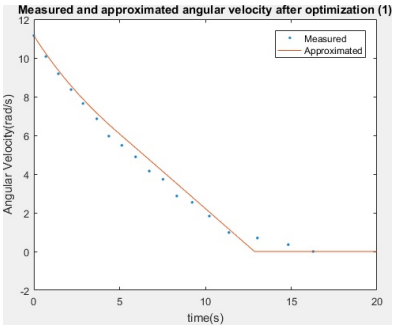


(c) The measured and approximated energy in the column with friction force modelled on reaction forces on the bearings over time before optimization 3

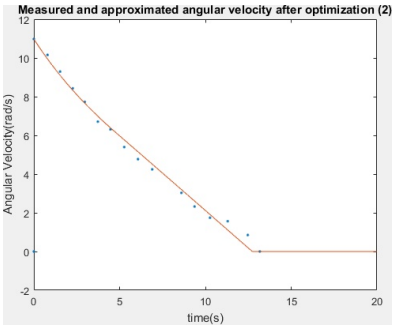


(d) The measured and approximated energy in the column with friction force modelled on reaction forces on the bearings over time before optimization 4

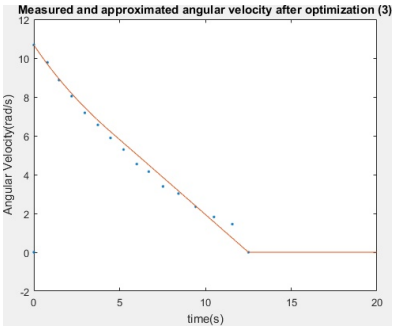
Figure 45: The resulting energy after testing the friction modelled on the reaction forces on the bearings before optimization



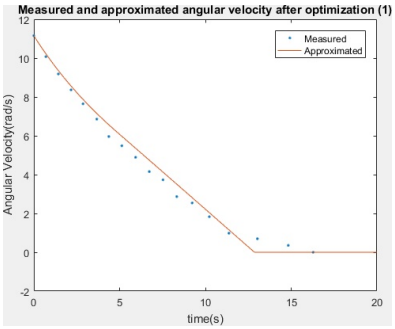
(a) The measured and approximated angular velocity in the column with friction force modelled on reaction forces on the bearings after optimization 1



(b) The measured and approximated angular velocity in the column with friction force modelled on reaction forces on the bearings after optimization 2

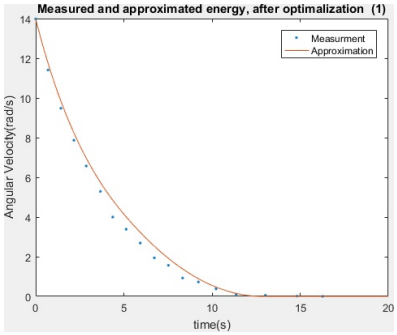


(c) The measured and approximated angular velocity in the column with friction force modelled on reaction forces on the bearings after optimization 3

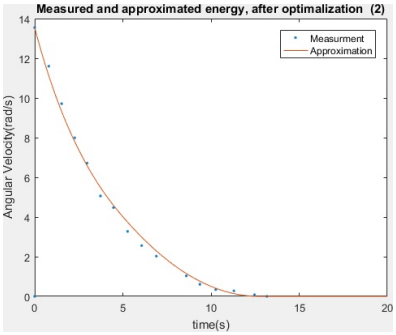


(d) The measured and approximated angular velocity in the column with friction force modelled on reaction forces on the bearings after optimization 4

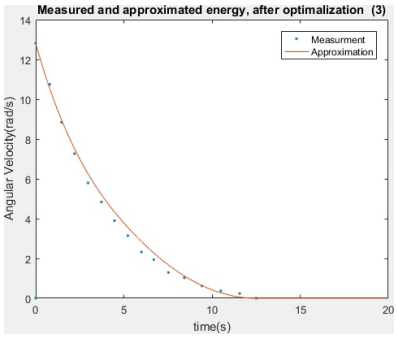
Figure 46: The resulting angular velocity after testing the friction modelled on the reaction forces on the bearings after optimization



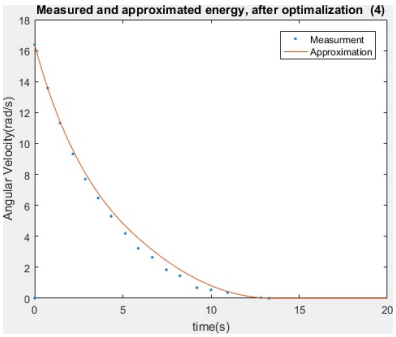
(a) The measured and approximated energy in the column with friction force modelled on reaction forces on the bearings over time after optimization 1



(b) The measured and approximated energy in the column with friction force modelled on reaction forces on the bearings over time after optimization 2

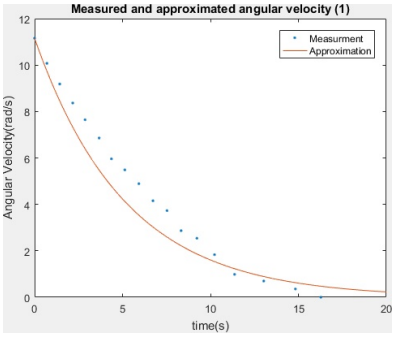


(c) The measured and approximated energy in the column with friction force modelled on reaction forces on the bearings over time after optimization 3

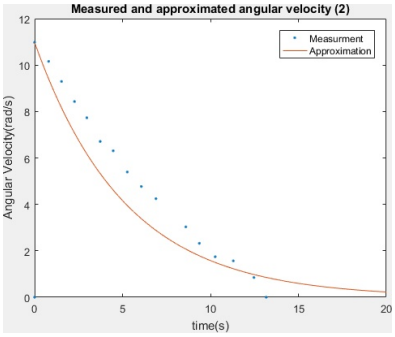


(d) The measured and approximated energy in the column with friction force modelled on reaction forces on the bearings over time after optimization 4

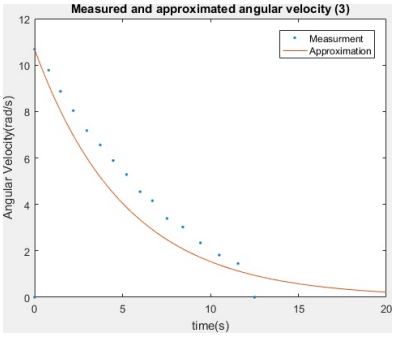
Figure 47: The resulting energy after testing the friction modelled on the reaction forces on the bearings after optimization



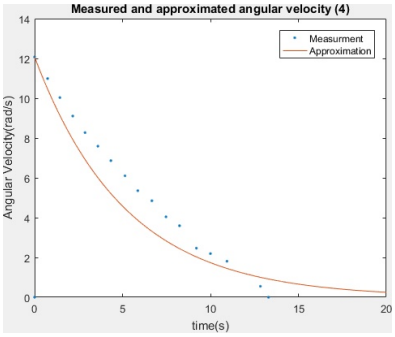
(a) The measured and approximated angular velocity in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ after optimization 1



(b) The measured and approximated angular velocity in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ after optimization 2

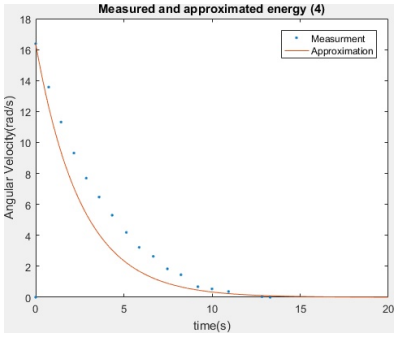


(c) The measured and approximated angular velocity in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ after optimization 3

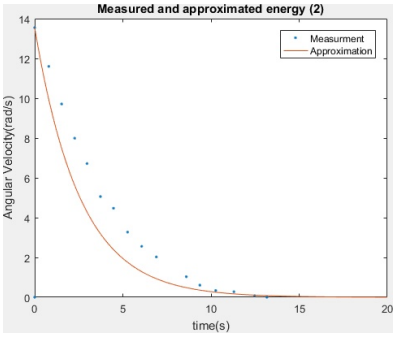


(d) The measured and approximated angular velocity in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ after optimization 4

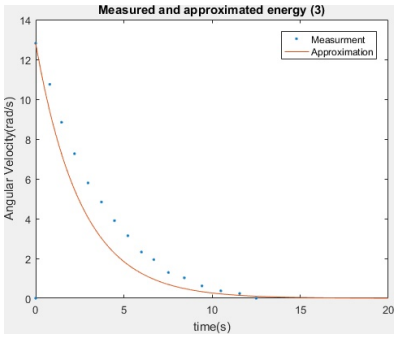
Figure 48: The resulting angular velocity after testing the friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$, before optimization



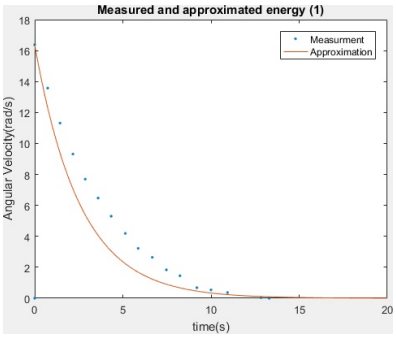
(a) The measured and approximated energy in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ before optimization 1



(b) The measured and approximated energy in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ before optimization 2

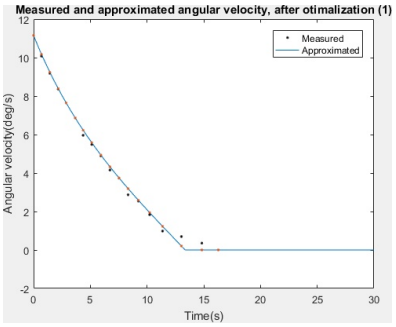


(c) The measured and approximated energy in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ before optimization 3

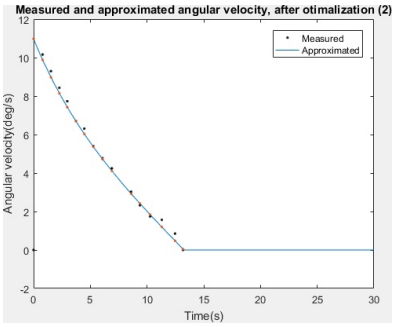


(d) The measured and approximated energy in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ before optimization 4

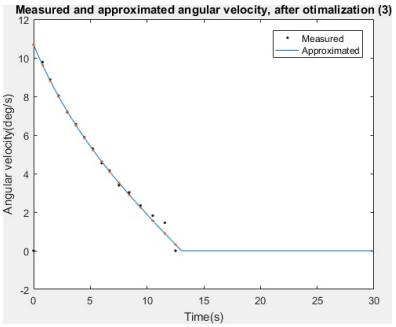
Figure 49: The resulting energy after testing the friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$, before optimization



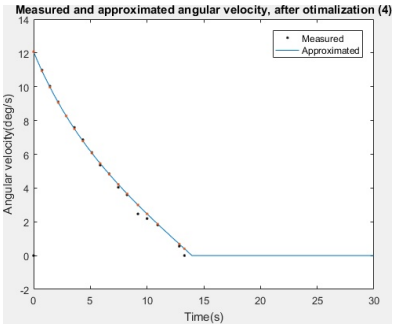
(a) The measured and approximated angular velocity in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ after optimization 1



(b) The measured and approximated angular velocity in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ after optimization 2

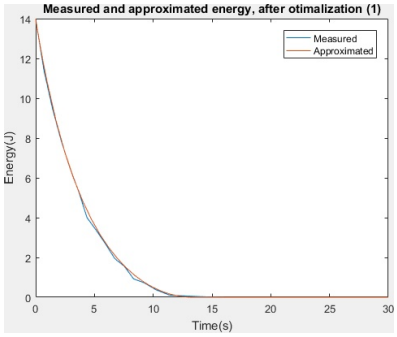


(c) The measured and approximated angular velocity in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ after optimization 3

