FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Master's thesis

Interpolation of suspension kinematics for the purpose of vehicle dynamics simulation

Václav Houdek

30.6.2020

Annotation

This thesis aims at the reduction of the computation time during the kinematic solution of multibody systems. Especially during vehicle simulations, including cornering test, bump test or moose test, only some parameters of the suspension are optimized, and the simulation is repeated many times. Suspensions are usually 1 DoF systems. The expression of the position of a wheel support in terms of some parameter can be obtained by solving constraints imposed by joints present in a multibody system. Due to these constraints, kinematic is described by a system of nonlinear algebraic equations. Its solution is usually ineffective and therefore time-consuming. The interpolation requires much less computation time at the cost of losing accuracy. This thesis proposes a strategy consisting of solving the kinematic problem at position and velocity level for particular points and using these points to calculate all other positions by interpolation. The introduced interpolation is continuous in acceleration level. At the end of this work, computed results are compared with results obtained by commercial software and code implemented in accordance with common multibody theory.

Acknowledgement

I cannot express enough thanks to my supervisor Ing. Luboš Smolík, Ph.D. and my consultant doc. Ing. Michal Hajžman, Ph.D. for their continued support, encouragement and patience. I would like to express my sincere appreciation to both of them.
My completion of this project could not have been accomplished without the support of prof. Olivier Verlinden and the Department of Theoretical Mechanics, Dynamics and Vibration at the University of Mons.
Finally, I would like to thank my family and my girlfriend for a big support and understanding during the writing process of this work.

Declaration

I, the undersigned, hereby declare that this master's thesis is my own original work I have prepared on the basis of consultations with my supervisors and that all sources have been accurately reported and acknowledged, and that this document has not been previously, in its entirety or in part, submitted at any university in order to obtain academic qualifications.

In Pilsen Václav Houdek

Contents

Chapter 1

Introduction

Kinematics is a branch of classical mechanics that describes the motion of points, bodies and systems of bodies (*multibody systems*) without considering the forces that cause them to move [1].
The multibody system is defined to be a collection of subsystems called *bodies, components* or *structures*. The motion of the subsystems is kinematically constrained because of different types of joints, and each subsystem or component may undergo large translations and rotational displacements [2]. The kinematic analysis, which aims at describing all the possible motions for the mechanism, has two parts [3]:

- *Solving*, i.e. to find all the sets of values corresponding to real solutions for a given set of values of the input parameters. It has to be done many times for a wide range of values of these parameters.
- *Following trajectories*, by describing how each solution varies when the input is changing continuously, with particular attention to pay to singular points, where these trajectories are crossing.

1.1 Goals and structure of the thesis

This thesis is focused on the kinematic solving of the car suspension. The main aim of this work is a research on this field, a solution of the suspension kinematics and its interpolation. The final goal is a computer implementation of the proposed methods and application on a real car model with recommendations for the next work.

Structure of this thesis strongly corresponds with its goals and can be described in the following points:

- state of the art - standard approaches how to solve kinematics of a car suspension,
- kinematics solving,
- interpolation of the kinematics,
- computer implementation of the kinematics and the interpolation,
- application on a real car suspension and recommendations for the next work.

1.2 State of the art

However, there are many hidden points of view of solving the kinematics, including kinematics with imperfect joints, with rigid or deformable components, forward kinematics and kinematic synthesis.
Raghavan and Roth reviewed three important exact computational methods [15]. They are respectively representative of three families of methods [3]:

- **Symbolic Methods** compute explicit symbolic expressions of the output parameters in terms of the input and fixed parameters. This method is not practicable, even for a simple McPherson mechanism. Gröbner basis or Triangular set also belong to symbolic methods [3].
- **Symbolic-numeric methods** have been mostly developed for solving a set of polynomial equations. Typically a pre-processing of these equations is done, computing a matrix from the coefficients of the polynomial equations. Using this matrix, one can transform the solving problem into a linear eigenvalues computation problem or a univariate polynomial solving problem. These methods provide a numerical result for each given set of input parameters even if the intermediate matrix is by construction very large [4].
- **Certified Numerical Methods** may be used even if the equations are not algebraic, which is advantageous when using the Cartesian coordinate approach. They are based on a numerical scheme with adaptive steps or a homotopy algorithm [5].

Yves A. Papegay et al. in [3] presented two methods to solve exactly the kinematics of the suspension a symbolic approach and an interval-based approach. Both methods allow detecting singularities in the mechanism. However, the symbolic approach has the advantage that it allows us to determine what parameters have to be changed to avoid the singularity. It must be noted that both methods are exact in the sense that the computed trajectories are guaranteed. This guarantee is a large improvement as undetected jumps between branches may occur when using the usual method based on an iterative use of Newton-Raphson scheme.
The main aim of the thesis is to solve kinematics and interpolate it, so the following text is focused on the first part of the kinematic analysis, i.e. *Solving.*
Many problems in mechanisms analysis and synthesis and robotics lead naturally to systems of polynomial equations. Typical constraints state that two points on a rigid body remain a fixed distance apart or that the angle between two lines in a rigid body must remain constant. Such constraints are generally expressed by vector dot and cross-products, and they result in polynomial equations usually of the second degree [15].
In [25], there are another two mathematical approaches based on the **Certified Numerical Methods** that can be used to proceed with kinematic solving:

- **Algebraic Equations Formulation**

  In this case, in mathematical terms, the kinematics problem is a system of coupled nonlinear algebraic equations. If each constraint is a function of the coordinates $\mathbf{p}$, they can be collected in a vector as:

$$\mathbf{\Phi}(\mathbf{p}) = \mathbf{0}. \tag{1.1}$$

  Typically, the number of coordinates (the length of the vector $\mathbf{p}$) would be larger than the number of constraints (the length of the vector $\mathbf{\Phi}$) by one, with the difference corresponding to one degree of freedom associated with the suspension travel. [25]. A typical double wishbone suspension, as shown in figure 1.1, can be modelled with five constraints, chosen as follows: pick any two points that lie on the axis of rotation of the upper arm, e.g., the mounting points of the arm on the chassis (points A and B in figure 1.1). The distance from each of these two points to the upper ball joint must be a constant value. The equations can be written:

$$\mathbf{\Phi} = \begin{Bmatrix} (x_A - x_C)^2 + (y_A - y_C)^2 + (z_A - z_C)^2 - l_{AC}^2 \\ (x_B - x_C)^2 + (y_B - y_C)^2 + (z_B - z_C)^2 - l_{BC}^2 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \tag{1.2}$$

At the end there will be 8 equations between points A−C, B−C, E−G, F−G, H−D, C−D, G−D and C−G. A value for any of the nine coordinates of points G, C and D can be chosen [25].

- **Differential Equations Formulation**

  An alternative approach to the suspension kinematics problem is based on casting the problem as a set of differential equations. Consider the time derivative of the constraint equations, allowing that the constraints are not time-dependent, i.e $\mathbf{\Phi} \neq \mathbf{\Phi}(t)$.

$$\frac{\partial \mathbf{\Phi}}{\partial \mathbf{p}} \dot{\mathbf{p}} = \mathbf{0}. \tag{1.3}$$

The differential constraints are effectively statements that the velocities of the three points on the upright must be perpendicular to the arms themselves, i.e. the rate of a stretch of the arms is zero. These equations are then combined with an additional requirement that some point on the upright has a given velocity, e.g., the point G has a constant vertical speed of 1 $\mathrm{m} \cdot \mathrm{s}^{-1}$. The additional condition is written as a linear function of the velocities:

$$\mathbf{\Psi}(\mathbf{p})\dot{\mathbf{p}} = 1. \tag{1.4}$$

Combining equations (1.3) and (1.4) gives

$$\begin{Bmatrix} \dfrac{\partial \mathbf{\Phi}}{\partial \mathbf{p}} \\ \\ \mathbf{\Psi} \end{Bmatrix} \dot{\mathbf{p}} = \begin{Bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{Bmatrix} \tag{1.5}$$

The result is a linear system of equations in $\dot{\mathbf{p}}$ and in fact, it is an ODE, and so can be solved using a standard routine, e.g. Runge−Kutta. A shortcoming of this approach is that an initial solution for $\mathbf{p}$ must be known, or found using a recursive method as described previously, to serve as the initial condition of the ODE solution [25].

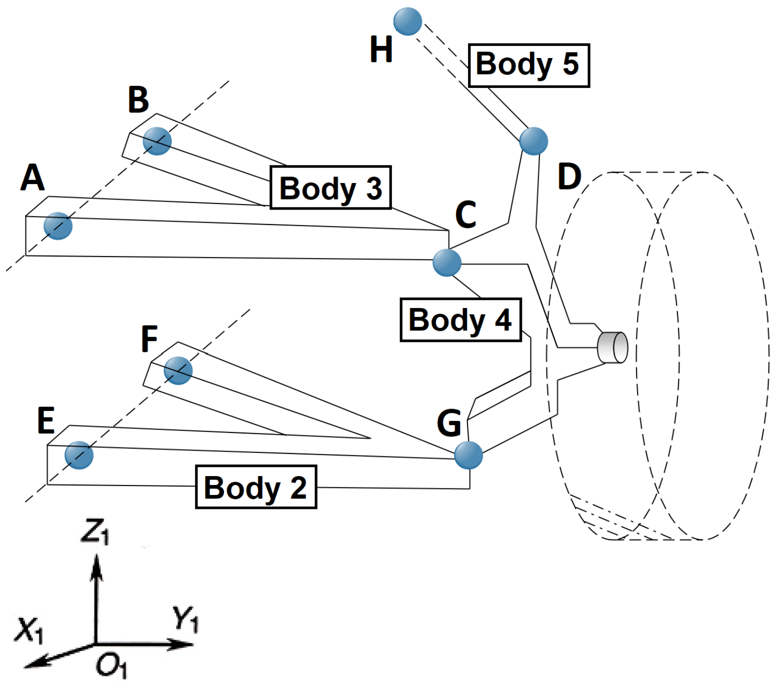


Fig. 1.1: Double wishbone suspension example

Another part of this thesis is focused on the interpolation of kinematics. The interpolation of particle kinematics is just a matter of position, velocity and acceleration interpolation which can be done in a straightforward way by using, e.g. Hermite splines and boundary conditions.
While the interpolation of position and its derivatives can be a simple problem, the interpolation of orientation and its derivatives is more demanding. There are several methods on how to proceed with this interpolation, however, they could be separated into two groups:

- **Methods using quaternions**
  - quaternion slerp [13],
  - C^2-continuous B-spline Quaternion Curve Interpolating a Given Sequence of Solid Orientations [9],
  - Interpolating Solid Orientations with Circular Blending Quaternion Curves [14].
- **Non-quaternion methods**

- geometric slerp (spherical linear interpolation),
- three-axis interpolation [8],
- method based on axis-angle representation [10].

The bridge between the kinematics and the interpolation is a lookup table. In computer science, a lookup table is an array that replaces runtime computation with a more straightforward array indexing operation. The savings in terms of processing time can be significant, since retrieving a value from memory is often faster than undergoing an "expensive" computation or input/output operation [6]. In the case of computational dynamics are the lookup tables mostly used as tables with pre-calculated kinematic data, which are then interpolated [11].

Chapter 2

Suspension kinematic analysis

In this chapter, the kinematics of a typical rear and front suspension is solved. Different types of suspensions exist, McPherson and double wishbone to name a few, but all correspond to a one degree of freedom mechanism. That means a table of transformation matrices giving the successive configuration and their first derivatives of wheel support could be generated in terms of chosen parameter. The suspension is a set of bodies connected by joints, so the Cartesian coordinates approach under Matlab can be used to perform the kinematics analysis. Moreover, spatial kinematics of a body in the Cartesian coordinates, the definition of the transformation matrices and constraint equations and their derivatives are explained. At the end of the chapter, the results are included and discussed.

2.1 Methodology

The Cartesian coordinates approach contains two steps. At the first step, all bodies are considered independently and each body is given an independent set of configuration parameters to let it move freely in space. Here, 6 parameters for each body in space – three for the position and three for the orientation (e.g. Bryant angles) are defined. The second step consists of defining joints between bodies using constraint equations. These equations should be added to the equations of motion of the system when solving dynamics problems. The kinematic analysis comes down to solving the system of constraint equation [16].

The configuration of a multibody system can be described using measurable quantities such as displacements, velocities and accelerations. These are vector quantities that have to be measured with respect to a proper *frame of reference* or *coordinate* *system* [2]. In this text, the term *coordinate system*, which can be represented by three orthogonal axes that are rigidly connected at a point called the origin of this system, is frequently used.

Generally, two types of coordinate systems are required in dealing with multibody systems. The first is a coordinate system that is fixed in time and represents a unique standard for all bodies in the system. This coordinate system will be referred to as a *global coordinate system*. In addition to this we assign a body coordinate system to each component in the system. The body coordinate system translates and rotates with the body; therefore, its location and orientation develop changes with respect to the global coordinate system and with time [2].

If the body coordinate system is located in the centre of mass of the body, the coordinate system is called a *centroidal body coordinate system* [7].

2.1.1 Transformation matrix

The homogeneous transformation matrix giving the configuration of the coordinate system of body i with respect to the global coordinate system (denoted as 0) can be written as

$$\mathbf{T}_{0,i} = \begin{pmatrix} & \mathbf{R}_{0,i}(\boldsymbol{\theta}_i) & & \{\mathbf{e}_i\}_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2.1}$$

where the columns of $\mathbf{R}_{0,i} = [\boldsymbol{x}_i\ \boldsymbol{y}_i\ \boldsymbol{z}_i]$ give the orientation of the axes of the coordinate system of body i in the global coordinate system and $\{\mathbf{e}_i\}_0$ gives the position of the coordinate system of the body i in the global coordinate system.