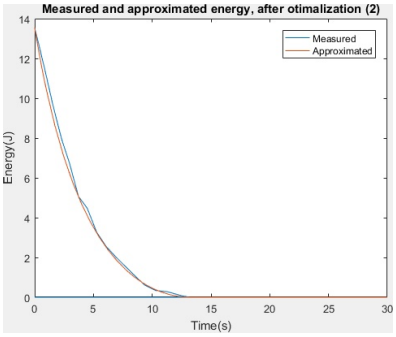


(d) The measured and approximated angular velocity in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ after optimization 4

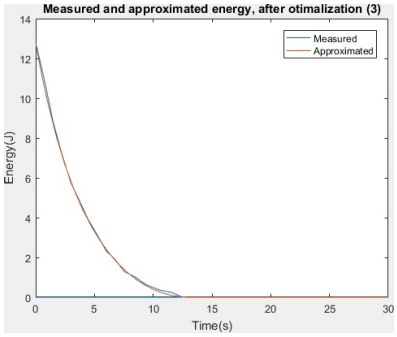
Figure 50: The resulting angular velocity after testing the friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$, after optimization



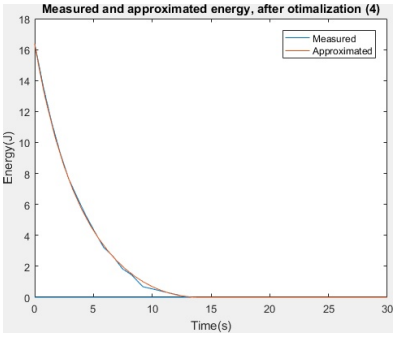
(a) The measured and approximated energy in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ after optimization 1



(b) The measured and approximated energy in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ after optimization 2



(c) The measured and approximated energy in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ after optimization 3



(d) The measured and approximated energy in the column with friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$ after optimization 4

Figure 51: The resulting energy after testing the friction $F_{Column} = \left(C_7 \dot{\theta^{(1)}}^2 + C_8\right) n(\dot{\theta}^{(1)})$, after optimization

C Matlab

C.1 Testing Algorithm

```
%% Chaine with 4 identical links
%Links are moddeled as hollow cylinders
%Gravity constant. Is by default negative.
Gravity=9.81;
%Size of the bodies:
rmin=0.2;
rmaks=0.4;
h=0.1;
%Volume of a link
V=pi*h*(rmaks^2-rmin^2);
%The mass of the bodies.
M=V*7750;%density of steel times volume
Mass=[M,M,M,M,M,M];
%Mass moment of inertia tensors for each body
J1=[Mass(1)*(3*(rmaks^2+rmin^2)+h^2)/12   0           0
0         Mass(1)*(3*(rmaks^2+rmin^2)+h^2)/12    0
0                     0    Mass(1)*((rmaks^2+rmin^2))/2];
J2=[Mass(1)*(3*(rmaks^2+rmin^2)+h^2)/12   0           0
0          Mass(1)*((rmaks^2+rmin^2))/2    0
0                     0     Mass(1)*(3*(rmaks^2+rmin^2)+h^2)/12 ];
J3=J1;
J4=J2;
J5=J1;
Jlist=[J1,J2,J3,J4,J5,J2];
%Vectors from last centre of mass to the point of rotation
PreVList = [ 0,  rmin,    rmin,rmin,rmin,rmin,0
0,    0,      0,   0,0,0,0
0,    0,      0,   0 ,0,0,0];

%Axis of rotation for the frames
Axis     = [ 3, 2 ,3,2,3,2];

%Vector pointing from last point of rotation to next centre of mass
PostVList= [rmaks, rmin ,rmin,rmin  ,rmin ,rmin
```

```
0,    0 ,  0   ,0      ,0,0
0,    0  ,0    ,0,0      ,0 ];
%Initial values to test func file with.
Init=randi(2,18,1);
%% 1 Body
%Create the function giving the angular accelerations of the system
NFrames=1;
tic
MFMNumEquMaker( PreVList,Axis,PostVList,NFrames,Mass,Jlist,Gravity )
toc
%test func file
tic
ddth=func(0,Init);
toc

%Elapsed time is 3.269385 seconds. with simplify
%Func size 79 byte
%To run func one time 0.0014041  seconds.

%Without simlify
% Elapsed time is 2.703553 seconds.
% Elapsed time is 0.002504 seconds.
% Size: 79 byte (79 byte)
%% 2 Bodies
NFrames=2;
%Create the function giving the angular accelerations of the system
tic
MFMNumEquMaker( PreVList,Axis,PostVList,NFrames,Mass,Jlist,Gravity )
toc
%test func file
tic
ddth=func(0,Init);
toc

%Elapsed time is 3.654462 seconds.
%299 byte
%Time used: Elapsed time is 0.002808 seconds.
%without simplify
% Elapsed time is 2.438575 seconds.
% Elapsed time is 0.002712 seconds.
%287 byte (287 byte)
```

```
%% 3 bodies
NFrames=3;
%Create the function giving the angular accelerations of the system
tic
MFMNumEquMaker( PreVList,Axis,PostVList,NFrames,Mass,Jlist,Gravity )
toc
%test func file
tic
ddth=func(0,Init);
toc

%Elapsed time is 12.205891 seconds.
%9,60 kB (9 840 byte)
%Time to evaluate one timestep:Elapsed time is 0.051221 seconds.

% Elapsed time is 5.163580 seconds.
% Elapsed time is 0.076503 seconds.
%11,3 kB (11 593 byte)
%% 4 bodies
NFrames=4;
%Create the function giving the angular accelerations of the system
tic
MFMNumEquMaker( PreVList,Axis,PostVList,NFrames,Mass,Jlist,Gravity )
toc
%test func file
tic
ddth=func(0,Init);
toc

%Without simplify
%Elapsed time is 11.057459 seconds.
%Size: 446 kB (456 841 byte)
%Time to evaluate one timestep :Elapsed time is 2.359062 seconds.
%% 5 bodies
NFrames=5;
%Create the function giving the angular accelerations of the system
tic
MFMNumEquMaker( PreVList,Axis,PostVList,NFrames,Mass,Jlist,Gravity )
toc
%Test func file
tic
```

```
ddth=func(0,Init);
toc

%Elapsed time is 88.447121 seconds.
%SIze: 13,0 MB (13 684 736 byte)
%Time to evaluate: Elapsed time is 14.432317 seconds.
%% 6 bodies
NFrames=6;
%Create the function giving the angular accelerations of the system
tic
MFMNumEquMaker( PreVList,Axis,PostVList,NFrames,Mass,Jlist,Gravity )
toc
%Test func file
tic
ddth=func(0,Init);
toc

%Elapsed time is 1597.289072 seconds.
%Size: 298 MB (313 408 093 byte)
%Time to evaluate one timestep: NA-Out of memory
```

C.2 Validating algorithm manually

C.2.1 Planar Pendulum

```
clc
clear all
%{
This sets up the equation for a 2D pendulum to compare with the symbolic
solver
%}
% syms L1 m1 j1 F1 dTh1 g Th1(t)
% E1=EMatrixMaker([0,0,0],2,[L1,0,0],Th1(t));

%OMG1=simplify(expand(InvE(E1)*diff(E1,t)));
% OMG1=[    0, 0, dTh1,        0;
%           0, 0,    0,        0;
%       -dTh1, 0,    0, -L1*dTh1;
%           0, 0,    0,       0];
%EInert=E1*OMG1(1:4,4)
```

```
% EInert=[-L1*dTh1*sin(Th1(t))
%                       0
%          -L1*dTh1*cos(Th1(t))];
%
syms L1 m1 J11 J15 J19 F1 g dTh1 Th1 real
B=[ -L1*sin(Th1);
0;
-L1*cos(Th1);
0;
1;
0];
dB=[-L1*cos(Th1)*dTh1;
0;
L1*sin(Th1)*dTh1;
0;
0;
0];



M=[  m1     0     0     0     0     0
0     m1     0     0     0     0
0     0     m1     0     0     0
0     0     0     J11     0     0
0     0     0     0     J15     0
0     0     0     0     0     J19];

D =[ 0, 0, 0,     0, 0,    0
0, 0, 0,     0, 0,    0
0, 0, 0,     0, 0,    0
0, 0, 0,     0, 0, dTh1
0, 0, 0,     0, 0,    0
0, 0, 0, -dTh1, 0,    0];
F=[0;0;-g*m1;0;F1;0];

dthV=dTh1;
syms L1 m1 j1 F1 g dTh1 Th1 real
MStar=B'*M*B;
FStar   = B'*F;
NStar   = B'*(M*dB+D*M*B);
```

```
Mstar=simplify(MStar);
NStar=simplify(NStar);
InvMStar= InvMStarMatrixMaker(1,MStar);
ddTh=simplify(expand(InvMStar*(FStar-NStar*dthV)));
%% Algoritm
PreVList = [ 0,   0  ;
0,   0  ;
0,   0  ];
PostVList= [ L1
0
0 ];

Axis=2;
NFrames=1;
IsMassLess=0;
IsPerfBalance=1;
filename='2DPendulum';
ddThAlg=MFMSymEquMaker( PreVList,Axis,PostVList,NFrames,IsMassLess,...
IsPerfBalance,filename );

diff=simplify(expand(ddTh-ddThAlg));
%{
Result from comand window
diff =

0
%}
```

C.2.2 Constructed Pendulum

```
clc
clear all
%Defining the symbols to be used
syms Th1 Th2 dTh1 dTh2 J11 J12 J13 J14 J15 J16 J17 J18 J19 J21 ...
J25 J29 m1 m2 g L11 L12 L13 l1 l3 L21 F1 F2 real

FileName='PendulumOnAStickManualEqu';
%The temporary angular velocities in a vector
dthV=[dTh1;dTh2];
%Make the rotational matrises
```

```
R1=RotMaker(3,Th1);
R21=RotMaker(2,Th2);
%Make the time derivatives of the rotational matrices
dR1=dRotMaker(3,Th1,dTh1);
dR21=dRotMaker(2,Th2,dTh2);
%Defining the angular velocity matrises
omg1M=[   0,-dTh1,0;
dTh1,    0,0;
0,    0,0];
omg21M=[    0,0,dTh2;
0,0,   0;
-dTh2,0,   0];
omg2M=simplify(expand(R21'*omg1M*R21+omg21M));

%The angular velocity vectors
omg1=[0;0;dTh1];
omg2=UnSqewThis(omg2M);

%Defining the position vectors
r1=[L11;L12;L13];%Position of centre of mass 1 from inertial
rp1=[l1;-L12;l3];%Position of the Rotational point between arm and column
r2p=[L21;0;0];%Position of centre of mass 2 from the joint
r2=r1+rp1+R21*r2p;
%% D matrix
D=sym(zeros(12));
D(4:6,4:6)=omg1M;
D(10:12,10:12)=omg2M;
%% Mass matrix
M= sym(zeros(12));
%Mass of the first body
M(1:3,1:3) =[m1,  0,  0;
0, m1,  0;
0,  0, m1];
%Mass of the second body
M(4:6,4:6)=[J11,J12,J13;J14,J15,J16;J17,J18,J19];
%Massmoment of inertia of the first body
M(7:9,7:9)=[m2,  0,  0;
0, m2,  0;
0,  0, m2];
%Massmoment of inertia of the second body
M(10:12,10:12)=[J21,0,0;0,J25,0;0,0,J29];
```

```
%%
%Construct the B matrix
B=sym(zeros(12,2));
B(1:3,1)=-R1*SqewThis(r1)*sym([0;0;1]);
B(4:6,1)=sym([0;0;1]);
B(7:9,1)=-R1*SqewThis(r2)*sym([0;0;1]);
B(7:9,2)=-R1*R21*SqewThis(r2p)*sym([0;1;0]);
B(10:12,1)=R21'*sym([0;0;1]);
B(10:12,2)=sym([0;1;0]);
%% Time derivative of B
dB=sym(zeros(12,2));
dB(1:3,1)=-dR1*SqewThis(r1)*sym([0;0;1]);
dB(7:9,1)=-dR1*SqewThis(r2)*sym([0;0;1])-...
R1*SqewThis(dR21*r2p)*sym([0;0;1]);
dB(7:9,2)=-dR1*R21*SqewThis(r2p)*sym([0;1;0])-...
R1*dR21*SqewThis(r2p)*sym([0;1;0]);
dB(10:12,1)=dR21'*sym([0;0;1]);
%% F vector
F=sym(zeros(12,1));
F(3)=-m1*g;
F(6)=F1;
F(9)=-m2*g;
F(11)=F2;
%% Equation of Motion
MStar=B'*M*B;
FStar   = B'*F;
NStar   = B'*(M*dB+D*M*B);
Mstar=simplify(MStar);
NStar=simplify(NStar);
InvMStar= InvMStarMatrixMaker(2,MStar );
ddTh=simplify(expand(InvMStar*(FStar-NStar*dthV)));
%Create the function, must add symbolic comand manually
%{
syms Th2 dTh1 dTh2 J11 J12 J13 J14 J15 J16 J17 J18 J19 J21 ...
J25 J29 m1 m2 g L11 L12 L13 l1 l3 L21 F1 F2 real
%}
SaveSymEndEqu( ddTh,2,FileName );
%% Evaluate with algorithm
syms L11 L12 L13 L21 l1 l3
%Total number of frames in the system
NFrames=2;
```

```
filename='PendulumOnAStickByCode';
%Vectors from last centre of mass to the point of rotation
PreVList = [ 0, l1,  0  ;
0,  -L12,  0  ;
0,  l3,  0  ];

%Axis of rotation for the frames
Axis     = [ 3,  2];

%Vector pointing from last point of rotation to next centre of mass
PostVList= [ L11,  L21
L12,   0
L13,   0   ];

%Only needed if you want a end equation with only symbols
IsMassLess=logical([0,0]);
IsPerfBalance=logical([0,1]);
%-------------------------------------------------------
%Create the function, must add symbolic comand manually
%{
syms Th2 dTh1 dTh2 J11 J12 J13 J14 J15 J16 J17 J18 J19 J21 ...
J25 J29 m1 m2 g L11 L12 L13 l1 l3 L21 F1 F2 real
%}
MFMSymEquMaker( PreVList,Axis,PostVList,NFrames,IsMassLess,...
IsPerfBalance,filename )
%% Comparison section

Diff=simplify(expand(PendulumOnAStickManualEqu-PendulumOnAStickByCode));

%Result from command window:
%{
Diff =

0
0
0
0

%}
```