Body position and velocity

The natural parameters describing the position of the rigid body are of course 3 coordinates which define the position of the coordinate system of the body with respect to the global coordinate system [16]:

$$\{\mathbf{e}_i\}_0 = \mathbf{e}_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}. \tag{2.2}$$

Regardless of the choice of the parameters, the velocities of the centroidal coordinate systems are always a linear function of the time derivatives of the configuration

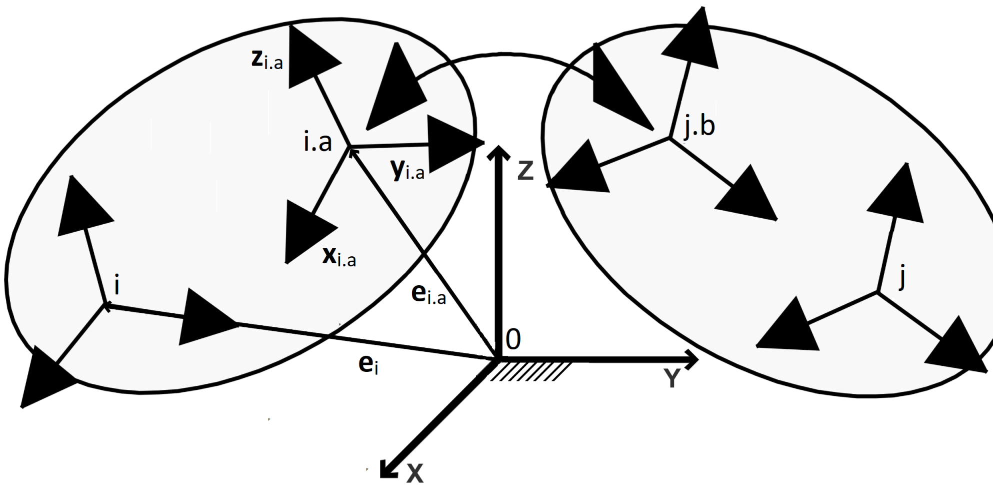


Fig. 2.1: Global coordinate system, centroidal coordinate systems of the bodies (denoted as *i* and *j*) and intermediate coordinate systems of the bodies (denoted as *i.a* and *j.b*) [7]

.

parameters. The tangent matrix $\mathbf{J}_i$ [16] of body *i* can be for the translation defined as

$$\{\mathbf{v}_i\}_0 = \mathbf{v}_i = \mathbf{J}_i\dot{\mathbf{e}}_i = \begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix} \cdot \begin{pmatrix}\dot{x}_i\\ \dot{y}_i\\ \dot{z}_i\end{pmatrix}, \tag{2.3}$$

so that the tangent matrix $\mathbf{J}_i$ is the identity matrix, $\dot{x}_i$, $\dot{y}_i$, $\dot{z}_i$ are scalars and $\boldsymbol{x}_i,\ \boldsymbol{y}_i,\ \boldsymbol{z}_i$ are directional vectors which are included in matrix $\mathbf{R}_{0,i}$. These vectors are used in equation (2.1).

Rotation angles and angular velocity

The rotation parameters were made due to the Bryant angles which go from global frame to local frame by 3 successive rotations about Z, Y' and X" local axes by angles classically denoted by ψ, θ and ϕ respectively, and called *yaw*, *pitch* and *roll* respectively. The corresponding rotation matrix of body *i* is obtained in the following way [16].

$$\mathbf{R}_{0,i} = \underbrace{\begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{T}_z(\psi)} \cdot \underbrace{\begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}}_{\mathbf{T}_{y'}(\theta)} \cdot \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}}_{\mathbf{T}_{x''}(\phi)} = \tag{2.4}$$

$$\begin{pmatrix} \cos\psi\ \cos\theta & \cos\psi\ \sin\phi\ \sin\theta - \cos\phi\ \sin\psi & \sin\phi\ \sin\psi + \cos\phi\ \cos\psi\ \sin\theta \\ \cos\theta\ \sin\psi & \cos\phi\ \cos\psi + \sin\phi\ \sin\psi\ \sin\theta & \cos\phi\ \sin\psi\ \sin\theta - \cos\psi\ \sin\phi \\ -\sin\theta & \cos\theta\ \sin\phi & \cos\phi\ \cos\theta \end{pmatrix}. \tag{2.5}$$

The angular velocity and the associated tangent matrix $\mathbf{\Omega}_i$ are given by [16]

$$\{\boldsymbol{\omega}_i\}_0 = \boldsymbol{\omega}_i = \mathbf{\Omega}_i \cdot \dot{\boldsymbol{\theta}}_i = \begin{pmatrix} 0 & -\sin\psi & \cos\psi\cos\theta \\ 0 & \cos\psi & \sin\psi\cos\theta \\ 1 & 0 & -\sin\theta \end{pmatrix} \cdot \begin{pmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{pmatrix}. \tag{2.6}$$

The complete definition of body i configuration in the global coordinate system is given by [16]

$$\mathbf{T}_{0,i} = \begin{pmatrix} \cos\psi\ \cos\theta & \cos\psi\ \sin\phi\ \sin\theta - \cos\phi\ \sin\psi & \sin\phi\ \sin\psi + \cos\phi\ \cos\psi\ \sin\theta & x \\ \cos\theta\ \sin\psi & \cos\phi\ \cos\psi + \sin\phi\ \sin\psi\ \sin\theta & \cos\phi\ \sin\psi\ \sin\theta - \cos\psi\ \sin\phi & y \\ -\sin\theta & \cos\theta\ \sin\phi & \cos\phi\ \cos\theta & z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

This matrix is written in the same form for all centroidal body coordinate systems, only a vector of body configuration parameters $\mathbf{q}_i = (x_i\ y_i\ z_i\ \phi_i\ \theta_i\ \psi_i)^T$ is changing with respect to the bodies. Another way to get matrix $\mathbf{T}_{0,i}$ is described in appendix [A.2].

The dimension of vector of configuration parameters of all bodies $\mathbf{q}$ (called *vector of configuration parameters* in the following sections) is $n_{cp} = 6 \times n_B$ in space, where n_B is number of all bodies.

$$\mathbf{q} = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ \phi_1 \\ \vdots \\ \psi_{n_B} \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ \vdots \\ q_{6\times n_B} \end{pmatrix}. \tag{2.7}$$

For a better overview the vector of configuration parameters of body i is written as

$$\mathbf{q}_i = (x_i \;\, y_i \;\, z_i \;\, \phi_i \;\, \theta_i \;\, \psi_i)^T = (q_{6i-5} \;\, q_{6i-4} \;\, q_{6i-3} \;\, q_{6i-2} \;\, q_{6i-1} \;\, q_{6i})^T. \tag{2.8}$$

2.1.2 Definition of kinematic constraints

Constraint equations allow to set joints between two arbitrary bodies (body i and body j). Each joint is defined between two different body coordinate systems. Since the joint is not necessarily located in the origins of these two body coordinate systems, two intermediate joint coordinate systems are introduced: the first in body i denoted as $i.a$ and the second in body j denoted as $j.b$. A set of constraint equations ${}^1b$, ${}^2b$, ... ,${}^6b$ is defined as

$$\begin{aligned}
{}^1b &\equiv \mathbf{x}_{i.a} \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) = 0, && (2.9)\\
{}^2b &\equiv \mathbf{y}_{i.a} \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) = 0, && (2.10)\\
{}^3b &\equiv \mathbf{z}_{i.a} \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) = 0, && (2.11)\\
{}^4b &\equiv \mathbf{y}_{i.a} \cdot \mathbf{z}_{j.b} = 0, && (2.12)\\
{}^5b &\equiv \mathbf{z}_{i.a} \cdot \mathbf{x}_{j.b} = 0, && (2.13)\\
{}^6b &\equiv \mathbf{x}_{i.a} \cdot \mathbf{y}_{j.b} = 0. && (2.14)
\end{aligned}$$

The constraint equations relative to the most common joints can be presented as a subset of the 6 previous generic equations. All of the vectors are usually projected in the global coordinate system ($\{\ \}_0$). Tables 2.1 $-$ 2.6 summarize the subset of constraint equations necessary to represent the classical kinematics joints.

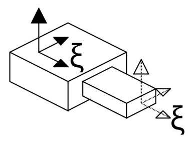
	Type of prismatic joint	Subset
	Prismatic joint along X ($\xi = \mathbf{x}$)	${}^2b$, ${}^3b$, ${}^4b$, ${}^5b$, ${}^6b$
	Prismatic joint along Y ($\xi = \mathbf{y}$)	${}^1b$, ${}^3b$, ${}^4b$, ${}^5b$, ${}^6b$
	Prismatic joint along Z ($\xi = \mathbf{z}$)	${}^1b$, ${}^2b$, ${}^4b$, ${}^5b$, ${}^6b$

Table 2.1: Prismatic (translational) joint [16]

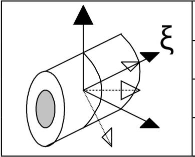
	Type of revolute joint	Subset
	Revolute joint about X ($\xi = \mathbf{x}$)	${}^1b$, ${}^2b$, ${}^3b$, ${}^5b$, ${}^6b$
	Revolute joint about Y ($\xi = \mathbf{y}$)	${}^1b$, ${}^2b$, ${}^3b$, ${}^4b$, ${}^6b$
	Revolute joint about Z ($\xi = \mathbf{z}$)	${}^1b$, ${}^2b$, ${}^3b$, ${}^4b$, ${}^5b$

Table 2.2: Revolute joint [16]

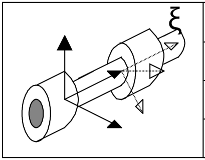
	Type of cylindrical joint	Subset
	Cylindrical joint about X ($\xi = \mathbf{x}$)	${}^2b$, ${}^3b$, ${}^5b$, ${}^6b$
	Cylindrical joint about Y ($\xi = \mathbf{y}$)	${}^1b$, ${}^3b$, ${}^4b$, ${}^6b$
	Cylindrical joint about Z ($\xi = \mathbf{z}$)	${}^1b$, ${}^2b$, ${}^4b$, ${}^5b$

Table 2.3: Cylindrical joint [16]

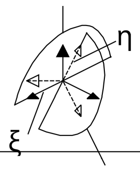
	Type of universal joint	Subset
	Universal joint among XY ($\xi = \mathbf{x}_1$, $\eta = \mathbf{y}_2$)	${}^1b$, ${}^2b$, ${}^3b$, ${}^6b$
	Universal joint among ZX ($\xi = \mathbf{z}_1$, $\eta = \mathbf{x}_2$)	${}^1b$, ${}^2b$, ${}^3b$, ${}^5b$
	Universal joint among YZ ($\xi = \mathbf{y}_1$, $\eta = \mathbf{z}_2$)	${}^1b$, ${}^2b$, ${}^3b$, ${}^4b$

Table 2.4: Universal (Cardan) joint [16]

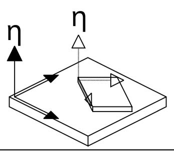
	Type of planar joint	Subset
	Planar joint perpendicular to X ($\eta = \mathbf{x}$)	${}^1b$, ${}^5b$, ${}^6b$
	Planar joint perpendicular to Y ($\eta = \mathbf{y}$)	${}^2b$, ${}^4b$, ${}^6b$
	Planar joint perpendicular to Z ($\eta = \mathbf{z}$)	${}^3b$, ${}^4b$, ${}^5b$

Table 2.5: Planar joint [16]

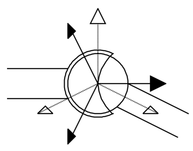
	Subset
	${}^1b$, ${}^2b$, ${}^3b$

Table 2.6: Spherical joint [16]

2.1.3 Constraint equations evaluation

Equations (2.9) − (2.14) are calculated from the vectors projected in the global coordinate system. To achieve this there is the need to define transformation matrix $\mathbf{T}_{i,i.a}$ between the centroidal coordinate system of body i and intermediate joint coordinate system of that body $i.a$ figure 2.1. These matrices are defined as

$$\mathbf{T}_{i,i.a} = \begin{pmatrix} & \mathbf{R}_{i,i.a}(\boldsymbol{\theta}_{i.a}) & & \{\mathbf{e}_{i,i.a}\}_i \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2.15}$$

where $\{\mathbf{e}_{i,i.a}\}_i$ is the position of the intermediate joint coordinate system with respect to the centroidal coordinate system of body i, and $\mathbf{R}_{i,i.a}$ represents the orientation of the intermediate joint coordinate system given by the Bryant angles

$$\mathbf{R}_{i,i.a}(\boldsymbol{\theta}_{i.a}) = \mathbf{T}_z(\psi_{i,i.a}) \cdot \mathbf{T}_{y'}(\theta_{i,i.a}) \cdot \mathbf{T}_{x''}(\phi_{i,i.a}). \tag{2.16}$$

But in this case $\boldsymbol{\theta}_{i.a}$ and $\{\mathbf{e}_{i,i.a}\}_i$ are given values of the orientation and position of the intermediate joint coordinate systems. These values are expressed in the centroidal coordinate system of body i.
Final transformation matrix between the global coordinate system and intermediate joint coordinate system $i.a$ is given by the dot product

$$\mathbf{T}_{0,i.a} = \mathbf{T}_{0,i} \cdot \mathbf{T}_{i,i.a} = \begin{pmatrix} \mathbf{x}_{i.a} & \mathbf{y}_{i.a} & \mathbf{z}_{i.a} & \mathbf{e}_{i.a} \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2.17}$$

where $\mathbf{x}_{i.a}$, $\mathbf{y}_{i.a}$ and $\mathbf{z}_{i.a}$ are the unit vectors of the axes of intermediate joint coordinate system $i.a$, and $\mathbf{e}_{i.a}$ is the position of the origin of this coordinate system. All vectors are written in the global coordinate system. At this moment it is possible to write down all necessary equations which could be solved. The problem is that the equations are transcendent and they needed to be solved numerically.