Manual equation

```
function  [dY]=PendulumOnAStickManualEqu

syms Th2 dTh1 dTh2 J11 J12 J13 J14 J15 J16 J17 J18 J19 J21 ...
J25 J29 m1 m2 g L11 L12 L13 l1 l3 L21 F1 F2 real

dY=[dTh1;
(F1 - J21*dTh1*dTh2*sin(2*Th2) + J29*dTh1*dTh2*sin(2*Th2) +...
L21^2*dTh1*dTh2*m2*sin(2*Th2) + 2*L21*dTh1*dTh2*l1*m2*sin(Th2) + ...
2*L11*L21*dTh1*dTh2*m2*sin(Th2))/(J19 + J21 + L11^2*m1 + L11^2*m2 +...
L12^2*m1 + l1^2*m2 - J21*cos(Th2)^2 + J29*cos(Th2)^2 + 2*L11*l1*m2 + ...
L21^2*m2*cos(Th2)^2 + 2*L11*L21*m2*cos(Th2) + 2*L21*l1*m2*cos(Th2));
dTh2;
-((J29*dTh1^2*sin(2*Th2))/2 - (J21*dTh1^2*sin(2*Th2))/2 - F2 - ...
L21*g*m2*cos(Th2) + (L21^2*dTh1^2*m2*sin(2*Th2))/2 + ...
L21*dTh1^2*l1*m2*sin(Th2) + L11*L21*dTh1^2*m2*sin(Th2))/(J25 + L21^2*m2)];

end
```

Equation By Algorithm

```
function  [dY]=PendulumOnAStickByCode

syms Th2 dTh1 dTh2 J11 J12 J13 J14 J15 J16 J17 J18 J19 J21 ...
J25 J29 m1 m2 g L11 L12 L13 l1 l3 L21 F1 F2 real

dY=[dTh1;
(F1 - J21*dTh1*dTh2*sin(2*Th2) + J29*dTh1*dTh2*sin(2*Th2) +...
L21^2*dTh1*dTh2*m2*sin(2*Th2) + 2*L21*dTh1*dTh2*l1*m2*sin(Th2) +...
2*L11*L21*dTh1*dTh2*m2*sin(Th2))/(J19 + J21 + L11^2*m1 + L11^2*m2 + ...
L12^2*m1 + l1^2*m2 - J21*cos(Th2)^2 + J29*cos(Th2)^2 + 2*L11*l1*m2 + ...
L21^2*m2*cos(Th2)^2 + 2*L11*L21*m2*cos(Th2) + 2*L21*l1*m2*cos(Th2));
dTh2;
-(J29*dTh1^2*sin(2*Th2) - J21*dTh1^2*sin(2*Th2) - 2*F2 -...
2*L21*g*m2*cos(Th2) + L21^2*dTh1^2*m2*sin(2*Th2) + ...
2*L21*dTh1^2*l1*m2*sin(Th2) + ...
2*L11*L21*dTh1^2*m2*sin(Th2))/(2*m2*L21^2 + 2*J25)];
```

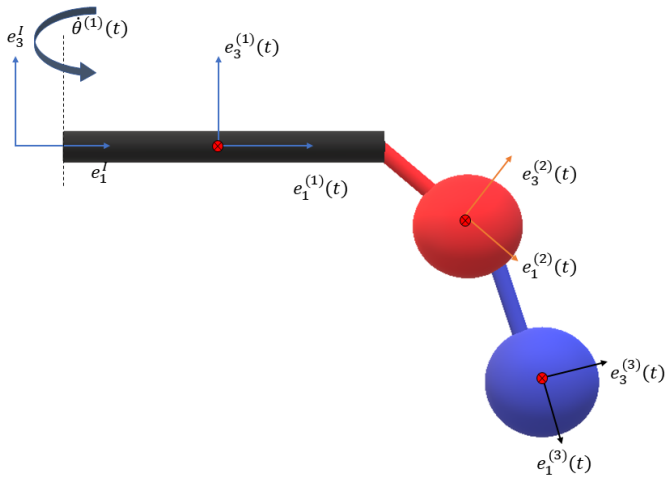


Figure 52: Double pendulum attached at the end of a rod.

```
end
```

C.2.3 Rotated Double Pendulum

```
%Solving the system for a dpuble pendulum rotated around the 3 inertial
%axis. Comparing equations solved manually and by algorithm.

clc
clear all

%Defining symbols
syms L1 L2 L3 dTh1 dTh2 dTh3 m1 m2 m3 J11 J15 J19 J21 J25 J29 J31 J35...
J39 g r1 rp1 r2p r32 R1 R1T R21 R21T R32 R32T dR1 dR21 dR32 omg1...
omg21 omg32  Th1 Th2 Th3  F1 F2 F3 real

NFrames=3;
FileName='DoublePendulumOnAStickByHand';

%Vectors
rr1=[L1,0,0]';
rrp1=[L1,0,0]';
rr2p=[L2,0,0]';
rr32=[L3,0,0]';

%Rotation matrices
```

```
RR1=RotMaker(3,Th1);
RR21=RotMaker(2,Th2);
RR2=RR1*RR21;
RR32=RotMaker(2,Th3);

%Time derivative of the rotational matrices
dRR1=dRotMaker(3,Th1,dTh1);
dRR21=dRotMaker(2,Th2,dTh2);
dRR32=dRotMaker(2,Th3,dTh3);

%% Mass matrix:
%Creating the mass matrix
MH=sym(eye(3*6));
I=sym(eye(3));
MH(1:3,1:3)=I*m1;
MH(7:9,7:9)=I*m2;
MH(13:15,13:15)=I*m3;
J1V=[J11;J15;J19];
J2V=[J21;J25;J29];
J3V=[J31;J35;J39];
MH(4:6,4:6)=diag(J1V,0);
MH(10:12,10:12)=diag(J2V,0);
MH(16:18,16:18)=diag(J3V,0);
E1=[R1,0;0,1]*[1,r1;0,1];
E21=[1,rp1;0,1]*[R21,0;0,1]*[1,r2p;0,1];
E32=[1,0;0,1]*[R32,0;0,1]*[1,r32;0,1];

%% B matrix
%Omg1=[R1T,-R1T*r1;0,1]*[dR1,dR1*r1;0,0]
Omg1=[omg1,omg1*r1;0,0];

%Omg2=[R21T,-R21T*(rp1+R21*r2p);0,1]*Omg1*E21+...
%     [R21T,-R21T*(rp1+R21*r2p);0,1]*[dR21,dR21*r2p;0,0]
Omg2=[omg21 + R21*R21T*omg1,  R21T*omg1*(rp1 + R21*r2p)...
+ R21T*dR21*r2p + R21T*omg1*r1;0,0];
Omg3=[R32T,-R32T*(R32*r32);0,1]*Omg2*E32+...
[R32T,-R32T*(R32*r32);0,1]*[dR32,dR32*r32;0,0];

BH=sym(zeros(18,3));
BH(1:3,1)=-RR1*SqewThis(rr1)*[0,0,1]';
BH(4:6,1)=[0,0,1]';
```

```
BH(7:9,1)=-RR1*SqewThis(rrp1+RR21*rr2p)*[0,0,1]'-RR1*SqewThis(rr1)*[0,0,1]';
BH(7:9,2)=-RR2*SqewThis(rr2p)*[0,1,0]';
BH(10:12,1)=RR21'*[0,0,1]';
BH(10:12,2)=[0,1,0]';
BH(13:15,1)=-RR1*(SqewThis(rrp1+RR21*rr2p)*[0,0,1]'...
+SqewThis(rr1)*[0,0,1]')+RR1*SqewThis([0,0,1])*RR21*RR32*rr32;
BH(13:15,2)=-RR2*SqewThis(rr2p)*[0,1,0]'+...
RR1*RR21*SqewThis([0,1,0])*RR32*rr32;
BH(13:15,3)=-RR1*RR21*RR32*SqewThis(rr32)*[0,1,0]';
BH(16:18,1)=RR32'*RR21'*[0,0,1]';
BH(16:18,2)=RR32'*[0,1,0]';
BH(16:18,3)=[0,1,0]';
BH=simplify(expand(BH));
%% Time derivative of B
dBH=sym(zeros(18,3));
dBH(1:3,1)=-dRR1*SqewThis(rr1)*[0,0,1]';
dBH(7:9,1)=-dRR1*SqewThis(rrp1+RR21*rr2p)*[0,0,1]'-...
RR1*SqewThis(dRR21*rr2p)*[0,0,1]'-dRR1*SqewThis(rr1)*[0,0,1]';
dBH(7:9,2)=-dRR1*RR21*SqewThis(rr2p)*[0,1,0]'-...
RR1*dRR21*SqewThis(rr2p)*[0,1,0]';
dBH(10:12,1)=dRR21'*[0,0,1]';
dBH(13:15,1)=-dRR1*(SqewThis(rrp1+RR21*rr2p)*[0,0,1]'+...
SqewThis(rr1)*[0,0,1]')-RR1*SqewThis(dRR21*rr2p)*[0,0,1]'+...
dRR1*SqewThis([0,0,1])*RR21*RR32*rr32+...
RR1*SqewThis([0,0,1])*dRR21*RR32*rr32+...
RR1*SqewThis([0,0,1])*RR21*dRR32*rr32;
dBH(13:15,2)=-dRR1*RR21*SqewThis(rr2p)*[0,1,0]'-...
RR1*dRR21*SqewThis(rr2p)*[0,1,0]'+...
dRR1*RR21*SqewThis([0,1,0])*RR32*rr32+...
RR1*dRR21*SqewThis([0,1,0])*RR32*rr32+...
RR1*RR21*SqewThis([0,1,0])*dRR32*rr32;
dBH(13:15,3)=-dRR1*RR21*RR32*SqewThis(rr32)*[0,1,0]'-...
RR1*dRR21*RR32*SqewThis(rr32)*[0,1,0]'-...
RR1*RR21*dRR32*SqewThis(rr32)*[0,1,0]';
dBH(16:18,1)=dRR32'*RR21'*[0,0,1]'+RR32'*dRR21'*[0,0,1]';
dBH(16:18,2)=dRR32'*[0,1,0]';
dBH=simplify(expand(dBH));
%%
omg1=simplify(RR1'*dRR1);
omg2=simplify(RR21'*omg1*RR21+RR21'*dRR21);
omg3=simplify(RR32'*omg2*RR32+RR32'*dRR32);
```

```
%D matrix
DH=sym(zeros(18));
DH(4:6,4:6)=omg1;
DH(10:12,10:12)=omg2;
DH(16:18,16:18)=omg3;

%F vector
FH=sym(zeros(3*6,1));
FH(3,1)=-m1*g;
FH(9,1)=-m2*g;
FH(15,1)=-m3*g;
FH(5,1)=-F2+F3;
FH(6,1)=F1;
FH(11,1)=F2-F3;
FH(17,1)=F3;

%%
MstarH=BH'*MH*BH;
NstarH=BH'*(MH*dBH+DH*MH*BH);
FstarH=BH'*FH;
%%
for i=1:sum(length(MstarH))
MstarH(i)=simplify(MstarH(i));
NstarH(i)=simplify(NstarH(i));
end
InvMstarH=InvMStarMatrixMaker(NFrames,MstarH);

parfor i=1:9
simplify(InvMstarH(i));
end
%%
dThVector=[dTh1;dTh2;dTh3];

EndEq=InvMstarH*(FstarH-NstarH*dThVector);
for i=1:3
EndEq(i)= simplify(EndEq(i));
end
%%
%% Algorithm
%Function creator section:
```

```
%Total number of frames in the system
NFrames=3;

filename='DobblePendulumOnAStickByCode';
%Vectors from last centre of mass to the point of rotation
PreVList = [ 0, L1,  0,  0;
0,  0,  0,  0;
0,  0,  0,  0];

%Axis of rotation for the frames
Axis     = [ 3,  2,  2];

%Vector pointing from last point of rotation to next centre of mass
PostVList= [ L1,  L2,  L3
0,   0,   0
0,   0,   0];


IsMassLess=logical([0,0,0]);
IsPerfBalance=logical([1,1,1]);
%-------------------------------------------------------
ddThAlg=MFMSymEquMaker( PreVList,Axis,PostVList,NFrames,IsMassLess,...
IsPerfBalance,filename );

%% Comparing

Diff=simplify(expand(EndEq-ddThAlg));
% Output from command window.
Diff =

0
0
0
```