2.1.4 Derivative of constraint equations

For the purpose of numerical solution, it is important to define derivation of constraints. At first velocity $\mathbf{v}_i$ and angular velocity $\boldsymbol{\omega}_i$ of body i are defined. Let us consider

$$\mathbf{T}_{0,i}(\mathbf{q}_i) = \begin{pmatrix} & & & q_{6i-5} \\ & \mathbf{R}(q_{6i-2}, q_{6i-1}, q_{6i}) & & q_{6i-4} \\ & & & q_{6i-3} \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.18}$$

For the velocity of principal frame of body i holds

$$\{\mathbf{v}_i\}_0 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \underbrace{0}_{\mathbf{d}_{i,2}} & 1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \dot{q}_{6i-5} \\ \dot{q}_{6i-4} \\ \dot{q}_{6i-3} \\ \dot{q}_{6i-2} \\ \dot{q}_{6i-1} \\ \dot{q}_{6i} \end{pmatrix}, \tag{2.19}$$

where vector $\mathbf{d}_{i,k}$ can be expressed as

$$\mathbf{d}_{i,k} = \frac{\partial \mathbf{e}_i}{\partial q_k} = \frac{\partial \mathbf{v}_i}{\partial \dot{q}_k}. \tag{2.20}$$

Given the previous assumption and eq. (2.6), it is also possible to write

$$\boldsymbol{\omega}_i = \begin{pmatrix} 0 & 0 & 0 & \cos q_{6i} \cos q_{6i-1} & -\sin q_{6i} & 0 \\ 0 & 0 & 0 & \sin q_{6i} \cos q_{6i-1} & \cos q_{6i} & 0 \\ 0 & 0 & 0 & \underbrace{-\sin q_{6i-1}}_{\boldsymbol{\delta}_{i,4}} & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \dot{q}_{6i-5} \\ \dot{q}_{6i-4} \\ \dot{q}_{6i-3} \\ \dot{q}_{6i-2} \\ \dot{q}_{6i-1} \\ \dot{q}_{6i} \end{pmatrix}, \tag{2.21}$$

where vector $\boldsymbol{\delta}_{i,k}$ can be expressed as

$$\boldsymbol{\delta}_{i,k} = \frac{\partial \omega_i}{\partial \dot{q}_k}, \tag{2.22}$$

It is important to realise, that one constraint is defined always between 2 rigid bodies in this work. It means that each constraint must be derived with respect to the configuration parameters of the 2 bodies which are connected.

The consideration of the following equation 2.23 is also crucial for simulation and numerical solution. Prove of this equation is added in appendix A.1.

$$\frac{\partial^i b}{\partial q_k} = \frac{\partial\, {}^i\dot{b}}{\partial \dot{q}_k} \tag{2.23}$$

Derivative with respect to the first body

It is considered that q_k relates to body i, then the following relations hold:

$$ {}^1b = \mathbf{x}_{i.a} \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}), \tag{2.24}$$

$$\begin{aligned} \frac{d^1b}{dt} = {}^1\dot{b} &= \frac{d\mathbf{x}_{i.a}}{dt} \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) + \mathbf{x}_{i.a} \cdot \frac{d\mathbf{e}_{i.a}}{dt} - \mathbf{x}_{i.a} \cdot \frac{d\mathbf{e}_{j.b}}{dt} \\ &= (\boldsymbol{\omega}_{i.a} \times \mathbf{x}_{i.a}) \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) + \mathbf{x}_{i.a} \cdot \dot{\mathbf{e}}_{i.a} - \mathbf{x}_{i.a} \cdot \dot{\mathbf{e}}_{j.b}. \end{aligned} \tag{2.25}$$

Orientation vector $\mathbf{x}_{i.a}$ is composed of sines and cosines which gives only orientation of the intermediate joint coordinate system. Its derivation is then given by general known formula

$$\frac{\partial \mathbf{x}_{i.a}}{\partial t} = \boldsymbol{\omega}_{i.a} \times \mathbf{x}_{i.a}, \tag{2.26}$$

$$\begin{aligned} \frac{\partial^1\dot{b}}{\partial \dot{q}_k} &= \left(\frac{\partial \boldsymbol{\omega}_{i.a}}{\partial \dot{q}_k} \times \mathbf{x}_{i.a}\right) \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) + \mathbf{x}_{i.a} \cdot \frac{d\dot{\mathbf{e}}_{i.a}}{\partial \dot{q}_k} + 0, \\ &= (\boldsymbol{\delta}_{i.a,k} \times \mathbf{x}_{i.a}) \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) + \mathbf{x}_{i.a} \cdot \mathbf{d}_{i.a,k}. \end{aligned} \tag{2.27}$$

The frame velocities are given directly by

$$\mathbf{v}_{i.a} = \mathbf{v}_i + \boldsymbol{\omega}_i \times \mathbf{r}_{i.a}, \qquad \boldsymbol{\omega}_{i.a} = \boldsymbol{\omega}_i, \tag{2.28}$$

$$\mathbf{v}_{j.b} = \mathbf{v}_j + \boldsymbol{\omega}_j \times \mathbf{r}_{j.b}, \qquad \boldsymbol{\omega}_{j.b} = \boldsymbol{\omega}_j, \tag{2.29}$$

so

$$\mathbf{d}_{i.a,k} = \frac{\partial \mathbf{v}_{i.a,k}}{\partial \dot{q}_k} = \frac{\partial\left(\mathbf{v}_{i,k} + \boldsymbol{\omega}_{i,k} \times \mathbf{r}_{i.a}\right)}{\partial \dot{q}_k} = \tag{2.30}$$

$$= \frac{\partial \mathbf{v}_{i,k}}{\partial \dot{q}_k} + \frac{\partial \boldsymbol{\omega}_{i,k}}{\partial \dot{q}_k} \times \mathbf{r}_{i.a} = \mathbf{d}_{i,k} + \boldsymbol{\delta}_{i,k} \times (\mathbf{e}_{i.a} - \mathbf{e}_i) \tag{2.31}$$

All points on a rigid body experience the same angular velocity at all times. It is then possible to assume $\boldsymbol{\delta}_{i.a,k} = \boldsymbol{\delta}_{i,k}$, and the previous derivative and derivatives of ${}^2b$, ${}^3b$ with respect to the body i can be written as

$$\frac{\partial^1\dot{b}}{\partial \dot{q}_k} = (\boldsymbol{\delta}_{i,k} \times \mathbf{x}_{i.a}) \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) + \mathbf{x}_{i.a} \cdot \mathbf{d}_{i.a,k}, \tag{2.32}$$

$$\frac{\partial^2\dot{b}}{\partial \dot{q}_k} = (\boldsymbol{\delta}_{i,k} \times \mathbf{y}_{i.a}) \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) + \mathbf{y}_{i.a} \cdot \mathbf{d}_{i.a,k}, \tag{2.33}$$

$$\frac{\partial^3\dot{b}}{\partial \dot{q}_k} = (\boldsymbol{\delta}_{i,k} \times \mathbf{z}_{i.a}) \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) + \mathbf{z}_{i.a} \cdot \mathbf{d}_{i.a,k}. \tag{2.34}$$

Derivation of ${}^k b$ can be found analogously:

$$ {}^4 b = \mathbf{y}_{i.a} \cdot \mathbf{z}_{j.b}, \tag{2.35} $$

$$ \frac{d{}^4 b}{dt} = \frac{d\mathbf{y}_{i.a}}{dt} \cdot \mathbf{z}_{j.b} + \mathbf{y}_{i.a} \cdot \frac{d\mathbf{z}_{j.b}}{dt} \;=\; (\boldsymbol{\omega}_{i.a} \times \mathbf{y}_{i.a}) \cdot \mathbf{z}_{j.b} + \mathbf{y}_{i.a} \cdot (\boldsymbol{\omega}_{j.b} \times \mathbf{z}_{j.b}), \tag{2.36} $$

$$ \frac{\partial\, {}^4 \dot{b}}{\partial \dot{q}_k} = \frac{\partial}{\partial \dot{q}_k} \left[(\boldsymbol{\omega}_{i.a} \times \mathbf{y}_{i.a}) \cdot \mathbf{z}_{j.b} + \mathbf{y}_{i.a} \cdot (\boldsymbol{\omega}_{j.b} \times \mathbf{z}_{j.b}) \right] \;=\; (\boldsymbol{\delta}_{i.a,k} \times \mathbf{y}_{i.a}) \cdot \mathbf{z}_{j.b}. \tag{2.37} $$

With assumption that $\boldsymbol{\delta}_{i.a,k} = \boldsymbol{\delta}_{i,k}$ the derivatives of ${}^4 b$, ${}^5 b$ and ${}^6 b$ with respect to the body i can be written as

$$ \frac{\partial\, {}^4 \dot{b}}{\partial \dot{q}_k} = (\boldsymbol{\delta}_{i,k} \times \mathbf{y}_{i.a}) \cdot \mathbf{z}_{j.b}, \tag{2.38} $$

$$ \frac{\partial\, {}^5 \dot{b}}{\partial \dot{q}_k} = (\boldsymbol{\delta}_{i,k} \times \mathbf{z}_{i.a}) \cdot \mathbf{x}_{j.b}, \tag{2.39} $$

$$ \frac{\partial\, {}^6 \dot{b}}{\partial \dot{q}_k} = (\boldsymbol{\delta}_{i,k} \times \mathbf{x}_{i.a}) \cdot \mathbf{y}_{j.b}. \tag{2.40} $$

Derivative with respect to the second body

In consideration of q_k that relates to body j and with using the same steps as in the previous section. Let us consider

$$ {}^1 b = \mathbf{x}_{i.a} \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}), \tag{2.41} $$

$$ {}^1 \dot{b} = \dot{\mathbf{x}}_{i.a} \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) + \mathbf{x}_{i.a} \cdot (\dot{\mathbf{e}}_{i.a} - \dot{\mathbf{e}}_{j.b}), \tag{2.42} $$

$$ \frac{\partial^1 \dot{b}}{\partial \dot{q}_k} = \mathbf{x}_{i.a} \cdot \left(-\frac{\partial \dot{\mathbf{e}}_{j.b}}{\partial \dot{q}_k} \right), \tag{2.43} $$

$$ \frac{\partial^1 \dot{b}}{\partial \dot{q}_k} = -\mathbf{x}_{i.a} \cdot \mathbf{d}_{j.b,k}. \tag{2.44} $$

Then the derivatives of ${}^1 b$, ${}^2 b$ and ${}^3 b$ could be for the joint coordinate systems of the bodies i and j written as

$$ \frac{\partial^1 \dot{b}}{\partial \dot{q}_k} = -\mathbf{x}_{i.a} \cdot \mathbf{d}_{j.b,k}, \tag{2.45} $$

$$ \frac{\partial^2 \dot{b}}{\partial \dot{q}_k} = -\mathbf{y}_{i.a} \cdot \mathbf{d}_{j.b,k}, \tag{2.46} $$

$$ \frac{\partial^3 \dot{b}}{\partial \dot{q}_k} = -\mathbf{z}_{i.a} \cdot \mathbf{d}_{j.b,k}, \tag{2.47} $$

where

$$\mathbf{d}_{j.b,k} = \mathbf{d}_{j,k} + \boldsymbol{\delta}_{j,k} \times (\mathbf{e}_{j.b} - \mathbf{e}_j). \tag{2.48}$$

Derivation of ${}^k b$ can be again found analogously:

$$\begin{aligned}
{}^4 b &= \mathbf{y}_{i.a} \cdot \mathbf{z}_{j.b}, && (2.49)\\
{}^4 \dot{b} &= \frac{d\mathbf{y}_{i.a}}{dt} \cdot \mathbf{z}_{j.b} + \mathbf{y}_{i.a} \cdot \frac{d\mathbf{z}_{j.b}}{dt} \;=\; (\boldsymbol{\omega}_{i.a} \times \mathbf{y}_{i.a}) \cdot \mathbf{z}_{j.b} + \mathbf{y}_{i.a} \cdot (\boldsymbol{\omega}_{j.b} \times \mathbf{z}_{j.b}), && (2.50)\\
\frac{\partial\, {}^4 \dot{b}}{\partial \dot{q}_k} &= \mathbf{y}_{i.a} \cdot \left(\frac{\partial \boldsymbol{\omega}_{j.b,k}}{\partial \dot{q}_k} \times \mathbf{z}_{j.b} \right) \;=\; \mathbf{y}_{i.a} \cdot (\boldsymbol{\delta}_{j.b,k} \times \mathbf{z}_{j.b}). && (2.51)
\end{aligned}$$

With assumption that $\boldsymbol{\delta}_{j.b,k} = \boldsymbol{\delta}_{j,k}$, the derivatives of ${}^4 b$, ${}^5 b$ and ${}^6 b$ with respect to the body j can be written as

$$\frac{\partial\, {}^4 \dot{b}}{\partial \dot{q}_k} = \mathbf{y}_{i.a} \cdot (\boldsymbol{\delta}_{j,k} \times \mathbf{z}_{j.b}), \tag{2.52}$$

$$\frac{\partial\, {}^5 \dot{b}}{\partial \dot{q}_k} = \mathbf{z}_{i.a} \cdot (\boldsymbol{\delta}_{j,k} \times \mathbf{x}_{j.b}), \tag{2.53}$$

$$\frac{\partial\, {}^6 \dot{b}}{\partial \dot{q}_k} = \mathbf{x}_{i.a} \cdot (\boldsymbol{\delta}_{j,k} \times \mathbf{y}_{j.b}). \tag{2.54}$$

2.2 Solving equations

It is important to realise that the number of configuration parameters ($6 \times n_B$) is higher than number of constraints except for static and hyperstatic systems. E.g. a spatial pendulum connected to the ground by a revolute joint has 6 unknown configuration parameters and only 5 constraint equations, see table 2.2. For this reason n parameters are optional and have to be selected to solve kinematics of multibody systems. Then it is possible to write:

$$n = 6 \cdot n_B - n_C, \tag{2.55}$$

where n_C is the number of constraints. It is clear that the number of constraints could be counted as

$$n_C = 5 \cdot (n_{pr} + n_{re}) + 4 \cdot (n_{cy} + n_{un}) + 3 \cdot (n_{pl} + n_{sp}), \tag{2.56}$$

where n_{pr}, n_{re}, n_{cy}, n_{un}, n_{pl}, n_{sp} are numbers of prismatic, revolute, cylindrical, universal, planar and spherical joints respectively. In fact, the formulas for a quick calculation of mobility known in the literature and the presented relation 2.55 does yield correct results for many classical or modern mechanisms [23].