C.3 Optimization Function

C.3.1 Main Optimization

This script optimizes the friction functions stated in the funcDiff.mat using lsqnonlin and plots the results.

Contents

- Equation of Motion
- Measured energy calculation
- Optimization section
- Integrating section
- Plot results
- Calculate error

```
clc
clear all
%Read measurments from file
[NumM,~,~]=xlsread('MainBehToBUsed.xls');
%Set correct 0 point and change to radians, as the measuring tool was
%fitted 180deg wrong.
NumM(:,2)=(NumM(:,2)+7+180)*2*pi/360;
%Change RPM to rad/s
NumM(:,1)=NumM(:,1)*2*pi/60;
%Set solution to a matrix to facilitate energy calculation
MeSolM=[zeros(1,size(NumM,1));NumM(:,1)';NumM(:,2)';zeros(1,size(NumM,1))];
```

Equation of Motion

```
% How many basises there is
NFrames=2;

%Gravity constant in m/s^2
g=9.81;

% Mass of the bodies in kg
Mass=[3.3645051,4.6811];

%Mass moment of inertia of the bodies in kg*m^2
```

```
J1=[0.0832128030   0.00767990280  -0.00353398330
    0.0076799028   0.11564199000   0.00059466628
   -0.0035339833   0.00059466628   0.06172657200 ];

J2=[4.1857*10^(-3)     0        0
    0      1.9213*10^(-2)       0
    0           0       1.97783*10^(-2)];

Jlist=[J1,J2];

%Vectors from last centre of mass to the point of rotation in meters
PreVList = [ 0,     0.13294558,      0
             0,  1.9696113e-02,      0
             0,    -0.00897338,      0];

%Axis of rotation of the bodies
Axis     = [ 3, 2 ];

%Vector pointing from last point of rotation to next centre of mass
%in meters
PostVList= [ 1.1705442e-01,       0.12675638
            -1.9696113e-02,           0
             3.0102662e-01,           0];

% Create the equation of motion and save it to a file called func.mat
MFMNumEquMaker( PreVList,Axis,PostVList,NFrames,Mass,Jlist,g )
```

Measured energy calculation

```
%Lowest point the centre of masses can reach in meters, for energy
%calculation
LP=[0.30102662,     0.16529686];

[ MeKinEn,MePotEn,MeEnergy] = EnergyFunc( MeSolM,NFrames,Jlist,Mass,g,LP);
```

Optimization section

```
%{
Constants found by the other optimalizations, and manually adjusted to
save time in the actual optimalization.
```

```
%}
%Constants for the analysis with full force
% X1=[  0.001523114066306, 0.001282456249998];
% X2=[  0.001592442688858, 0.000394592000684];
%Constants for analysis for simplified force
X1=[  0.000301026806606,  0.001303840754625,  0.000185026806606 ];
X2=[  0.00023091557841332,   0.000309473598742528,  0.00156091557841332];
K=[X1,X2];
%Set options for lsqnonlin
optionslsq=optimoptions('lsqnonlin','StepTolerance',10e-8,...
                        'MaxIterations',5000);
%Run the optimalization, constants must be positive.
X=lsqnonlin(@OptiMain,K,[0,0,0,0,0,0],[],optionslsq);
```

Integrating section

```
%Set out of bound values in measurments
SafeNumA=size(NumM,1);
SafeNumV=41;
%Initial Step size for the integration.
h=  (2.5000e-04);
% Start and end time.
TimeSpan=[0,60];
%Set initial values
InitVal=[0,NumM(1,1),NumM(1,2),0,0,0];
acc=funcDiff(0,InitVal,K);
InitVal(1,5:6)=acc(5:6,1)';
%Set options for ode45, define event function
options=odeset('Events',@MainEvent,'InitialStep',h);
%Solve the equations of motion
[T45,SolM45,T45e,Y45e,~]=ode45(@funcDiff,TimeSpan,InitVal',options,X);
%Calculate enery for each timestep
[ ApKinEn,ApPotEn,ApEnergy] = EnergyFunc( SolM45',NFrames,Jlist,Mass,g,LP);
```

Plot results

```
figure(1)
plot(T45(),SolM45(:,2),NumM(1:SafeNumV,3),NumM(1:SafeNumV,1),'.')
title('Measured and approximated Angular velocity 1')
ylabel('Angular velocity(rad/s)')
```

```
xlabel('Time(s)')
legend('Approximated','Measured')


figure(2)
plot(T45,SolM45(:,3),NumM(:,3),NumM(:,2),'.')
title('Measured and approximated Angle 2')
ylabel('Anlge(rad)')
xlabel('Time(s)')
legend('Approximated','Measured')

figure(3)
plot(NumM(:,3),MeEnergy,T45,ApEnergy)
title('Measured and approximated energy')
ylabel('Energy(J)')
xlabel('Time(s)')
legend('Measured','Approximated')

ErrT=T45e(2:SafeNumA)-NumM(2:end,3);
AvgTerr=sum(abs(ErrT))/numel(ErrT);
figure(4)
plot(ErrT)
title('Time Error')
ylabel('Error(s)')
legend('Time Error')
xlabel('Measurment nr')

ErrAngV=Y45e(2:SafeNumV,2)-NumM(2:SafeNumV,1);
AvgAngVErr=sum(abs(ErrAngV))/numel(ErrAngV);
figure(5)
plot(abs(ErrAngV))
title('Angular Velocity 1 Error')
ylabel('Error(rad/s)')
legend('Angular Velocity Error')
xlabel('Measurment nr')
ErrAng=Y45e(2:SafeNumA,3)-NumM(2:end,2);
ErrAngErr=sum(abs(ErrAng))/numel(ErrAng);
figure(6)
plot(abs(ErrAng))
title('Angle 2 Error')
ylabel('Error(rad)')
```

```
legend('Angle Error')
xlabel('Measurment nr')

%-------------------------------------------------------------------------
```

C.3.2 OptiMain

```
function [Error]= OptiMain(K)
%This function takes in the friction konstants and returns the error.

%Read measured values for error calculation
[NumM,~,~]=xlsread('MainBehToBUsed.xls');
NumM(:,2)=(NumM(:,2)+7+180)*2*pi/360;
NumM(:,1)=NumM(:,1)*2*pi/60;
%Out of bounds values
SafeNumA=size(NumM,1);
SafeNumV=42;%This number is 'the Answer' by coincidence, not design.
%Set a known value for End, used as a sequrity chech.
End=[9001,9001];
%Pre allocate error matrix
Error=zeros(SafeNumA,5);
%Initial Step size for the integration.
h=  (2.5000e-04);
% Start and end time.
TimeSpan=[0,60];
%Wheight for error norm
Weight=10;
%Set initial values
InitVal=[0,NumM(1,1),NumM(1,2),0,0,0];
acc=funcDiff(0,InitVal,K);
InitVal(1,5:6)=acc(5:6,1)';
%Set options for ode45, set event function
options=odeset('Events',@MainEvent,'InitialStep',h);
[~,~,T45e,Y45e,~]=ode45(@funcDiff,TimeSpan,InitVal',options,K);
```

Calculate error

```
%Find 'end point', where the pendulum should have stopped.
for i=1:numel(T45e)
   if  Y45e(i,3)<pi/2+0.01 && Y45e(i,3)>pi/2-0.01
```

```
        End=[T45e(i),Y45e(i,3)];
        break
    end
end
%If the pendulum did not stop within the time span given to ode45
if End(1)==9001
End=[T45e(end),Y45e(end,3)];
end
%Calculate error
Error(1:SafeNumV,1)=(Y45e(1:SafeNumV,2)-NumM(1:SafeNumV,1));
Error(1:SafeNumV,2)=(T45e(1:SafeNumV,1)-NumM(1:SafeNumV,3));
Error(1:SafeNumA,3)=(Y45e(1:SafeNumA,3)-NumM(1:SafeNumA,2))*Weight;
Error(1:SafeNumA,4)=T45e(1:SafeNumA,1)-NumM(1:SafeNumA,3);
Error(1,5)=(End(1)-NumM(122,3))*Weight;
end
```

C.3.3 MainEvent

```
function [ value,isterminal,direction ] = MainEvent(t,Y,K )
%This function takes the result from ode45 and returns the angular velocity o

value=[Y(4),0];
isterminal=0;
direction=0;

end
```

C.4 Algorithm for the equation of motion

C.4.1 MFMNumEquMaker

```
function MFMNumEquMaker(PreVList,Axis,PostVList,NFrames,Mass,Jlist,Gravity)
%{
Moving Frame Numeric Equation Maker creates the equation of motion for a
system of linked rigid bodies that is ready to be used in a numerical
integrater. The equation of motion is saved to a file in the folder
TempFiles with the name func.
%}

%{
```

```
Input:
PreVList      - The list of vectors pointing to the next point of rotation
                from the previous frame

Axis          - The directions of the rotational axises for each frame

PostVList     - The list of vectors pointing to the next frame from the
                point of rotation

NFrames       - The total number of frames

Mass          - The mass of the bodies

Jlist         - Matrix containing the mass moment of inertia of the bodies

Gravity       - Gravity constant
%}
%Create the needed symbols
[ SymThV,SymdThV,MomList,~,~] = SymVMaker( NFrames );
%------------------------------------------------------------------------

% D matrix, B matrix and the time derivative of B
[ D, B, dB ]=D_B_dB_Maker(NFrames, SymThV,SymdThV,Axis,PreVList,PostVList);
%------------------------------------------------------------------------
%Create the F vector
F  = Fmaker( Axis,NFrames,MomList,Mass,Gravity );
%Save the velocities as a function, it is needed for energy analysis.
Velocities=B*SymdThV;
VelocitySaver( Velocities, NFrames );
%------------------------------------------------------------------------
%Save the functions giving the positions of the centre of mass. Needed for
%energy analysis
PosVectorFunctionMaker( NFrames,PreVList,PostVList,Axis )
%------------------------------------------------------------------------

%Create the mass matrix
MassMatrix  = NumMassMatrixMaker( NFrames,Mass,Jlist );
%------------------------------------------------------------------------

%Evaluate the F*, M* and N* matrices
FStar   = B'*F;
```

```
MStar   = B'*MassMatrix*B;
NStar   = B'*(MassMatrix*dB+D*MassMatrix*B);
%--------------------------------------------------------------------------

%Simplify M* and N*. And show a progress message
parfor i=1:numel(MStar)
    disp(strcat('Simplifying M and N number',num2str(i)))
MStar(i)=simplify(MStar(i));
NStar(i)=simplify(NStar(i));
end
%Create and simplify the inverse of M*
InvMStar=InvMStarMatrixMaker(NFrames,MStar );
for i=1:numel(InvMStar)
    disp(strcat('Simplifying M^(-1) number',num2str(i)))
   simplify(InvMStar(i));
end

%--------------------------------------------------------------------------

%Evaluate the equation of motion
ddTh=InvMStar*(FStar-NStar*SymdThV);
%--------------------------------------------------------------------------

%Simplify the equation of motion, if the number of frames is below 4.
if NFrames<5
    for i=1:numel(ddTh)
         disp(strcat('Simplifying EndEqu number',num2str(i)))
    ddTh(i)=simplify((ddTh(i)));
    end
end
%--------------------------------------------------------------------------

%Saving the end equation to a .m file with the name stated by filename.
disp('Saving EndEqu')
SaveNumEndEqu( ddTh,NFrames );
%--------------------------------------------------------------------------
end
```

C.4.2 SymVMaker

```
function [ SymThV,SymdThV,MomList,MassList,Gravity ] = SymVMaker( NFrames )
%SymVMaker
% This function creates the neccesary symbols that the functions
%MFMNumEquMaker and MFMSymEquMaker needs to construct the equations of
%motion

%Gravity symbol
syms g
Gravity=g;
%Pre allocate
SymThV  = sym(zeros(NFrames,1));
SymdThV = SymThV;
MomList=SymThV;
MassList=SymThV;
%Create the angle, angular velocity, mass and moment symbols.
for i=1:NFrames

    % Theta

    Theta = strcat(' Th',num2str(i));

    String =strcat('syms',Theta,' real');

    eval(String)

    SymThV(i,1) = eval(Theta);

    % ------------------------------------------------------------------
    % dTheta

    dTheta = strcat(' dTh',num2str(i));

    String =strcat('syms',dTheta,' real');

    eval(String)

    SymdThV(i,1) = eval(dTheta);

    % ------------------------------------------------------------------
```

```
    % Momets

    F=strcat(' F',num2str(i));
    String=strcat('syms',F,' real');
    eval(String)
    MomList(i)=eval(F);

    %--------------------------------------------------------------------

    % Mass

    M=strcat(' m',num2str(i));
    String=strcat('syms',M,' real');
    eval(String)
    MassList(i)=eval(M);

end
end
```

C.4.3 D-B-dB-Maker

Contents

- Pre allocating
- Angular parts of B
- dB and Linear parts of B
- J loop
- The D matrix
- Simplify

```
function [D, B, dB] =D_B_dB_Maker(NFrames,SymThV,SymdThV,Axis,PreVList,PostVL

%{
D_B_dB_Maker
This function takes the physical definition of the multibody system as
input and constructs the D, B and dB matrices.

Input:
       NFrames  : Number of bodies
       SymThV   : Vector containing the Symbols of the angles in the system
```

```
       SymdThV  : Vector containing the symbols for the angular velocities
       Axis     : Vector containing the rotational axis for the rotations
       PreVlist : Matrix with the vectors pointing from the last
                  centre of mas to the next point of rotation
       PostVList: Matrix containing the vectors from the last point of
                  rotation to the next centre of mass

Output:
       D : Matrix containing the angular velocity matrises every second 3X3
           position alng the diagonal. Starting with [4:6,4:6]
       B : Matrix containing the factorized velocities of the system
       dB: Time dreivative of the B matrix.
%}
```

Pre allocating

```
B=sym(zeros(6*NFrames,NFrames));
dB=B;
D=sym(zeros(6*NFrames));
OmgVList=sym(zeros(3,NFrames));
LocalOmegaVList=OmgVList;
R=sym(zeros(3,3*NFrames));
OmegaMatrix=R;
dAbsR=R;
dR=R;
RListfB=sym(zeros(3,NFrames*3));
dRListfB=RListfB;
```

Angular parts of B

```
SymOne=sym(1);

for i=1:NFrames
%Counter for the different rotation matrix lists
Rc=-2+3*i:3*i;
%-------------------------------------------------------------------------
R(1:3,Rc) =RotMaker(Axis(i),SymThV(i));
[dR(1:3,Rc)]=dRotMaker( Axis(i), SymThV(i) , SymdThV(i) );
%-------------------------------------------------------------------------
%Ready a list of the correct local omegas for both the B matrix and the D
```

```
%Insert the moments into the Force vector
switch Axis(i)

    case{1}
        OmgVList(1,i)=SymOne;
        LocalOmegaVList(1,i)=SymdThV(i);


    case{2}
        OmgVList(2,i)=SymOne;
        LocalOmegaVList(2,i)=SymdThV(i);


    case{3}
        OmgVList(3,i)=SymOne;
        LocalOmegaVList(3,i)=SymdThV(i);

end

%-------------------------------------------------------------------------
%Calculate the omegas and put them into the B matrix

OmegaMatrix(1:3,Rc)=SqewThis(OmgVList(1:3,i));
%Vertical B counter
VBc=-2+i*6:6*i;
%Insert the last local omega vector
    B(VBc,i)=OmgVList(1:3,i);
for j=1:i-1
%Pull out the omega vector above, update and insert
    B(VBc,j)=R(1:3,Rc)'*B(VBc-6,j);
end
end
%-------------------------------------------------------------------------
%Simplify the B matrix before further use
B=simplify(expand(B));
```

dB and Linear parts of B

```
%Absolute rotation matrix, updated for each frame
AbsR=R(1:3,1:3);
```

```
%Time derivative of the absolute rotation matrix, updated for each frame
dAbsR(1:3,1:3)=dR(1:3,1:3);
%-------------------------------------------------------------------------
%First linear input to the B matrix, omegatilda multiplied with the post
%position vector and given in the inertial frame by the absolute rotation
B(1:3,1)=AbsR*OmegaMatrix(1:3,1:3)*PostVList(1:3,1);
%Time derivative of the first input. Note that omega does not hold theta
%dot, and only the rotation matrix is a function of time
dB(1:3,1)=dAbsR(1:3,1:3)*OmegaMatrix(1:3,1:3)*PostVList(1:3,1);
%-------------------------------------------------------------------------
%First update of the rotation matrises and the time derivative of them
%Used to calculate the linear velocities
RListfB(1:3,1:3)=R(1:3,1:3)*OmegaMatrix(1:3,1:3);
dRListfB(1:3,1:3)=dR(1:3,1:3)*OmegaMatrix(1:3,1:3);
%-------------------------------------------------------------------------
for i=2:NFrames

Ric=i*3-2:i*3; %Counter dependent on i for the rotation matrix lists below

%Update the time derivative of the absolute rotation matrix for the alpha
%frame
   dAbsR(1:3,Ric)=dAbsR(1:3,Ric-3)*R(1:3,Ric)  +  AbsR*dR(1:3,Ric);
%Update the absolute rotation matrix for alpha+1 frame
   AbsR = AbsR*R(1:3,Ric);
%-------------------------------------------------------------------------
LBc=i*6-5:6*i-3;%Counter for the linear part of B and dB
ABc=-2+i*6:6*i; %Counter for the angular part of B and dB
%B and dB section

%The last terms of the linear velocities calculated using the local omegas
   B(LBc,i)=AbsR*OmegaMatrix(1:3,Ric)*PostVList(1:3,i);
%The last terms of the time derivative of the linear parts of the B matrix
   dB(LBc,i)=dAbsR(1:3,Ric)*OmegaMatrix(1:3,Ric)*PostVList(1:3,i);
%-------------------------------------------------------------------------
```

J loop

```
for j=1:i-1

Rjc=j*3-2:j*3;%Counter dependent on j for the rotation matrix lists below
```

```
%The linear velocities calculated and put into the B matrix
   B(LBc,j)=B(LBc-6,j)+RListfB(1:3,Rjc)*...
            (PreVList(1:3,i)+R(1:3,Ric)*PostVList(1:3,i));

%Time derivative of the angular part of the B matrix calculated.
   dB(ABc,j)=dR(1:3,Ric)'*B(ABc-6,j) + R(1:3,Ric)'*dB(ABc-6,j);

%The derivative of the linear parts of the B matrix calculated and inserted
%into dB
   dB(LBc,j)=dB(LBc-6,j)+dRListfB(1:3,Rjc)*(PreVList(1:3,i)+...
              R(1:3,Ric)*PostVList(1:3,i))+...
               RListfB(1:3,Rjc)*(dR(1:3,Ric)*PostVList(1:3,i));

%Update the rotational and time derivative of the rotatons for the linear
%parts of B and dB.
dRListfB(1:3,Rjc)=dRListfB(1:3,Rjc)*R(1:3,Ric)+...
                      RListfB(1:3,Rjc)*dR(1:3,Ric);

RListfB(1:3,Rjc) =RListfB(1:3,Rjc)*R(1:3,Ric);

end
%The i'th input calculated as a special case for each frame
RListfB(1:3,Ric)=AbsR*OmegaMatrix(1:3,Ric);
dRListfB(1:3,Ric)=dAbsR(1:3,Ric)*OmegaMatrix(1:3,Ric);

end
```

The D matrix

```
OMG=LocalOmegaVList(1:3,1);
D(4:6,4:6)=SqewThis(OMG);
for i=2:NFrames
%Counters
Rc=i*3-2:i*3;
Dc=i*6-2:i*6;
%---------------------------------------
%Calculate the omega and insert into D
OMG=R(1:3,Rc)'*OMG+LocalOmegaVList(1:3,i);
D(Dc,Dc)=SqewThis(OMG);
end
```

Simplify

```
%Simplify the matrises before output
D=simplify(expand(D));
B=simplify(expand(B));
dB=simplify(expand(dB));
end
```

C.4.4 Fmaker

```
function [ F ] = Fmaker( Axis,NFrames,MomList,MassList,g )
%Fmaker This function constructs the force vector for the MFMEquMaker
%functions

%Pre allocate
F=sym(zeros(NFrames*6,1));
FC=6;

%Set the forces and moments
for i=1:NFrames
    F(i*6-3,1)=-g*MassList(i);
    switch Axis(i)

    case{1}
          F(FC-2,1)=MomList(i);
    case{2}
          F(FC-1,1)=MomList(i);
    case{3}
          F(FC,1)=MomList(i);
    end
    FC=FC+6;

end
% Set the driving moments to the previous bodies, with oposite sign.
FC=6*NFrames-6;
for i=1:NFrames-1
    F(FC-2:FC,1)= F(FC-2:FC,1)- F((FC+6)-2:(FC+6),1);
    FC=FC-6;
end
end
```

C.4.5 NumMassMatrixMaker

```
function [ MassMatrix ] = NumMassMatrixMaker( NFrames,Mass,Jlist )
%NumMassMatrixMaker takes the size of the system, the mass and mass moment
%of inertia for each body and setts them to the correct positions in the
%mass matrix.

%Pre allocating
MassMatrix=zeros(NFrames*6);
I=eye(3);

%Construct the mass matrix:
for i=1:NFrames

Mc=i*6-5:i*6-3;
Jc=i*6-2:i*6;
Jlistc=i*3-2:i*3;
MassMatrix(Mc,Mc)=I*Mass(i);
MassMatrix(Jc,Jc)=Jlist(1:3,Jlistc);

end
end
```