2.2.1 Numerical solution

The non-linear constraint equations can be solved by Newton-Raphson. The demonstration of this method consist in consideration of a system of equations $\mathbf{F}(\mathbf{x}) = \mathbf{0}$. If derivative of equations (constraints) exists, than the system $\mathbf{F}(\mathbf{x})$ can be expanded at the point $\mathbf{x} = \mathbf{x}^k$. With the use of a Taylor series it is possible to write

$$\mathbf{F}(\mathbf{x}) = \mathbf{F}(\mathbf{x}^k) + \mathbf{J}(\mathbf{x}^k)(\mathbf{x} - \mathbf{x}^k) + \frac{1}{2}\ddot{\mathbf{F}}(\boldsymbol{\xi})(\mathbf{x} - \mathbf{x}^k)^2, \tag{2.57}$$

where elements of $\mathbf{J}(\mathbf{x}^k)$ are defined as

$$J_{ij} = \frac{\partial F_i}{\partial q_j}. \tag{2.58}$$

The system of equations $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ can be approximated by a system of linear equations with $\mathbf{x}$ becomes only the next approximation to the root, giving us

$$\mathbf{F}(\mathbf{x}^k) + \mathbf{J}(\mathbf{x}^k)\underbrace{(\mathbf{x}^{k+1} - \mathbf{x}^k)}_{\mathbf{h}^k} = \mathbf{0}. \tag{2.59}$$

This can be rewritten as

$$\mathbf{J}(\mathbf{x}^k)\mathbf{h}^k = -\mathbf{F}(\mathbf{x}^k). \tag{2.60}$$

The solution of this equation can be expressed as

$$\mathbf{h}^k = -\mathbf{J}^{-1}(\mathbf{x}^k) \cdot \mathbf{F}(\mathbf{x}^k), \tag{2.61}$$

and a new iteration ($\mathbf{x}^{k+1}$) is obtained from the following expression

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{h}^k. \tag{2.62}$$

$\mathbf{J}(\mathbf{x}^k)$ is a Jacobian matrix of the matrix $\mathbf{F}(\mathbf{x})$ at the point ($\mathbf{x}^k$). The solution ends when

$$||\mathbf{x}^{k+1} - \mathbf{x}^k|| < \epsilon. \tag{2.63}$$

2.2.2 Jacobian matrix

The system of the constraint equations can be expressed

$$\mathbf{F}(\mathbf{x}) = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n_C} \end{pmatrix} = \mathbf{0}, \tag{2.64}$$

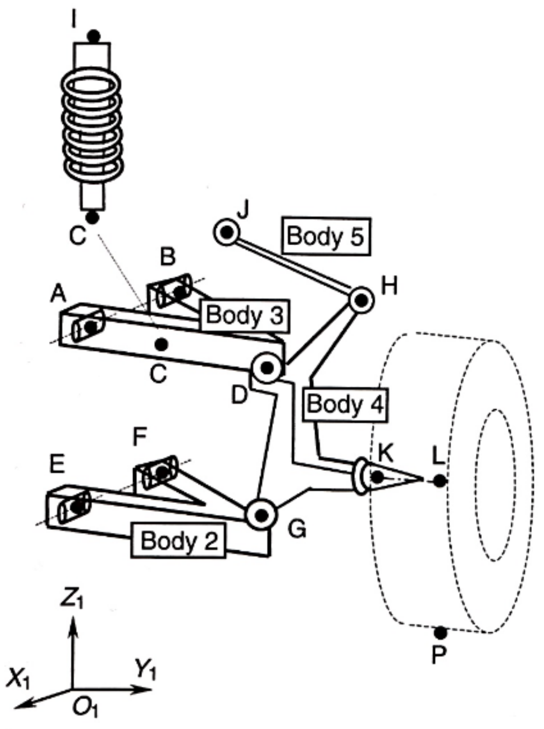
where (${}^j b \neq b_j$). Than the Jacobian matrix can be expressed as

$$\mathbf{J} = (\mathbf{x}) = \begin{pmatrix} \frac{\partial b_1}{\partial q_1} & \dots & \frac{\partial b_1}{\partial q_{n_{cp}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial b_{n_C}}{\partial q_1} & \dots & \frac{\partial b_{n_C}}{\partial q_{n_{cp}}} \end{pmatrix}, \tag{2.65}$$

where n_{cp} and n_C are the number of configuration parameters and the number of constraints respectively. At the end of this chapter necessary methods needed to solve kinematics of multibody systems with introduced joints are defined.

2.3 Application example

The system of double wishbone suspension was chosen to demonstrate kinematics solving. All necessary data are in figure 2.2 ([17], p. 177, 203).



POINT	X (mm)	Y (mm)	Z (mm)
$\underline{A}$	103	350	142
$\underline{B}$	−127	350	128
C	−12	491	104
D	−12	589	127
$\underline{E}$	122	345	−80
$\underline{F}$	−108	345	−80
G	7	620	−89
H	−156	545	178
I	−15	500	540
$\underline{J}$	−156	317	186
P	0	678	−265
K	0	600	0
L	0	678	0

Fig. 2.2: Double wishbone suspension example geometry data [17]. The underlined letters are fixed in space.

The system of car suspension has two degrees of freedom. Rotation of the wheel and its position and orientation which depend on the geometry of the suspension. The centroidal coordinate system of a car is located between the front wheels with the same orientation as in fig. 2.2. Every part of suspension (*body 1 - 6*, where *body 1* is the origin and body 6 is the wheel) has at least one intermediate joint coordinate system.

E.g. for body 1 three intermediate joint coordinate systems are defined. First is located in point B, second in point F and third in point J. There is no need to define secondary frames in points A and E, because it is enough to define only one revolute joint for each wishbone support (see fig. 2.2).

For example the coordinate system of *body 3* is located in point D. Its configuration in space is given by the transformation matrix $\mathbf{T}_{1,3}$ which depends on $q_{13} - q_{18}$. For example transformation matrix of the intermediate joint coordinate system in point

B, which is used to define revolute joint for upper arm, is defined

$$\mathbf{T}_{1,3.B} = \mathbf{T}_{1,3} \cdot \mathbf{T}_{3.B} = \mathbf{T}_{1,3} \cdot \mathbf{T}(\mathbf{B}, \mathbf{0}) \cdot \mathbf{T}_y(\theta), \tag{2.66}$$

where $\mathbf{B} = (B(1), B(2), B(3))$, $\mathbf{0} = (0, 0, 0)$ and θ is the direction of vector $\vec{AB} = [230 \;\; 0 \;\; 14]$, which is given by

$$\theta = \arctan \frac{14}{230}. \tag{2.67}$$

Transformation matrix between coordinate system of body 3 and intermediate joint coordinate system of body 3 in point B is defined as

$$\mathbf{T}_{3.B} = \begin{pmatrix} 1 & 0 & 0 & \vec{DB}(1) \\ 0 & 1 & 0 & \vec{DB}(2) \\ 0 & 0 & 1 & \vec{DB}(3) \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2.68}$$

where $\vec{DB} = B - D$. The two revolute joints and the only one universal joint are fixed to the car chassis or fixed in space. Due to this scheme and formulas (2.55) and (2.56) we got

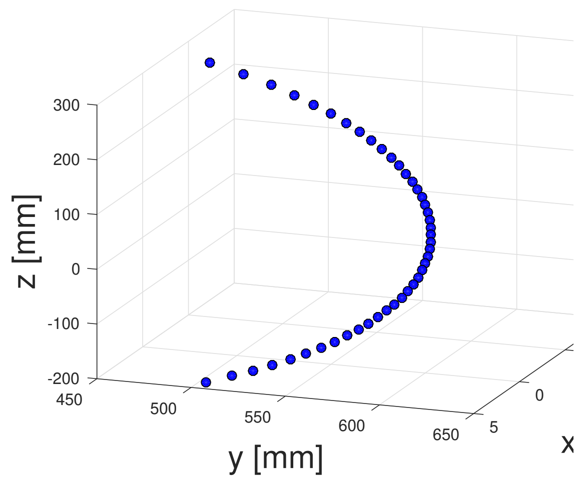
$$n_B = 5 \cdot (n_{pr} + n_{re}) + 4 \cdot (n_{cy} + n_{un}) + 3 \cdot (n_{pl} + n_{sp}) \tag{2.69}$$

2.4 Results and discussion

The two given parameters are the angle of the wheel rotation (about Y axis see fig. 2.2) and its z coordinate (vertical), which were evaluated from -170 to 240 mm with a step of 10 mm. All higher or lower positions did not converge because the solution does not exist due to the limitations of the suspension geometry. Orientation of an arbitrary body is the same for all points of that body in case of rigid bodies. Rotations ψ, θ and ϕ are given as rotations around Z,Y' and X" axis due to Bryant angles.

All of the coordinates of point K are continuous due to given coordinate z and all of the calculated points converged quickly (6 to 10 iterations). With $\epsilon = 10^{-13}$. The numerical solution is stable and results can be used in the next work. The Matlab program is written universally and can be use for different types of suspensions.

Figure 2.3 shows the position of point K from figure 2.2. Figures 2.4a $-$ 2.7 show the dependence of x, y, ϕ, θ and ψ on given parameter z.



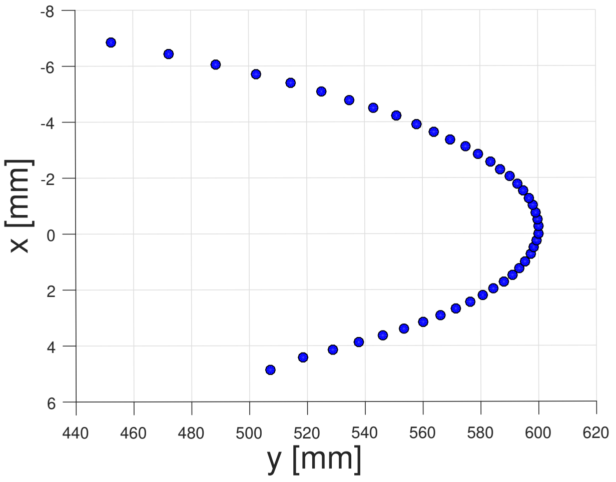
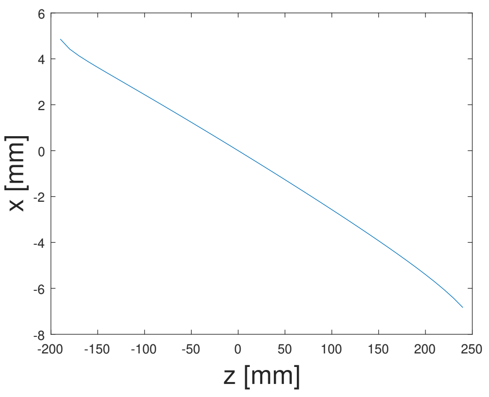
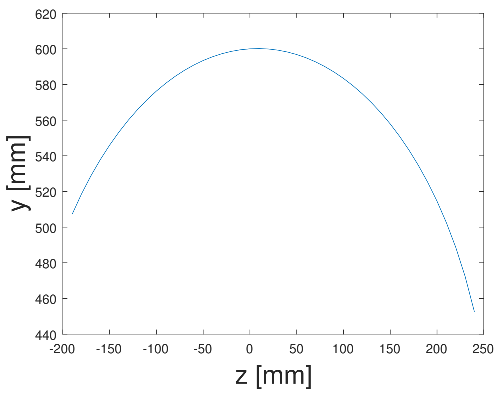


Fig. 2.3: Position of the wheel support (point K - fig. 2.2) in 3D [mm]



(a) x coordinate on z



(b) y coordinate on z

Fig. 2.4: Free coordinates of the point K (fig. 2.2) depending on the z coordinate