C.4.6 InvMStarMatrixMaker

```
function [ InvMStarMatrix ] = InvMStarMatrixMaker( TotalNumberOfFrames,MStarM
%{
%Note:
%MstarMatrix is used in a eval function, but matlab does not recognize this.
%}

%This function takes the Mstar matrix as imput and inverts it using a
%general matrix of the same size

% Preallocate
SymMatrix = sym(zeros(TotalNumberOfFrames));
%-----------------------------------

% Create Generic symbolic matrix
 Counter = 0;
for i = 1:TotalNumberOfFrames
```

```
    for j = 1:TotalNumberOfFrames

        Counter = Counter+1;

        Symbol = strcat(' Symb',num2str(Counter));

        String =strcat('syms',Symbol);

        eval(String)

        SymMatrix(i,j) = eval(Symbol);

    end
end
%-------------------------------------------------

% Create determinant and inverse of generic symbol matrix
DetSym = det(SymMatrix);

InvSymMatrix = inv(SymMatrix);

%-------------------------------------------------

% Create inverted generic matrix multiplied with the determinant
InvSymMatrixDetMultiplied = InvSymMatrix*DetSym;
%-----------------------------------------------------------------

% Preallocate
StringCell = cell(TotalNumberOfFrames);
%-----------------------------------------------

% Create command strings
parfor i = 1:TotalNumberOfFrames

for j = 1:TotalNumberOfFrames

   String = char(SymMatrix(i,j));

   String2 = strcat('MStarMatrix(',num2str(i),',',num2str(j),')');
```

```
   StringCell{i,j} = strcat(String,'=',String2,';');

end
end
% ------------------------------------------------------------

% Evaluate command strings
for i = 1:TotalNumberOfFrames
for j = 1:TotalNumberOfFrames

   eval(StringCell{i,j});

end
end
% ------------------------------------------------------------

% Evaluate Matrices
InvMStarDetMultiplied = eval(InvSymMatrixDetMultiplied);
DetSym = eval(DetSym);
%-------------------------------------

% Return to inverse form by dividing by previously multiplied determinant
DetSymDivided =1/DetSym;

InvMStarMatrix = InvMStarDetMultiplied*DetSymDivided;
%-------------------------------------------------------------------------



end
```

C.4.7 SaveNumEndEqu

```
function SaveNumEndEqu( ddTh,NFrames )
%SaveSymEndEqu: This function takes the equation of motion created by
%MFMNumEquationMaker and saves it to a file witht the name func.mat
%The equations are saved on the form as a vector:
%{
dY=[dY1;ddY1,dY2;ddY2;...DYn;ddYn]
%}
```

```
%Create header
Header='function  [dY]=func(t,Y)' ;
%Create the file
fid=fopen('C:\Users\Thorstein\Dropbox\Master\Master\Matlab\TempFiles\func.m','wt');
%Save the header
fprintf(fid,'%s \n\n',Header);

%Save the equations of motion
Yc=2;
fprintf(fid,'%s','dY=[');
for i=1:NFrames
    %Save the velocity from input
    fprintf(fid,'%s;\n',strcat('Y(',num2str(Yc),')'));
    Yc=Yc+2;
%Prepare acceleration string
    ddThStr=char(ddTh(i));
    InitdThc=2;
    InitThc=1;
%Replace Symbols with input name
    for j=1:NFrames

ddThStr=regexprep(ddThStr,strcat('dTh',num2str(j)),...
    strcat('Y(',num2str(InitdThc),')'));
InitdThc=InitdThc+2;

ddThStr=regexprep(ddThStr,strcat('Th',num2str(j)),...
    strcat('Y(',num2str(InitThc),')'));
InitThc=InitThc+2;

ddThStr=regexprep(ddThStr,strcat('F',num2str(j)),...
    strcat('Y(',num2str(j+NFrames*2),')'));

    end
    %Save acceleration string
    fprintf(fid,'%s;\n',ddThStr);

end

%Last moment input strings
for i=1:NFrames-1
    fprintf(fid,'%s;\n',strcat('Y(',num2str(i+NFrames*2),')'));
```

```
end
fprintf(fid,'%s',strcat('Y(',num2str(NFrames*2+NFrames),')'));
fprintf(fid,'%s\n\n',']';','end');
fclose(fid);
end
```

C.4.8 func.mat

This function is generated by the function MFMNumEquMaker. It takes the position and velocity as input and returns the acceleration and velocity.

```
function  [dY]=func(t,Y)

dY=[Y(2);
(365375409332725729550921208179070754913983135744000 0*Y(5) + ...
10839944680892622445727729358419045242440469866086 40*Y(2)*Y(4)*sin(Y(3))...
+ 6635559110113107502625917418796637386386914903326 72*Y(2)*Y(4)...
*cos(Y(3))*sin(Y(3)))/...
(10839944680892622445727729358419045242440469866086 40*cos(Y(3))...
+ 3317779555056553751312958709398318693193457451663 36*cos(Y(3))^2 ...
+ 1483006911929803967409527443548496397633963394237775);
Y(4);
(20282409603651670423947251286016 0000*Y(6))...
/191516806205625713911229573624419 79 ...
+ (11806095803152574104161318652656 00000*...
cos(Y(3)))/191516806205625713911229573624419 79 -...
(48139024681559938504305981503335 2555*Y(2)^2*sin(Y(3)))/...
30642688992900114225796731779907 1664 -...
(18417376263271965926643140617481 979*Y(2)^2*...
cos(Y(3))*sin(Y(3)))/191516806205625713911229573624419 79;
Y(5);
Y(6)];

end
```

C.5 General Symbolic functions

C.5.1 VelocitySaver

```
function VelocitySaver( Vel, NFrames )
```

```
%VelocitySaver
%This function takes the equations that give the velocities
%of the bodies and saves them to a file called velocities.mat.


%   The equations are saved such that the resulting function can take the
%   entire solution matrix in one go, without the need for a for loop.

%Create the file and save the function header to it
fid=fopen('C:\Users\Thorstein\Dropbox\Master\Master\Matlab\TempFiles\Velocities.m','
fprintf(fid,'%s\n\n','function [V]=Velocities(InP)');

%Start the velocity vector
fprintf(fid,'%s','V=[');

% Fix the equations so that they can be used as a function
for i=1:NFrames*6-1
    %If a velocity is 0, make sure the position follows the size of the
    %input matrix.
if Vel(i,1)==0
   fprintf(fid,'%s;\n','zeros(1,size(InP,2))');
else
%Set the equation to be evaluated and saved
VelStr=char(Vel(i,1));
Thc=1;
dThC=2;
%Switch the symbols with input numbers.
for k=1:NFrames
VelStr=regexprep(VelStr,strcat('dTh',num2str(k)),strcat('InP(',num2str(dThC),',:)'))
VelStr=regexprep(VelStr,strcat('Th',num2str(k)),strcat('InP(',num2str(Thc),',:)'));
Thc=Thc+2;
dThC=dThC+2;
end
%Save the expressions
VelStr=regexprep(VelStr,'*','.*');
fprintf(fid,'%s;\n',VelStr);
end
end

%The last input
Thc=1;
```

```
dThC=2;
%If a velocity is 0, make sure the position follows the size of the
%input matrix.
if Vel(end,1)==0
   fprintf(fid,'%s];\n\n','zeros(1,size(InP,2))');
   fprintf(fid,'%s','end');
else
%Set the equation to be evaluated and saved
VelStr=char(Vel(end,1));
%Switch the symbols with input numbers.
for i=1:NFrames
VelStr=regexprep(VelStr,strcat('dTh',num2str(i)),strcat('InP(',num2str(dThC),
VelStr=regexprep(VelStr,strcat('Th',num2str(i)),strcat('InP(',num2str(Thc),',
Thc=Thc+2;
dThC=dThC+2;
end
%Save the last expression
VelStr=regexprep(VelStr,'*','.*');
fprintf(fid,'%s];\n\n',VelStr);
fprintf(fid,'%s','end');
end
end
```

C.5.2 PosVectorFunctionMaker

```
function PosVectorFunctionMaker( NFrames,PreVList,PostVList,Axis )
%{
PosVectorFunctionMaker writes two functions and saves them to the active
folder. The two functions take the angles as input and gives the position
of the centre of masses and the position of the points of rotations as
output respectively. They are to be used to plot the system.
Started by MFMNumEquMaker
%}

%{
Input:
NFrames: The total amount of frames/bodies in the system

PreVList: The list of position vectors pointing to the points of rotation
from the last frame
```

```
PostVList: The list of position vectors pointing from the last point of
rotation to the next centre of mass

SymThV: The list containing the thetas as symbolic variables.

Axis: The axis of rotation for the frames
%}

%Create the theta symbols
[ SymThV,~,~,~,~] = SymVMaker( NFrames );
%------------------------------------------------------------------------

%MAke the first connetction matrix
EMatrix=EMatrixMaker(PreVList(:,1),Axis(1),PostVList(:,1),SymThV(1));
%Pre allocate
CMPosVFunc=sym(zeros(3*NFrames,1));
RotPosFunc=sym(zeros(3*NFrames+6,1));
%------------------------------------------------------------------------

%Set the first positions
CMPosVFunc(1:3,1)= EMatrix(1:3,4);
RotPosFunc(4:6,1)= PreVList(:,1);
RotPosFunc(7:9,1)= EMatrix(1:3,1:4)*[PreVList(:,2);1];
%------------------------------------------------------------------------

%Kontroll vector, incase the Cm or points of rotaion does not move with
%respect to the X, Y and Z axis.
CMControl=zeros(NFrames*3,1);
RotControl=CMControl;
CMControl(1:3,1)=PreVList(:,1)+PostVList(:,1);
RotControl(1:3,1)=CMControl(1:3,1)+PreVList(:,2);
for i=2:NFrames
    CMControl(i*3-2:i*3,1)=CMControl((i-1)*3-2:(i-1)*3,1)+PreVList(:,i)...
        +PostVList(:,i);
    RotControl(i*3-2:i*3,1)=CMControl(i*3-2:i*3,1)+PreVList(:,i+1);
end
%------------------------------------------------------------------------

%Set the positions of the centre of mass and the points of rotation and the
%last end point for the last body
```

```
for i=2:NFrames
    EMatrix=EMatrix*EMatrixMaker(PreVList(:,i),Axis(i),PostVList(:,i),...
        SymThV(i));
    CMPosVFunc(i*3-2:i*3,1)= EMatrix(1:3,4);
    RotPosFunc(i*3+4:i*3+6,1)=EMatrix(1:3,1:4)*[PreVList(:,i+1);1];
end
%--------------------------------------------------------------------------

%Simplify the equations
CMPosVFunc=simplify(CMPosVFunc);
RotPosFunc=simplify(RotPosFunc);
%--------------------------------------------------------------------------

%Prepare the header strings for the functions
HeaderCM='function  [CMPosVList]=CMPosFunction(Thetas)';
HeaderRot='function  [RotPosVList]=RotPosFunction(Thetas)';
%--------------------------------------------------------------------------

%Print the header,the variable names and prealocation comand for the
%functions
SavePath='C:\Users\Thorstein\Dropbox\Master\Master\Matlab\TempFiles\';
FileNaAmeCM='CMPosFunction.m';
FidNameCM=strcat(SavePath,FileNaAmeCM);


FileNaAmeRot='RotPosFunction.m';
FidNameRot=strcat(SavePath,FileNaAmeRot);

fidCM=fopen(FidNameCM,'wt');
fidRot=fopen(FidNameRot,'wt');

fprintf(fidCM,'%s \n\n',HeaderCM);
fprintf(fidCM,'%s','CMPosVList=[');


fprintf(fidRot,'%s \n\n',HeaderRot);
fprintf(fidRot,'%s','RotPosVList=[');
%--------------------------------------------------------------------------

%Print the function that gives the position of the centre of masses
for i=1:numel(CMPosVFunc)-1
```

```
if CMPosVFunc(i)==0
    fprintf(fidCM,'%s;\n','zeros(1,size(Thetas,2))');
elseif CMPosVFunc(i)==CMControl(i)
    fprintf(fidCM,'%s;\n',strcat('ones(1,size(Thetas,2))*',...
        char((CMPosVFunc(i)))));
else
CmPosString=char(CMPosVFunc(i));
InC=1; %Input counter
    for j=1:NFrames
    CmPosString=regexprep(CmPosString,strcat('Th',num2str(j)),...
        strcat('Thetas(',num2str(InC),',:)'));
    InC=InC+2;
    end
CmPosString=regexprep(CmPosString,'*','.*');
fprintf(fidCM,'%s;\n',CmPosString);
end
end



if CMPosVFunc(end)==0
    fprintf(fidCM,'%s];\n\n end','zeros(1,size(Thetas,2))');
elseif CMPosVFunc(end)==CMControl(end)
    fprintf(fidCM,'%s];\n\n end',strcat('ones(1,size(Thetas,2))*',...
        char((CMPosVFunc(end)))));
else
InC=1;
CmPosString=char(CMPosVFunc(end));
for j=1:NFrames
    CmPosString=regexprep(CmPosString,strcat('Th',num2str(j)),...
        strcat('Thetas(',num2str(InC),',:)'));
InC=InC+2;
end
CmPosString=regexprep(CmPosString,'*','.*');
fprintf(fidCM,'%s];\n\n end',CmPosString);
end
%-------------------------------------------------------------------------
%Print the function giving the points of rotations
fprintf(fidRot,'%s;\n','zeros(1,size(Thetas,2))',...
    'zeros(1,size(Thetas,2))','zeros(1,size(Thetas,2))');
for i =4:6
```

```
if RotPosFunc(i)==0
    fprintf(fidRot,'%s;\n','zeros(1,size(Thetas,2))');
else
    fprintf(fidRot,'%s;\n',strcat('ones(1,size(Thetas,2))*',...
        char((RotPosFunc(i)))));
end
end


for i=7:numel(RotPosFunc)-1
if RotPosFunc(i)==0
    fprintf(fidRot,'%s;\n','zeros(1,size(Thetas,2))');
elseif RotPosFunc(i)==RotControl(i-6)
    fprintf(fidRot,'%s;\n',strcat('ones(1,size(Thetas,2))*',...
        char((RotPosFunc(i)))));
else
    InC=1;
    RotString=char(RotPosFunc(i));
    RotString=strcat('ones(1,size(Thetas,2))*',RotString);
    for j=1:NFrames
    RotString=regexprep(RotString,strcat('Th',num2str(j)),...
        strcat('Thetas(',num2str(InC),',:)'));
    InC=InC+2;
    end
    RotString=regexprep(RotString,'*','.*');
    fprintf(fidRot,'%s;\n',RotString);
end
end

if RotPosFunc(end)==0
    fprintf(fidRot,'%s];\n\n end','zeros(1,size(Thetas,2))');
elseif RotPosFunc(end)==RotControl(end)
    fprintf(fidRot,'%s];\n\n end',strcat('ones(1,size(Thetas,2))*'...
        ,char((RotPosFunc(end)))));
else
RotString=char(RotPosFunc(end));
RotString=strcat('ones(1,size(Thetas,2))*',RotString);
InC=1;
for j=1:NFrames
    RotString=regexprep(RotString,strcat('Th',num2str(j)),...
        strcat('Thetas(',num2str(InC),',:)'));
```

```
    InC=InC+2;
end
RotString=regexprep(RotString,'*','.*');
fprintf(fidRot,'%s];\n\n end',RotString);
end
%------------------------------------------------------------------------
%Close the files
fclose(fidCM);
fclose(fidRot);
end
```

C.5.3 RotMaker

```
function [ RotMatrix ] = RotMaker( axis, ang )
%RotMaker:
%Uses the axi defined by axis and the symbol in ang to create the
%rotational matrix.

sym(ang);
if axis==1
RotMatrix=sym([1,0,0;0,cos(ang),-sin(ang);0,sin(ang),cos(ang)]);

elseif axis==2
RotMatrix=sym([cos(ang),0,sin(ang);0,1,0;-sin(ang),0,cos(ang)]);

elseif axis==3
RotMatrix=sym([cos(ang),-sin(ang),0;sin(ang),cos(ang),0;0,0,1;]);

else
    error('axis must be 1, 2 or 3')

end
```

C.5.4 dRotMaker

```
function [ dRotMatrix ] = dRotMaker( axis, ang , dang )
%dRotMaker This function sets up the time derivative of the Rotation matrix
%denoted by axis and ang. It is used by B_dB_D_Maker, in the creation of dB

%Create the time derivative of the rotational matrix
```

```
sym(ang);
sym(dang);
switch axis
case{1}
dRotMatrix=sym([0,0,0;0,-sin(ang),-cos(ang);0,cos(ang),-sin(ang)]*dang);


case{2}
dRotMatrix=sym([-sin(ang),0,cos(ang);0,0,0;-cos(ang),0,-sin(ang)]*dang);


case{3}
dRotMatrix=sym([-sin(ang),-cos(ang),0;cos(ang),-sin(ang),0;0,0,0;]*dang);

end
end
```

C.5.5 SqewThis

```
function [ SqewdMatrix ] = SqewThis( Vector )
%This function takes a vector and returns the hat map of the 3 first
%entries
SqewdMatrix=[0       ,-Vector(3),  Vector(2);
            Vector(3),    0     , -Vector(1);
           -Vector(2),Vector(1) ,     0    ];
end
```

C.5.6 UnSqewThis

```
function [ UnSqewedVector ] = UnSqewThis( SqewMatrix )
%V map of a 3X3 matrix

if size(SqewMatrix,1)==3 && size(SqewMatrix,2)==3
UnSqewedVector=[-SqewMatrix(2,3);SqewMatrix(1,3);-SqewMatrix(1,2)];
else
    error('The matrix must be a 3X3 matrix')
end
end
```

C.5.7 EMatrixMaker

```
function [ EMatrix ] = EMatrixMaker( PreV,axis,PostV,ang )
%EMatrixMAker:
%THis function takes the vector befor the rotation, the axis of rotation
%and the vector after to create the corresponding connection matrix

sym(ang);
%Pre allocate
PreM=sym(eye(4));
RotM=PreM;
PostM=PreM;

%set vectors
PreM(1:3,4)=PreV;
PostM(1:3,4)=PostV;
%Construct rotational matrix
RotM(1:3,1:3)=RotMaker(axis,ang);



%Evaluate connection matrix
EMatrix=PreM*RotM*PostM;
EMatrix=simplify(EMatrix);
end
```

C.5.8 InvE

```
function [ InvertedE ] = InvE( EMatrix )
%Invert the E matrix using the closed form.
InvRot=inv(EMatrix(1:3,1:3));
InvPos=-InvRot*EMatrix(1:3,4);
InvertedE=sym(eye(4));
InvertedE(1:3,1:3)=InvRot;
InvertedE(1:3,4)=InvPos;
InvertedE=simplify(InvertedE);
end
```

C.5.9 Canonical Equations of Motion

```
%Defining the symbols to be used
syms Th1 Th2 p1 p2 dTh1 dTh2 J11 J12 J13 J14 J15 J16 J17 J18 J19 J21 ...
    J22 J23 m1 m2 g L1 L2 L3 l11 l12 l13 l2 real


%The temporary angular velocities in a vector
dqTemp=[dTh1;dTh2];
%Make the rotational matrises
R1=RotMaker(3,Th1);
R21=RotMaker(2,Th2);

%Defining the angular velocity matrises
omg1M=[   0,-dTh1,0;
      dTh1,    0,0;
         0,    0,0];
omg21M=[    0,0,dTh2;
           0,0,   0;
       -dTh2,0,   0];
omg2M=simplify(expand(R21'*omg1M*R21+omg21M));

%The angular velocity vectors
omg1=[0;0;dTh1];
omg2=UnSqewThis(omg2M);

%Defining the position vectors
r1=[L1;L2;L3];%Position of centre of mass 1 from inertial
rp1=[l11;l12;l13];%Position of the Rotational point between arm and column
r2p=[l2;0;0];%Position of centre of mass 2 from the joint
r2=r1+rp1+R21*r2p;
M= sym(zeros(12));
%Mass of the first body
M(1:3,1:3) =[m1,  0,  0;
              0, m1,  0;
              0,  0, m1];
%Mass of the second body
M(4:6,4:6)=[J11,J12,J13;J14,J15,J16;J17,J18,J19];
%Massmoment of inertia of the first body
M(7:9,7:9)=[m2,  0,  0;
             0, m2,  0;
```

```
             0,  0, m2];
%Massmoment of inertia of the second body
M(10:12,10:12)=[J21,0,0;0,J22,0;0,0,J23];

%Potential energy
V=(l2 -sin(Th2)*l2 )*m2*g;
```

B matrix

```
B=sym(zeros(12,2));
B(1:3,1)=-R1*SqewThis(r1)*sym([0;0;1]);
B(4:6,1)=sym([0;0;1]);
B(7:9,1)=-R1*SqewThis(r2)*sym([0;0;1]);
B(7:9,2)=-R1*R21*SqewThis(r2p)*sym([0;1;0]);
B(10:12,1)=R21'*sym([0;0;1]);
B(10:12,2)=sym([0;1;0]);
```

Canonical momenta

```
dX=B*dqTemp;
L=expand(simplify((1/2).*((dX)'*M*(dX))-V));
P1=simplify(expand(diff(expand(L),dTh1)));
P2=simplify(expand(diff(expand(L),dTh2)));
P=[P1;P2];

%Construct the Q matrix
Q=sym(zeros(2));
Q(1,1)=simplify(expand(diff(P1,dTh1)));
Q(1,2)=simplify(expand(diff(P1,dTh2)));
Q(2,1)=simplify(expand(diff(P2,dTh1)));
Q(2,2)=simplify(expand(diff(P2,dTh2)));

%simplify(Q*dqTemp-P) %Test to make sure Q is correct.

%Prepare for use in the Hamiltonian
InvQ=simplify(expand(inv(Q)));
dq=simplify(expand(InvQ*[p1;p2]));
```

Hamiltonian

```
H=simplify(expand(dq'*[p1;p2]-((1/2)*(dq'*B'*M*B*dq)-V)));
```

Equations of motion

```
dP1=-simplify(expand(diff(H,Th1)));
dP2=-simplify(expand(diff(H,Th2)));

dq1=simplify(expand(diff(H,p1)));
dq2=simplify(expand(diff(H,p2)));

%The equations are copy/pasted from the comand window to the function funcH
```

C.6 The numerical schemes

C.6.1 Symplectic Midpoint

Contents

- Set physical Values
- Pre allocate
- Integration loop

```
function [T,SolM ]=IntSMid(StepSize,Tfin,InitVal,NFrames,MaksErr,SaveStep)

%{
%IntSMid integrates the canonical equations of motion saved in funcH by use
 of the midpoint rule.

   Innput:
            StepSize : The size of the time step.
            Tfin     : End time for the integration
            InitVal  : Initial values for the integration on the form
                       InitVal=[th1;dth1;th2;dth2]
            NFrames  : The number of frames in the problem
            MaksErr  : Maximum global error
            SaveStep : The interval of solution points wanted, incase not
                       all solution points is needed to the output
```

```
   Output:
            T    : Time vector as a hx1 vector
            SolM : Solution matrix size(NFrames*3,Steps). It is on the form
                    Th1
                    dTh1
                    Th2
                    dTh2
                    P1
                    P2
%-------------------------------------------------------------------------
%}
```