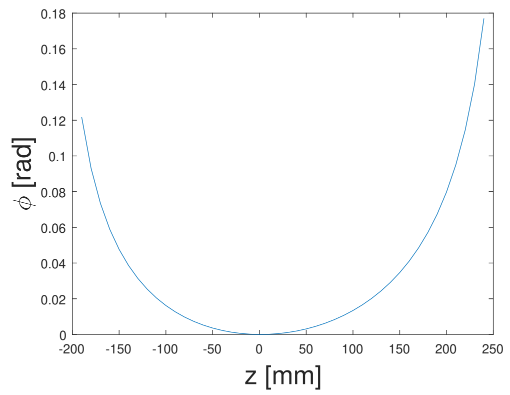


Fig. 2.5: ϕ coordinate on z

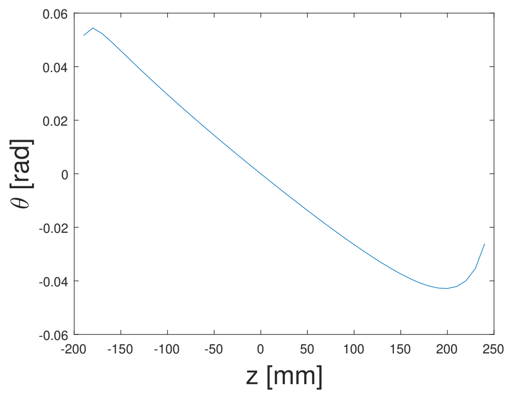


Fig. 2.6: θ coordinate on z

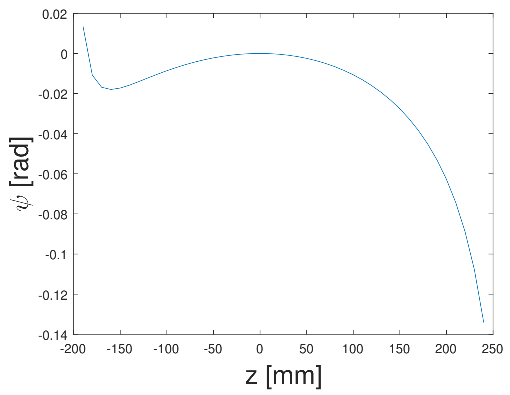


Fig. 2.7: ψ coordinate on z

Chapter 3

Kinematic interpolation

This chapter aim is to produce interpolated kinematics from a table created in the previous chapter on the level of position, velocity and acceleration in terms of the chosen parameter (one of six parameters which are defined for each body). A proposed method ensures smoothness on the velocity level and continuity on the level of acceleration. Hermite splines are used for interpolation terms; the interpolation is used for the evaluation of translations and rotations and their derivatives. The interpolation for rotations is implemented through successive rotations about specific axes. Error analysis, which discusses the norm of the differences between interpolated and exactly computed values, is also presented.

3.1 Input data

The data which are counted in the previous chapter are important for the interpolation. The interpolation assumes that the evolution of homogeneous transformation matrix $\mathbf{T}_{0,A}$ and its derivatives $\mathbf{d}_{A,u}$ and $\boldsymbol{\delta}_{A,u}$ have been calculated for some values of a parameter u, where vectors $\mathbf{d}_{A,u}$ and $\boldsymbol{\delta}_{A,u}$ are defined as [16]

$$\mathbf{d}_{A,u} = \frac{\partial \mathbf{e}_A}{\partial u} = \frac{\partial \mathbf{v}_A}{\partial \dot{u}}, \quad \boldsymbol{\delta}_{A,u} = \frac{\partial \boldsymbol{\omega}_A}{\partial \dot{u}}. \tag{3.1}$$

First, it is useful to store the data into the table in terms of a series of values u $(u_0,\ u_1,\ u_2,\ ...,u_N\ )$

$$\begin{array}{cccc}
u_0, & \mathbf{T}_{0,A}(u_0), & \{\mathbf{d}_{A,u}(u_0)\}_0, & \{\boldsymbol{\delta}_{A,u}(u_0)\}_0, \\
u_1, & \mathbf{T}_{0,A}(u_1), & \{\mathbf{d}_{A,u}(u_1)\}_0, & \{\boldsymbol{\delta}_{A,u}(u_1)\}_0, \\
u_2, & \mathbf{T}_{0,A}(u_2), & \{\mathbf{d}_{A,u}(u_2)\}_0, & \{\boldsymbol{\delta}_{A,u}(u_2)\}_0, \\
\vdots & \vdots & \vdots & \vdots \\
u_N, & \mathbf{T}_{0,A}(u_N), & \{\mathbf{d}_{A,u}(u_N)\}_0, & \{\boldsymbol{\delta}_{A,u}(u_N)\}_0.
\end{array}$$

with

$$\{\mathbf{v}_A\}_0 = \{\mathbf{d}_A\}_0 \cdot \dot{u} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{Bmatrix} \dot{q}_{A1} \\ \dot{q}_{A2} \\ \dot{q}_{A3} \\ \dot{q}_{A4} \\ \dot{q}_{A5} \\ \dot{q}_{A6} \end{Bmatrix}, \tag{3.2}$$

and

$$\{\boldsymbol{\omega}_A\}_0 = \{\boldsymbol{\delta}_A\}_0 \cdot \dot{u} = \begin{pmatrix} 0 & 0 & 0 & \cos(q_{A6})\cos(q_{A5}) & -\sin(q_{A6}) & 0 \\ 0 & 0 & 0 & \sin(q_{A6})\cos(q_{A5}) & \cos(q_{A6}) & 0 \\ 0 & 0 & 0 & -\sin(q_{A6}) & 0 & 1 \end{pmatrix} \cdot \begin{Bmatrix} \dot{q}_{A1} \\ \dot{q}_{A2} \\ \dot{q}_{A3} \\ \dot{q}_{A4} \\ \dot{q}_{A5} \\ \dot{q}_{A6} \end{Bmatrix}, \tag{3.3}$$

where $\mathbf{q}_A$ is a vector which was counted in the previous chapter as a result of Newton-Raphson method.

To get vector $\dot{\mathbf{q}}_A$, it is necessary to compute

$$\mathbf{B} \cdot \dot{q} = 0, \tag{3.4}$$

$$\dot{u} = 1, \tag{3.5}$$

where u, $\dot{u}$ is given parameter and its time derivative and $\mathbf{B}$ is the Jacobian matrix of the constraint equation system. It is also important to place $\dot{u}$ in the right place in the system of equations, because it substitutes the velocity of the chosen parameter.

If u replaces q_i, it is possible to rewrite previous two equations to

$$\begin{pmatrix} b_{11} & b_{12} & \dots & b_{1(i-1)} & b_{1(i+1)} & \dots & b_{1,n} & 0 \\ b_{21} & b_{22} & \dots & b_{2(i-1)} & b_{2(i+1)} & \dots & b_{2,n} & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & 0 \\ b_{n1} & b_{n2} & \dots & b_{n(i-1)} & b_{n(i+1)} & \dots & b_{n,n} & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 & 1 \end{pmatrix} \cdot \begin{Bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_{i-1} \\ \dot{q}_{i+1} \\ \vdots \\ \dot{q}_n \\ \dot{u} \end{Bmatrix} = \begin{Bmatrix} 0 \\ \vdots \\ 1 \end{Bmatrix} \tag{3.6}$$

Because we assume that $\dot{u} = 1$, then

$$\{\mathbf{v}_A\}_0 = \{\mathbf{d}_A\}_0, \tag{3.7}$$
$$\{\boldsymbol{\omega}_A\}_0 = \{\boldsymbol{\delta}_A\}_0. \tag{3.8}$$

3.2 Cubic Hermite splines

Cubic Hermite splines are typically used for interpolation of discrete numeric data specified at given argument values, u_0, u_1, u_2, , . . . , u_N, to obtain a smooth continuous function. The splines are also used as shape functions to model beams in finite element methods, where they pass through specified points of the plane or three-dimensional space. The spline functions are

$$h_{00}(\xi) = 2\xi^3 - 3\xi^2 + 1, \tag{3.9}$$
$$h_{01}(\xi) = -2\xi^3 + 3\xi^2, \tag{3.10}$$
$$h_{10}(\xi) = \xi^3 - 2\xi^2 + \xi, \tag{3.11}$$
$$h_{11}(\xi) = \xi^3 - \xi^2 \tag{3.12}$$

and they are plotted in figure 3.1. The first and second derivatives with respect to ξ can be expressed

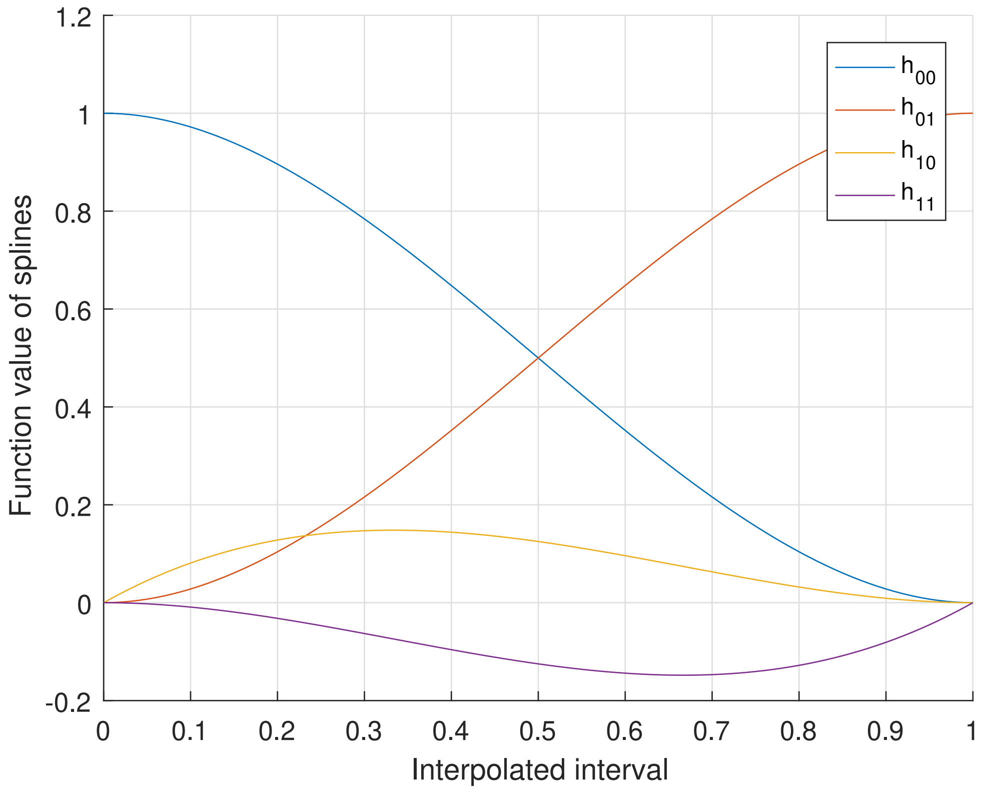


Fig. 3.1: Cubic Hermite splines

$$
\begin{aligned}
h_{00}^{'}(\xi) &= \frac{dh_{00}}{d\xi} = 6\xi^2 - 6\xi, & h_{00}^{''}(\xi) &= \frac{d^2h_{00}}{d\xi^2} = 12\xi - 6,\\
h_{01}^{'}(\xi) &= \frac{dh_{01}}{d\xi} = -6\xi^2 + 6\xi, & h_{01}^{''}(\xi) &= \frac{d^2h_{01}}{d\xi^2} = -12\xi + 6,\\
h_{10}^{'}(\xi) &= \frac{dh_{10}}{d\xi} = 3\xi^2 - 4\xi + 1, & h_{10}^{''}(\xi) &= \frac{d^2h_{10}}{d\xi^2} = 6\xi - 4,\\
h_{11}^{'}(\xi) &= \frac{dh_{11}}{d\xi} = 3\xi^2 - 2\xi, & h_{11}^{''}(\xi) &= \frac{d^2h_{11}}{d\xi^2} = 6\xi - 2. \qquad (3.13)
\end{aligned}
$$

After substituting of $\xi = 0$ and $\xi = 1$

$$\begin{aligned} &h_{00}(0) = 1, \quad h_{00}(1) = 0, \quad h'_{00}(0) = 0, \quad h'_{00}(1) = 0, \\ &h_{01}(0) = 0, \quad h_{01}(1) = 1, \quad h'_{01}(0) = 0, \quad h'_{01}(1) = 0, \\ &h_{10}(0) = 0, \quad h_{10}(1) = 0, \quad h'_{10}(0) = 1, \quad h'_{10}(1) = 0, \\ &h_{11}(0) = 0, \quad h_{11}(1) = 0, \quad h'_{11}(0) = 0, \quad h'_{11}(1) = 1. \end{aligned} \tag{3.14}$$

Now it is possible to interpolate an arbitrary function $f = f(x)$ between x_0 and x_1 with respect to boundary conditions

$$f(x_0) = f_0, \quad f(x_1) = f_1, \quad f'(x_0) = m_0, \quad f'(x_1) = m_1$$

by

$$\begin{aligned} f(x) = f_0\, h_{00}\left(\frac{x - x_0}{x_1 - x_0}\right) + f_1\, h_{01}\left(\frac{x - x_0}{x_1 - x_0}\right) + \\ + m_0\, h_{10}\left(\frac{x - x_0}{x_1 - x_0}\right) \cdot (x_1 - x_0) + m_1\, h_{11}\left(\frac{x - x_0}{x_1 - x_0}\right) \cdot (x_1 - x_0) \end{aligned} \tag{3.15}$$

from which it is easy to verify that the value of $f(x)$ is equal to f_0 when $x = x_0$ and to f_1 when $x = x_1$.
The first derivative of the interpolant 3.15 with respect to x of the interpolation is defined consequently by

$$\begin{aligned} f'(x) = f_0\, h'_{00}\left(\frac{x - x_0}{x_1 - x_0}\right) \cdot \frac{1}{x_1 - x_0} + f_1\, h'_{01}\left(\frac{x - x_0}{x_1 - x_0}\right) \cdot \frac{1}{x_1 - x_0} + \\ + m_0\, h'_{10}\left(\frac{x - x_0}{x_1 - x_0}\right) + m_1\, h'_{11}\left(\frac{x - x_0}{x_1 - x_0}\right), \end{aligned} \tag{3.16}$$

from which it is easy to verify that the value of $f'(x)$ is equal to m_0 when $x = x_0$ and to m_1 when $x = x_1$.
It is also possible to write the second derivative with respect to x as

$$\begin{aligned} f''(x) = f_0\, h''_{00}\left(\frac{x - x_0}{x_1 - x_0}\right) \cdot \frac{1}{(x_1 - x_0)^2} + f_1\, h''_{01}\left(\frac{x - x_0}{x_1 - x_0}\right) \cdot \frac{1}{(x_1 - x_0)^2} + \\ + m_0\, h''_{10}\left(\frac{x - x_0}{x_1 - x_0}\right) \cdot \frac{1}{x_1 - x_0} + m_1\, h''_{11}\left(\frac{x - x_0}{x_1 - x_0}\right) \cdot \frac{1}{x_1 - x_0}. \end{aligned} \tag{3.17}$$

3.3 Interpolation of position

If we assume that $u_i \leq u \leq u_{i+1}$ and if we define ξ by

$$\xi = \frac{u - u_i}{u_{i+1} - u_i}, \tag{3.18}$$

then cubic Hermite splines can be used straightforward, to interpolate the position part of the homogeneous transformation matrix:

$$\begin{aligned}\mathbf{e}_A(u) = \mathbf{e}_A(u_i)h_{00}(\xi) + \mathbf{e}_A(u_{i+1})h_{01}(\xi) + \mathbf{d}_{A,u}(u_i)h_{10}(\xi) \cdot (u_{i+1} - u_i)\\ + \mathbf{d}_{A,u}(u_{i+1})h_{11}(\xi) \cdot (u_{i+1} - u_i).\end{aligned} \tag{3.19}$$

The derivative of the position with respect to u is given by

$$\begin{aligned}\frac{d\mathbf{e}_A}{du}(u) = \mathbf{d}_{A,u}(u) = \mathbf{e}_A(u_i)h_{00}^{'}(\xi) \cdot \frac{1}{u_{i+1} - u_i} + \mathbf{e}_A(u_{i+1})h_{01}^{'}(\xi)\frac{1}{u_{i+1} - u_i}\\ + \mathbf{d}_{A,u}(u_i)h_{10}^{'}(\xi) + \mathbf{d}_{A,u}(u_{i+1})h_{11}^{'}(\xi).\end{aligned} \tag{3.20}$$

When $u = u_i$ than

$$\mathbf{e}_A(u) = \mathbf{e}_A(u_i), \tag{3.21}$$

$$\mathbf{d}_{A,u}(u) = \mathbf{d}_{A,u}(u_i), \tag{3.22}$$

and when $u = u_{i+1}$ than

$$\mathbf{e}_A(u) = \mathbf{e}_A(u_{i+1}) \tag{3.23}$$

$$\mathbf{d}_{A,u}(u) = \mathbf{d}_{A,u}(u_{i+1}). \tag{3.24}$$

Practically, the first derivative is used to compute the velocity of point A

$$\mathbf{v}_A = \frac{d\mathbf{e}_A}{dt} = \frac{\partial \mathbf{e}_A}{\partial u}\frac{du}{dt} = \mathbf{d}_{A,u}(u)\dot{u}. \tag{3.25}$$

The second time derivative of the position with respect to u can be calculated as well

$$\begin{aligned}\frac{\partial^2 \mathbf{e}_A}{\partial u^2} = \frac{\partial \mathbf{d}_{A,u}}{\partial u}(u) = \mathbf{e}_A(u_i)h_{00}^{''}(\xi) \cdot \frac{1}{(u_{i+1} - u_i)^2} + \mathbf{e}_A(u_{i+1})h_{01}^{''}(\xi)\frac{1}{(u_{i+1} - u_i)^2}\\ + \mathbf{d}_{A,u}(u_i)h_{10}^{''}(\xi) \cdot \frac{1}{u_{i+1} - u_i} + \mathbf{d}_{A,u}(u_{i+1})h_{11}^{''}(\xi) \cdot \frac{1}{u_{i+1} - u_i}.\end{aligned} \tag{3.26}$$

and is used to compute the acceleration of point A:

$$\begin{aligned}\mathbf{a}_A = \frac{d\mathbf{v}_A}{dt} &= \frac{\partial \mathbf{e}_A}{\partial u}\frac{d^2u}{dt^2} + \frac{\partial^2 \mathbf{e}_A}{\partial u^2}\left(\frac{du}{dt}\right)^2\\ &= \mathbf{d}_{A,u}\frac{d^2u}{dt^2} + \frac{\partial \mathbf{d}_{A,u}}{\partial u}\left(\frac{du}{dt}\right)^2 = \mathbf{d}_{A,u}\ddot{u} + \frac{\partial \mathbf{d}_{A,u}}{\partial u}\dot{u}^2.\end{aligned} \tag{3.27}$$

3.4 Interpolation of rotation

The purpose of this section is to interpolate the rotation matrix. Although the rotations cannot be summed, it is possible to find the interpolation of rotation between two steps (u_i and u_{i+1}). The rotation matrices in two successive configurations i and $i+1$ be $\mathbf{R}_{0,A}(u_i)$ and $\mathbf{R}_{0,A}(u_{i+1})$, respectively. These matrices fulfil the following relation:

$$\mathbf{R}_{0,A}(u_{i+1}) = \mathbf{R}_{0,A}(u_i) \cdot \mathbf{R}_{0,A}^{i,i+1}. \tag{3.28}$$

First, it is necessary to isolate the relative rotation matrix $\mathbf{R}_{0,A}^{i,i+1}$ between the two configurations so that

$$\mathbf{R}_{0,A}^{i,i+1} = \mathbf{R}_{0,A}^{-1}(u_i) \cdot \mathbf{R}_{0,A}(u_{i+1}) = \mathbf{R}_{0,A}^{T}(u_i) \cdot \mathbf{R}_{0,A}(u_{i+1}). \tag{3.29}$$

The interpolation of rotation is then based on the relative rotation matrix $\mathbf{R}_{0,A}^{int}$, where

$$\mathbf{R}_{0,A}(u) = \mathbf{R}_{0,A}(u_i) \cdot \mathbf{R}_{0,A}^{int}(u), \tag{3.30}$$

so in respect of the boundary conditions

$$\begin{aligned} \mathbf{R}_{0,A}(u_i) &= \mathbf{R}_{0,A}(u_i) \quad \rightarrow \quad \mathbf{R}_{0,A}^{int}(u_i) = \mathbf{I}, & (3.31)\\ \mathbf{R}_{0,A}(u_{i+1}) &= \mathbf{R}_{0,A}(u_{i+1}) \quad \rightarrow \quad \mathbf{R}_{0,A}^{int}(u_{i+1}) = \mathbf{R}_{0,A}^{i,i+1}. & (3.32) \end{aligned}$$

but also

$$\left[\tilde{\boldsymbol{\delta}}_{A,u}(u_i)\right]_0 = \frac{d\mathbf{R}_{0,A}}{du}(u_i)\,\mathbf{R}_{0,A}^{T}(u_i) \quad \rightarrow \left[\tilde{\boldsymbol{\delta}}_{A,u}(u_i)\right]_i = \frac{d\mathbf{R}_{0,A}^{int}}{du}(u_i)\,\mathbf{R}_{0,A}^{int^T}(u_i), \tag{3.33}$$

$$\left[\tilde{\boldsymbol{\delta}}_{A,u}(u_i)\right]_0 = \frac{d\mathbf{R}_{0,A}}{du}(u_{i+1})\,\mathbf{R}_{0,A}^{T}(u_{i+1}) \rightarrow \left[\tilde{\boldsymbol{\delta}}_{A,u}(u_{i+1})\right]_{i+1} = \frac{d\mathbf{R}_{0,A}^{int}}{du}(u_{i+1})\,\mathbf{R}_{0,A}^{int^T}(u_{i+1}), \tag{3.34}$$

with

$$[\tilde{\mathbf{a}}]_0 = \begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{pmatrix} \quad if \quad \{\mathbf{a}\}_0 = \begin{Bmatrix} a_x \\ a_y \\ a_z \end{Bmatrix}. \tag{3.35}$$

It is possible to express every spatial rotation as the rotation of the angle θ around a unit vector $\mathbf{n}$ which define $\mathbf{Rrotaxis}(\mathbf{n}, \theta)$ that is given by

$$\mathbf{Rrotaxis}(\mathbf{n}, \theta) = \begin{pmatrix} n_x^2 V + C & n_x n_y V - n_z S & N_x n_z V + n_y S \\ n_x n_y V + n_z S & n_y^2 V + C & n_y n_z V - n_x S \\ n_x n_z V - n_y S & n_y n_z V + n_x S & n_z^2 V + C \end{pmatrix}, \tag{3.36}$$

where $C = \cos(\theta)$, $S = \sin(\theta)$ and $V = 1 - \cos(\theta)$. In latter expression, $\mathbf{n}$ can be projected indifferently in the departure coordinate system ($\mathbf{T}_{0,A}(u_i)$) or the arrival coordinate system ($\mathbf{T}_{0,A}(u_{i+1})$) and even in any intermediary coordinate system during the rotation ($\mathbf{T}_{0,A}(u)$, where $u_i \leq u \leq u_{i+1}$). The direction of the rotation axis is invariant during the rotation and has the same projection in both frames.

Similarly, any rotation matrix ($\mathbf{R}_{0,A}^{i,i+1}$) can be used to establish the axis and the angle to which it corresponds. The angle is given directly by

$$\theta = \arccos\left(\frac{r_{11} + r_{22} + r_{33} - 1}{2}\right), \tag{3.37}$$

which assumes that $0 < \theta < \pi$.

The unit vector parallel to the axis can then be obtained by (if $\sin\theta \neq 0$),

$$\{\mathbf{n}^{i,i+1}\}_i = \frac{1}{2\sin\theta} \cdot \begin{Bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{Bmatrix}. \tag{3.38}$$

This information contained in eigenvalues and eigenvectors of $\mathbf{R}_{0,A}^{i,i+1}$. An orthonormal rotation matrix ($\mathbf{R}_{0,A}^{i,i+1}$) will always have one real eigenvalue $\lambda_1 = 1$ and one complex pair $\lambda_{2,3} = \cos\theta \pm i\sin\theta$, where θ is the rotation angle. From the definition of eigenvalues and eigenvectors we recall that

$$\mathbf{R}\mathbf{v}_i = \lambda_i \mathbf{v}, \tag{3.39}$$

where $\mathbf{v}_i$ is the eigenvector corresponding to λ_i. For case $\lambda_1 = 1$ then

$$\mathbf{R}\mathbf{v}_1 = \mathbf{v}_1, \tag{3.40}$$

which implies that the corresponding eigenvector $\mathbf{v}$ is *unchanged* by the rotation. There is only one such vector and that is the one about which the rotation occurs [20]. In other words, the direction of the rotation axis is the eigenvector of the rotation matrix corresponding to the eigenvalue $\lambda_1 = 1$.

In addition, it is important to remark that:

- if $\theta = 0$, the unit axis vector is arbitrary,
- if $\theta = \pi$, the formula for $\mathbf{n}$ can be retrieved easily from the matrix; its sense is arbitrary and
- the same rotation matrix is obtained with $-\theta$ and the opposite axis.

For the practical application, the relative rotation matrix is defined by the axis and angle corresponding to the rotation from the configuration when $u = u_i$ to the one when $u = u_{i+1}$:

$$\mathbf{R}_{0,A}^{i,i+1} = \mathbf{Rrotaxis}(\{\mathbf{n}^{i,i+1}\}_i, \theta^{i,i+1}). \tag{3.41}$$

It means the rotation from i to $i+1$ corresponds to a rotation through angle $\theta^{i,i+1}$ about axis $\{\mathbf{n}^{i,i+1}\}_i$. The axis unit vector $\{\mathbf{n}^{i,i+1}\}_i$ has been projected in coordinate system i but could have been projected into $i+1$ as well.