Set physical Values

```
%Mass
m1=3.3645051;
m2=4.6811099;

%Mass moment of inertia
J19= 0.06172657200;%kg*m^2
J21= 4.1857*10^(-3);%kg*m^2
J22=  1.9213*10^(-2);%kg*m^2
J23=  1.97783*10^(-2);%kg*m^2

% Lengths
l2=0.1267;%m
L1=1.1705442e-01;
L2=-1.9696113e-02;
l11=0.13294558;
l12=1.9696113e-02;

%The thetas
Th1=InitVal(1);
dTh1=InitVal(2);
Th2=InitVal(3);
dTh2=InitVal(4);

%Calculate the generalized momentas, equations generated by Hamiltonian.mat
P1 = J19*dTh1 + J21*dTh1 - J21*dTh1*cos(Th2)^2 + J23*dTh1*cos(Th2)^2 + ...
    L1^2*dTh1*m1 + L1^2*dTh1*m2 + L2^2*dTh1*m1 + L2^2*dTh1*m2 + ...
```

```
    dTh1*l11^2*m2 + dTh1*l12^2*m2 + dTh1*l2^2*m2*cos(Th2)^2 + ...
    2*L1*dTh1*l11*m2 + 2*L2*dTh1*l12*m2 + 2*L1*dTh1*l2*m2*cos(Th2) + ...
    L2*dTh2*l2*m2*sin(Th2) + 2*dTh1*l2*l11*m2*cos(Th2) + ...
    dTh2*l2*l12*m2*sin(Th2);
P2 = J22*dTh2 + dTh2*l2^2*m2 + L2*dTh1*l2*m2*sin(Th2) +...
     dTh1*l2*l12*m2*sin(Th2);


h=StepSize;
%Find the local maks error
LocEr=MaksErr*h/Tfin;
%--------------------------------------------------------------------------
```

Pre allocate

```
NSteps = round(Tfin/h);
y0=[Th1;Th2;P1;P2];
y1=y0;
SolM=zeros(NFrames*3,round(NSteps/SaveStep)+1);
T=zeros(round(NSteps/SaveStep)+1,1);
SaveCount=1;
SaveStepC=SaveStep;
ThC=1:2:NFrames*2;
dThC=2:2:NFrames*2;

SolM(ThC,1) =y0(1:NFrames);
SolM(NFrames*2+1:end,1) =y0(NFrames+1:end);
%Set First angular velocities;
 dTh=funcH(0,y1);
 SolM(dThC,1)=dTh(1:NFrames);


 it100C=0;
```

Integration loop

```
for i = 1:NSteps
%Prepare for the next loop
    nit=0;
    t=i*h;
```

```
    err=1;

while err>LocEr && nit<100
%{
Loop runs until the error is smaller than requiered, with a max of 100
itterations
%}
    yprev = y1; %Save previous result
    y1= y0+(h)*funcH(t,(y0+y1)/2);%Calculate new result
    err = norm(y1-yprev);%Calculate local error
    nit = nit+1;%Increase iteration counter

end
if nit==100
    it100C=it100C+1;
end
if i==SaveStepC || i==NSteps
   dTh=funcH(t,y1);
    SolM(dThC,SaveCount+1)=dTh(1:NFrames);%Save the angular velocities
    SolM(ThC,SaveCount+1)=y1(1:NFrames);%Save the angles
    %Save the canonical momentas
    SolM(NFrames*2+1:end,SaveCount+1)=y1(NFrames+1:end);
    T(SaveCount+1)=t;%Increase time counter
    SaveCount=SaveCount+1;
    SaveStepC=SaveStepC+SaveStep;
end
    y0=y1;%Save new start value

end
end
```

C.6.2 Midpoint rule

```
function [ T, SolM ] = IntMid(StepSize,Tfin,InitVal,Mom,NFrames,MaksErr )
%{
Integrator using the trapezoidal method scheme.

Input:
    Stepsize = The size of the timestep to be used.
    Tfin = The stop time for the calculation
```

```
    InitVal = The initial values for the integration. Given as a vector.
    Mom = The external moments applied to the system
    NFrames = The number of frames in the system.
    MaksErr = The maximum global error allowed during the fixed point
              itteration
Output
    T = The time steps as a vector
    SolM = The solution matrix
%}
%-------------------------------------------------------------------------
%Find the local error
LocEr=MaksErr/Tfin;
%Only use the needed initial values, in case the vector is too long.
InitVal=InitVal(1:NFrames*2);
Mom=Mom(1:NFrames);
h=StepSize;
%-------------------------------------------------------------------------
%Size of the initial values
InSize=size(InitVal);
%Check that the initial values and moment is a column vector, else
%transpose it
if InSize(1)==1
    InitVal=InitVal';
end

MoSize=size(Mom);
if MoSize(1)==1
    Mom=Mom';
end
%-------------------------------------------------------------------------
%Pre allocate and prepare for integration
y0 = [InitVal;Mom];
y1=y0;
NSteps = round(Tfin/h);
SolM=zeros(NFrames*3,NSteps+1);
SolM(:,1) =y0;
T=zeros(NSteps+1,1);

%Integration loop
for i = 1:NSteps
%Prepare for the next loop
```

```
    nit=0;
    t=i*h;
    err=1;
while err>LocEr && nit<100
%{
Loop runs until the error is smaller than requiered, with a max of 100
itterations
%}
    yprev = y1; %Save previous result
    y1= y0+(h)*(func(t,(y0+y1)/2));%Calculate new result
    err = norm(y1-yprev);%Calculate local error
    nit = nit+1;%Increase iteration counter
end
    y0=y1;%Save new start value
    SolM(:,i+1)=y1;%Save the solution
    SolM(NFrames*2+1:end,i+1)=Mom;%Update moments
    T(i+1)=t;%Increase time counter
end
end
```

C.7 Other functions

C.7.1 EAnalysis

```
function [ TotEnCh,Energy] = EAnalysis( SolM,NFrames, Jlist,Mass,Gravity,LP )
%EAnalysis
%This fuction takes the solution matrix from integration, the mass, mass
%moment of inertia, gravity constant and lowest points for the centres of
%mass and resturns the energy of each timestep in SolM
%{
Input:

        SolM : Solution matrix from integration, typicaly ode45
        NFrames : Number of bodies
        JList : Mass moment of inertia of the bodies
        Mass : Mass of the bodies
        Gravity : Gravity constant
        LP : Lowest point the repspective centre of masses can reach in the
        inertial 3. direction.
```

```
Output:

        ENergy: Vector containing the energy for each timestep in SolM
        TotEnergyCh : Total energy change



%}
%Pre allocate
Size=size(SolM);
Energy=zeros(1,Size(2));
CmPC=3:3:NFrames*3;
g=Gravity;

I=eye(3);
Zero=zeros(3);
M=zeros(NFrames*6);

%Get the velocities and centre of mass positions for the system, generated
%by MFMNumEquMaker.
Vel=Velocities(SolM);
CmP=CMPosFunction(SolM);

%Create the M matrix
for i=1:NFrames
MC=i*6-5:i*6;
JC=i*3-2:i*3;
M(MC,MC)=[Mass(i)*I,Zero;Zero,Jlist(1:3,JC)];
%LP(i)=min(CmP(i*3,:))
end
%Calculate energy
for i=1:Size(2)
    TempEk=0.5*(Vel(:,i)'*M*Vel(:,i)); %equ 13.22b ch13 in the book by Thomas
    TempEp=g*Mass(1:NFrames)*(CmP(CmPC,i)-LP(1:NFrames)');
    Energy(i)=TempEk+TempEp;
end
TotEnCh=sum(diff(Energy));
end
```

C.7.2 Velocities

```
function [V]=Velocities(InP)

%{
Velocities:
Takes the solution matrix from integration and returns the
generalized velocities
%}

V=[InP(2,:).*((1419254514679155.*cos(InP(1,:)))/72057594037927936 -
 (8434659876705113.*sin(InP(1,:)))/72057594037927936);
InP(2,:).*((8434659876705113.*cos(InP(1,:)))/72057594037927936 +
 (1419254514679155.*sin(InP(1,:)))/72057594037927936);
zeros(1,size(InP,2));
zeros(1,size(InP,2));
zeros(1,size(InP,2));
InP(2,:);
- (InP(2,:).*sin(InP(1,:)).*(9133759771757328.*cos(InP(3,:)) +
 18014398509481985))/72057594037927936 -
  (570859985734833.*InP(4,:).*cos(InP(1,:)).*
  sin(InP(3,:)))/4503599627370496;
(InP(2,:).*cos(InP(1,:)).*(9133759771757328.*cos(InP(3,:)) +
 18014398509481985))/72057594037927936 -
  (570859985734833.*InP(4,:).*sin(InP(1,:)).*sin(InP(3,:)))/
  4503599627370496;
-(570859985734833.*InP(4,:).*cos(InP(3,:)))/4503599627370496;
-InP(2,:).*sin(InP(3,:));
InP(4,:);
InP(2,:).*cos(InP(3,:))];

end
```

C.7.3 CMPosFunction

```
function  [CMPosVList]=CMPosFunction(Thetas)

%CMPosFunction
%This function takes a matrix of angles, and returns the centre of mass
%positions of the system. The function is created by PosVectorFunctionMaker
```

```
CMPosVList=[(8434659876705113.*cos(Thetas(1,:)))/72057594037927936 +...
    (1419254514679155.*sin(Thetas(1,:)))/72057594037927936;
(8434659876705113.*sin(Thetas(1,:)))/72057594037927936 - ...
(1419254514679155.*cos(Thetas(1,:)))/72057594037927936;
ones(1,size(Thetas,2))*169462921707575/562949953421312;
(cos(Thetas(1,:)).*(9133759771757328.*cos(Thetas(3,:)) +...
18014398509481985))/72057594037927936;
(sin(Thetas(1,:)).*(9133759771757328.*cos(Thetas(3,:)) + ...
18014398509481985))/72057594037927936;
84178615221526153/288230376151711744 -...
(570859985734833.*sin(Thetas(3,:)))/4503599627370496];

 end
```

C.7.4 RotPosFunction

```
function  [RotPosVList]=RotPosFunction(Thetas)

%RotPosFunction
%This function takes a matrix of angles, and returns the position of the
%rotation points positions of the system. The function is created by
%PosVectorFunctionMaker, and used by MFMPlotter.


RotPosVList=[zeros(1,size(Thetas,2));
zeros(1,size(Thetas,2));
zeros(1,size(Thetas,2));
zeros(1,size(Thetas,2));
zeros(1,size(Thetas,2));
ones(1,size(Thetas,2)).*(18014398509481985.*cos(Thetas(1,:)))...
/72057594037927936;
ones(1,size(Thetas,2)).*(18014398509481985.*sin(Thetas(1,:)))...
/72057594037927936;
ones(1,size(Thetas,2)).*84178615221526153/288230376151711744;
ones(1,size(Thetas,2)).*(cos(Thetas(1,:)).*(9133759771757328.*...
cos(Thetas(3,:)) + 18014398509481985))/72057594037927936;
```

```
ones(1,size(Thetas,2)).*(sin(Thetas(1,:)).*(9133759771757328.*...
cos(Thetas(3,:)) + 18014398509481985))/72057594037927936;
ones(1,size(Thetas,2)).*84178615221526153/288230376151711744 -...
(570859985734833.*sin(Thetas(3,:)))/4503599627370496];

end
```

C.7.5 MFMPlotter

```
function MFMPlotter( sol,NFrames )
%MFMPlotter This function takes the solution matrix from integration and
%plots the solution as a simple animation.


%Counters to get all X direction coordinates, all Y and all Z in one go.
XcCM=1:3:NFrames*3;
YcCM=2:3:NFrames*3;
ZcCM=3:3:NFrames*3;
%Get the position vectors for the centre of masses and points of rotation


    CMPoV =CMPosFunction(sol);
    RotPoV=RotPosFunction(sol);
LimAx=max(max(RotPoV));
for i=1:100:size(sol,2)-1

    plot3(CMPoV(XcCM,i),CMPoV(YcCM,i),CMPoV(ZcCM,i),'r*')
    grid on
    hold on

    for J=2:NFrames+2

       plot3([RotPoV(J*3-5,i),RotPoV(J*3-2,i)],...
             [RotPoV(J*3-4,i),RotPoV(J*3-1,i)],...
             [RotPoV(J*3-3,i),RotPoV(J*3,i)])
    end
    plot3(RotPoV(end-2,1:i),RotPoV(end-1,1:i),RotPoV(end,1:i))
```

```
    axis([-LimAx,LimAx,-LimAx,LimAx,-LimAx,LimAx]);
    %view([0,90]);
    xlabel('X')
    ylabel('Y')
    zlabel('Z')
    drawnow
    hold off
end
end
```