As it was written in the previous section, ξ is still defined by

$$\xi = \frac{u - u_i}{u_{i+1} - u_i}, \tag{3.42}$$

hence it is possible to calculate $\mathbf{R}_{0,A}^{int}(u)$ from the following expression

$$\mathbf{R}_{0,A}^{int}(u) = \mathbf{R}^1(u) \cdot \mathbf{R}^2(u) \cdot \mathbf{R}^3(u), \tag{3.43}$$

where matrices $\mathbf{R}^1(u)$, $\mathbf{R}^2(u)$ and $\mathbf{R}^3(u)$ correspond to

$$\mathbf{R}^1(\xi(u)) = \mathbf{Rrotaxis}(\frac{\{\boldsymbol{\delta}_{A,u}(u_i)\}_i}{\|\boldsymbol{\delta}_{A,u}(u_i)\|},\ \|\boldsymbol{\delta}_{A,u}(u_i)\| \cdot h_{10}(\xi) \cdot (u_{i+1} - u_i)), \tag{3.44}$$

$$\mathbf{R}^2(\xi(u)) = \mathbf{Rrotaxis}(\{\mathbf{n}_{A,u}^{i,i+1})\}_i,\ \theta^{i,i+1} \cdot h_{01}(\xi)), \tag{3.45}$$

$$\mathbf{R}^3(\xi(u)) = \mathbf{Rrotaxis}(\frac{\{\boldsymbol{\delta}_{A,u}(u_{i+1})\}_{i+1}}{\|\boldsymbol{\delta}_{A,u}(u_{i+1})\|},\ \|\boldsymbol{\delta}_{A,u}(u_{i+1})\| \cdot h_{11}(\xi) \cdot (u_{i+1} - u_i)). \tag{3.46}$$

However, vectors $\{\boldsymbol{\delta}_{A,u}(u_i)\}_0$ and $\{\boldsymbol{\delta}_{A,u}(u_{i+1})\}_0$ vectors are given from their coordinates in the global coordinate system ($\{\boldsymbol{\delta}_{A,u}(u_0)\}_0$). In the latter expressions, some $\boldsymbol{\delta}$ vectors have to be transferred to other coordinate systems

$$\{\boldsymbol{\delta}_{A,u}(u_i)\}_i = \mathbf{R}_{0,A}^T(u_i) \cdot \{\boldsymbol{\delta}_{A,u}(u_i)\}_0, \tag{3.47}$$

$$\{\boldsymbol{\delta}_{A,u}(u_{i+1})\}_{i+1} = \mathbf{R}_{0,A}^T(u_{i+1}) \cdot \{\boldsymbol{\delta}_{A,u}(u_{i+1})\}_0. \tag{3.48}$$

To sum up briefly, the full orientation matrix will then be calculated from

$$\mathbf{R}_{0,A}(u) = \mathbf{R}_{0,A}(u_i) \cdot \mathbf{R}_{0,A}^{int}(u) = \mathbf{R}_{0,A}(u_i) \cdot \mathbf{R}^1(u) \cdot \mathbf{R}^2(u) \cdot \mathbf{R}^3(u). \tag{3.49}$$

The angular velocity is given by summation of the contributions of the 3 successive rotations

$$\boldsymbol{\omega}_A = \boldsymbol{\delta}_{A,u}(u)\,\dot{u}, \tag{3.50}$$

with

$$\{\boldsymbol{\delta}_A(u)\}_0 = \{\boldsymbol{\delta}_1\}_0\, h_{10}^{'}(\xi) + \{\boldsymbol{\delta}_2(u)\}_0\, h_{01}^{'}(\xi)\frac{\theta^{i,i+1}}{u_{i+1}-u_i} + \{\boldsymbol{\delta}_3(u)\}_0\, h_{11}^{'}(\xi), \tag{3.51}$$

where

$$\{\boldsymbol{\delta}_1\}_0 = \{\boldsymbol{\delta}_{A,u}(u_i)\}_0, \tag{3.52}$$
$$\{\boldsymbol{\delta}_2(u)\}_0 = \mathbf{R}_{0,A}(u_i)\cdot\mathbf{R}^1(u)\cdot\{\mathbf{n}^{i,i+1}\}_i, \tag{3.53}$$
$$\{\boldsymbol{\delta}_3(u)\}_0 = \mathbf{R}_{0,A}(u_i)\cdot\mathbf{R}^1(u)\cdot\mathbf{R}^2(u)\cdot\{\boldsymbol{\delta}_{A,u}(u_{i+1})\}_{i+1}. \tag{3.54}$$

Finally, the angular acceleration, as the translational one, will consist of 2 contributions

$$\frac{d\boldsymbol{\omega}_A}{dt} = \boldsymbol{\delta}_{A,u}(u)\,\ddot{u} + \frac{d\boldsymbol{\delta}_{A,u}(u)}{du}\dot{u}^2, \tag{3.55}$$

where $\boldsymbol{\delta}_{A,u}$ is defined in eq. 3.51 and $\dfrac{d\boldsymbol{\delta}_{A,u}(u)}{du}$ is given by

$$\frac{d\boldsymbol{\delta}_{A,u}(u)}{du} = \frac{d\boldsymbol{\delta}_1(u)}{du} + \frac{d\boldsymbol{\delta}_2(u)}{du} + \frac{d\boldsymbol{\delta}_3(u)}{du}. \tag{3.56}$$

The latter are written

$$\left\{\frac{d\boldsymbol{\delta}_1(u)}{du}\right\}_0 = \{\boldsymbol{\delta}_{A,u}(u_i)\}_0\, h_{10}^{''}(\xi)\frac{1}{u_{i+1}-u_i}, \tag{3.57}$$
$$\left\{\frac{d\boldsymbol{\delta}_2(u)}{du}\right\}_0 = \mathbf{R}_{0,A}(u_i)\cdot\mathbf{R}^1(u)\cdot\left\{\mathbf{n}^{i,i+i}\right\}_i h_{01}^{''}(\xi)\frac{\theta^{i,i+1}}{(u_{i+1}-u_i)^2} + \{\boldsymbol{\delta}_1\times\boldsymbol{\delta}_2\}_0\,, \tag{3.58}$$
$$\left\{\frac{d\boldsymbol{\delta}_3(u)}{du}\right\}_0 = \mathbf{R}_{0,A}(u_i)\cdot\mathbf{R}^1(u)\cdot\mathbf{R}^2(u)\cdot\{\boldsymbol{\delta}_{A,u}(u_{i+1})\}_{i+1} h_{11}^{''}(\xi)\frac{1}{u_{i+1}-u_i} + \{(\boldsymbol{\delta}_1+\boldsymbol{\delta}_2)\times\boldsymbol{\delta}_3\}_0\,, \tag{3.59}$$

in which term

$$\{(\boldsymbol{\delta}_1+\boldsymbol{\delta}_2)\times\boldsymbol{\delta}_3\}_0 \tag{3.60}$$

corresponds to the complementary (or Résal) angular acceleration that results from the variation of the direction of the rotation axis.

3.5 Error analysis

There are compared the exact position (of a wheel or a sample for verifying the interpolation method) with the interpolated one in the following sections. The most important to compare in terms of errors is the evolution of:

- the absolute error in translation (norm of the distance vector between exact position ($\mathbf{e}_{ex}$) and interpolated one ($\mathbf{e}_{int}$),

$$e_e = ||\mathbf{e}_{ex} - \mathbf{e}_{int}||, \tag{3.61}$$

- the norm of the vector difference between the $\mathbf{d}_{A,u}$ vectors,

$$d_e = ||\mathbf{d}_{ex} - \mathbf{d}_{int}||, \tag{3.62}$$

- the norm of the vector difference between the $\boldsymbol{\delta}_{A,u}$ vectors,

$$\delta_e = ||\boldsymbol{\delta}_{ex} - \boldsymbol{\delta}_{int}||, \tag{3.63}$$

- the error in rotation θ_e.

The error in rotation (angle of the relative rotation) is computed from the formula given for the transformation from rotation matrix to axis-angle representation.

$$\mathbf{R}_{i,e} = \mathbf{R}_i^T \cdot \mathbf{R}_e, \tag{3.64}$$

where $\mathbf{R}_{i,e}$ is the relative rotation matrix between interpolated and exact position, $\mathbf{R}_i$ and $\mathbf{R}_e$ are interpolated and exact rotation matrices. Error in rotation is then given by formula (3.37), applied to $\mathbf{R}_{i,e}$.
In cases of suspensions we also study:

- the norm in translation between two steps

$$E_n(u_i, u_{i+1}) = ||\mathbf{e}(u_i) - \mathbf{e}(u_{i+1})||, \tag{3.65}$$

- the rotation angle between two steps Θ_n,

$$\mathbf{R}_a(u_i, u_{i+1}) = \mathbf{R}^T(u_i) \cdot \mathbf{R}(u_{i+1}), \tag{3.66}$$

where $\mathbf{R}_a(u_i, u_{i+1})$ is the relative rotation matrix between position at point u_i and u_{i+1}. Rotation angle is then given by formula (3.37), applied to $\mathbf{R}_a(u_i, u_{i+1})$.

3.5.1 Example for the verification of the interpolation

We make a simple sample for verifying the interpolation method in 3D space. It consists of a base and a rod that are connected by the revolute joint in space. It means it is a mechanism with one degree of freedom − angle ϕ as shown in figure 3.2. The angle ϕ was chosen as the given parameter u ($\dot{u} = 1$). Its value incremented from 0 rad to $\pi/2$ and was calculated with the step of 0.01 rad. The results of this simulation were compared with the interpolated values. The interpolated values were based on the values which were calculated with the step of 0.3 rad, it means that every 0.3 rad the errors should equal to zero.

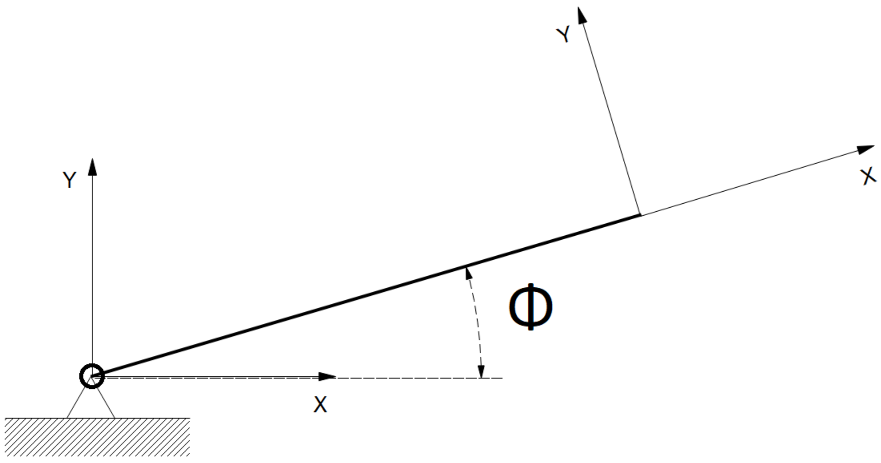


Fig. 3.2: A system for verification

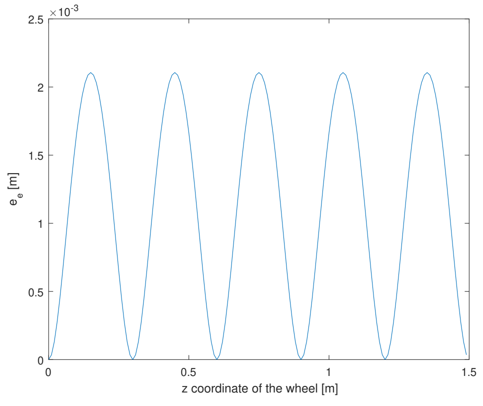


Fig. 3.3: Error in position

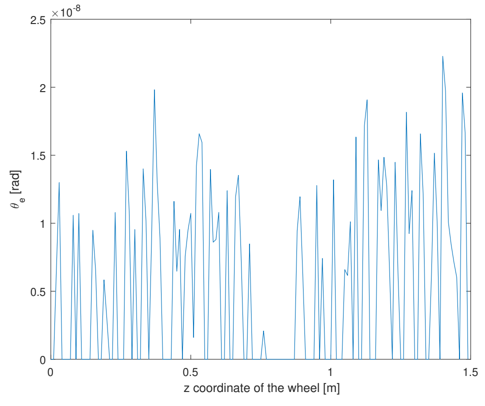


Fig. 3.4: Error in rotation

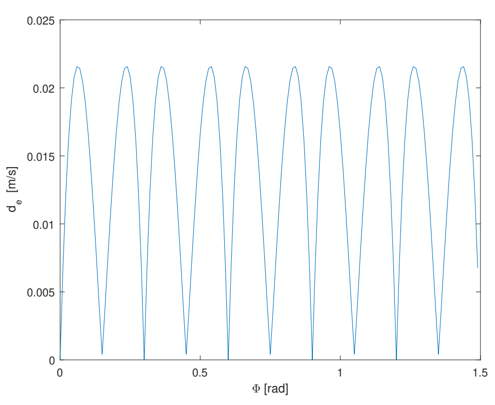


Fig. 3.5: Error of $\mathbf{d}_{A,u}$

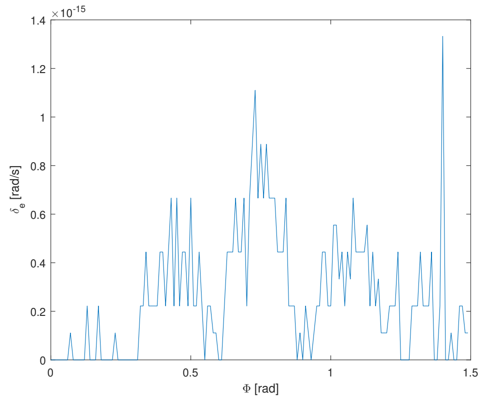
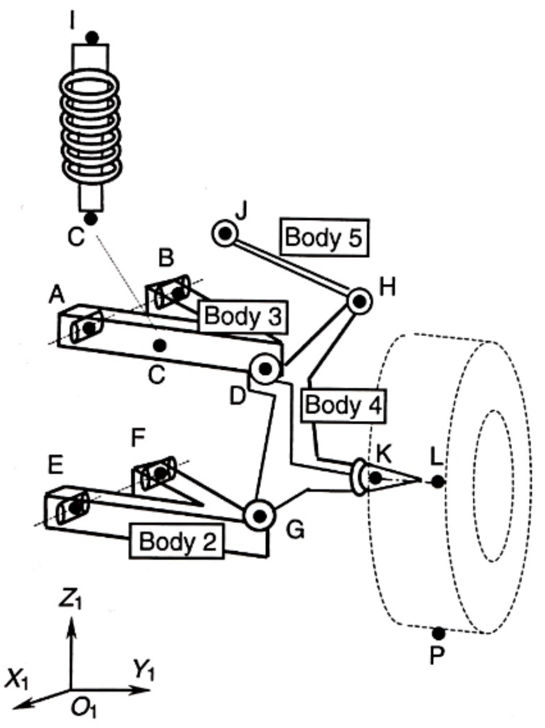


Fig. 3.6: Error of $\boldsymbol{\delta}_{A,u}$

3.5.2 Double wishbone suspension

The z-coordinate of the wheel was chosen as the given parameter u ($\dot{u} = 1$). Its value incremented from -190 mm (less is not possible due to the geometry of suspension) to 210 mm and was calculated with the step of 1 mm. The results of this simulation were compared with the interpolated values. The interpolated values were based on the values which were calculated with the step of 10 mm, it means that every 10 mm the errors should equal to zero. The norm between two successive steps is shown in figures 3.28 and 3.13.



POINT	X (mm)	Y (mm)	Z (mm)
A	103	350	142
B	−127	350	128
C	−12	491	104
D	−12	589	127
E	122	345	−80
F	−108	345	−80
G	7	620	−89
H	−156	545	178
I	−15	500	540
J	−156	317	186
P	0	678	−265
K	0	600	0
L	0	678	0

Fig. 3.7: Double wishbone suspension example geometry data [17]

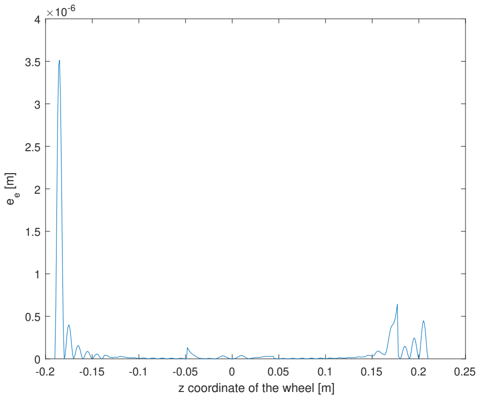


Fig. 3.8: Error in position

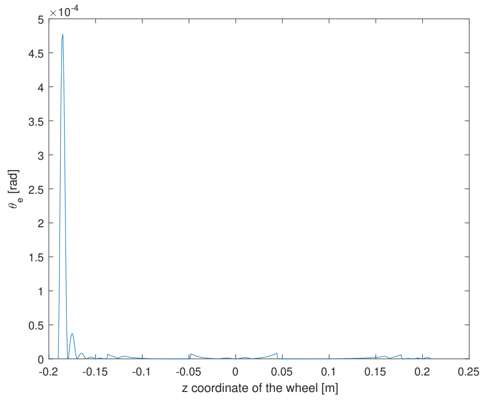


Fig. 3.9: Error in rotation

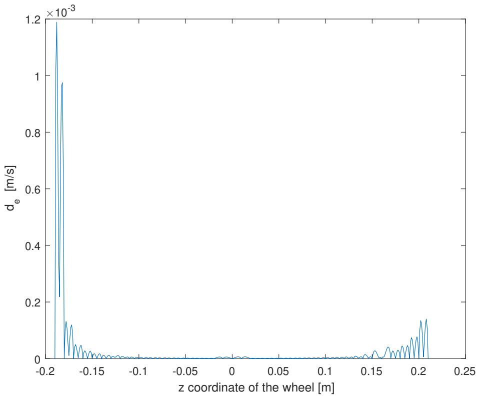


Fig. 3.10: Error of $\mathbf{d}_{A,u}$

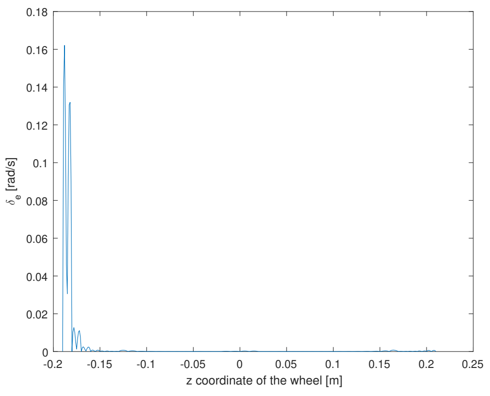


Fig. 3.11: Error of $\boldsymbol{\delta}_{A,u}$

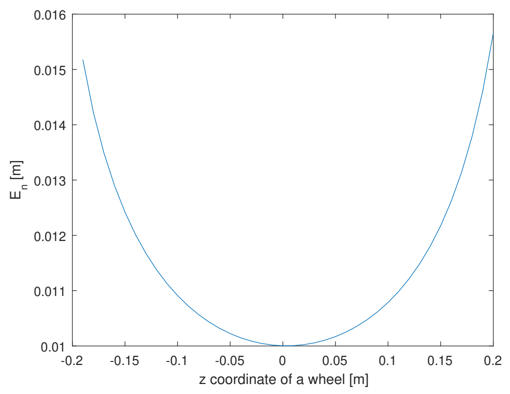


Fig. 3.12: Norm of the displacement between $\mathbf{e}(u_i)$ and $\mathbf{e}(u_{i+1})$

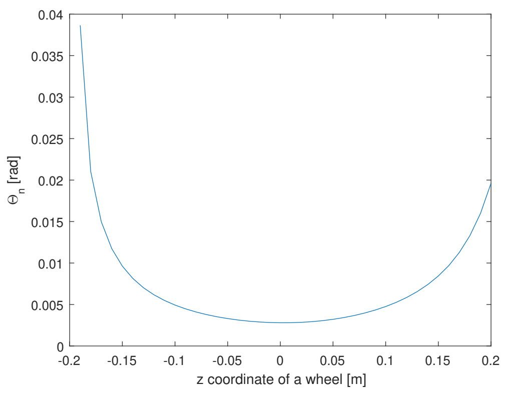
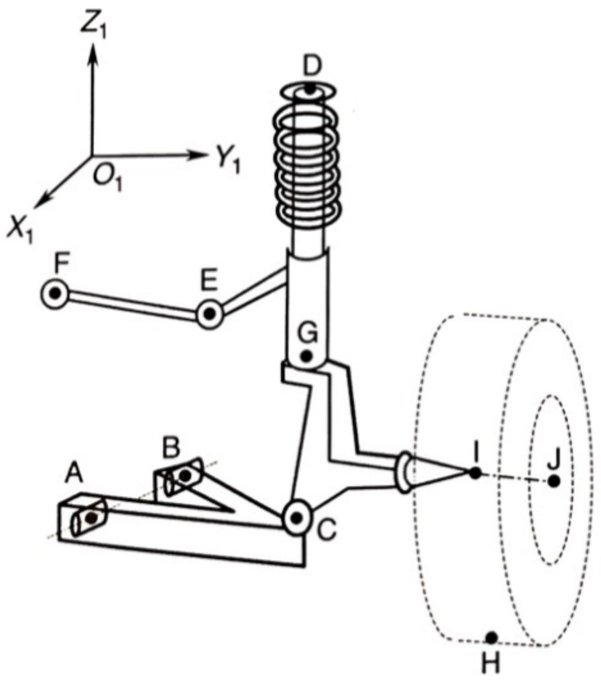
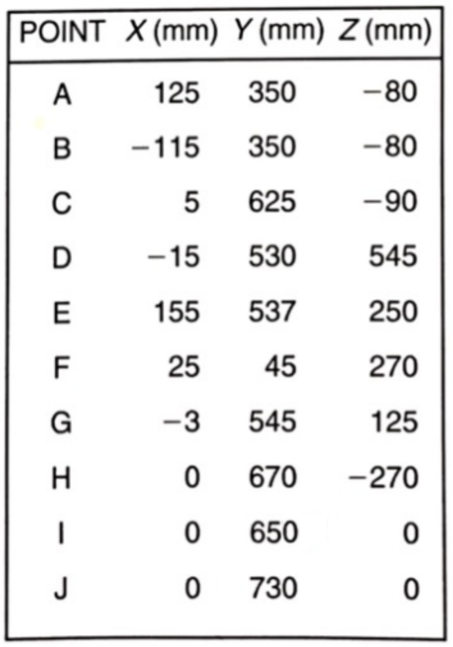


Fig. 3.13: Angle between $\mathbf{R}(u_i)$ and $\mathbf{R}(u_{i+1})$

3.5.3 McPherson suspension

The z-coordinate of the wheel was chosen as the given parameter u ($\dot{u} = 1$). Its value incremented from -200 mm to 200 mm and was calculated with the step of 1 mm. The results of this simulation were compared with the interpolated values. The interpolated values were based on the values which were calculated with the step of 10 mm, it means that every 10 mm the errors should equal to zero. The norm between two successive steps is shown in figures 3.19 and 3.20.





POINT	X (mm)	Y (mm)	Z (mm)
A	125	350	−80
B	−115	350	−80
C	5	625	−90
D	−15	530	545
E	155	537	250
F	25	45	270
G	−3	545	125
H	0	670	−270
I	0	650	0
J	0	730	0

Fig. 3.14: McPherson suspension example geometry data [17]

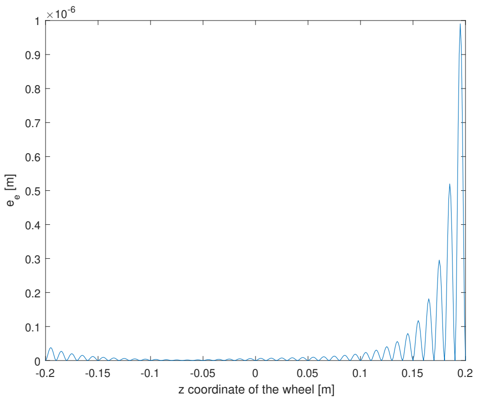


Fig. 3.15: Error in position

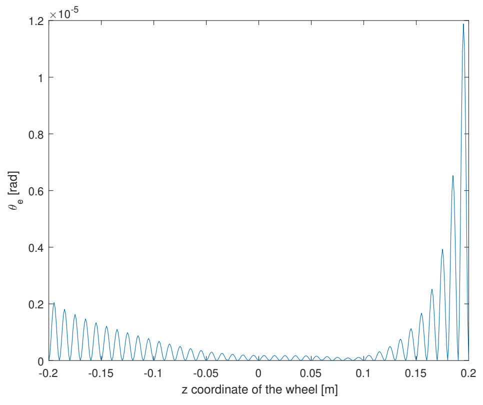


Fig. 3.16: Error in rotation

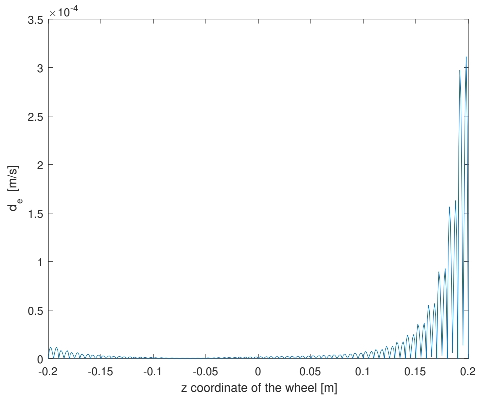


Fig. 3.17: Error of $\mathbf{d}_{A,u}$

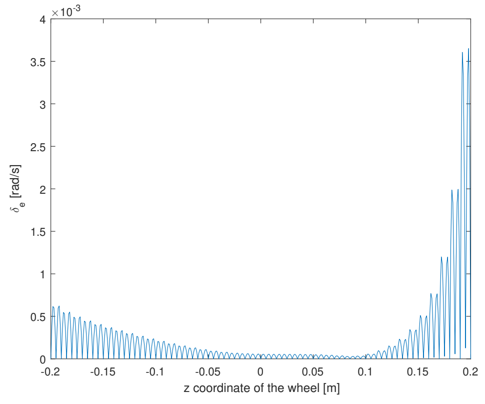


Fig. 3.18: Error of $\boldsymbol{\delta}_{A,u}$

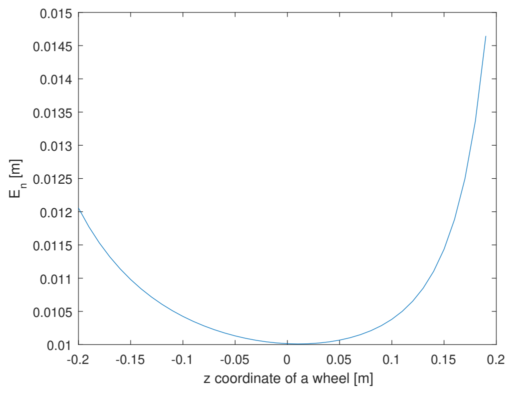


Fig. 3.19: Norm of displacement between two successive steps

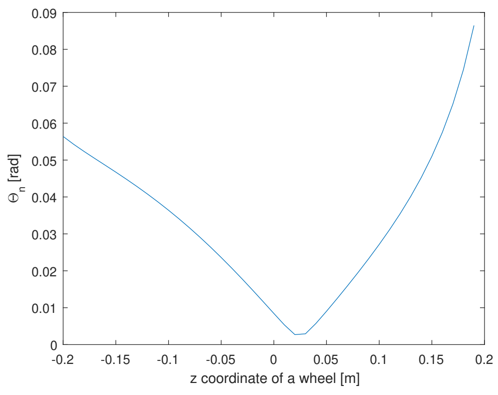


Fig. 3.20: Norm of rotation between two successive steps

3.5.4 5 Link suspension

The z-coordinate of the wheel was chosen as the given parameter u ($\dot{u} = 1$). Its value incremented from -200 mm to 200 mm and was calculated with the step of 1 mm. The results of this simulation were compared with the interpolated values. The interpolated values were based on the values which were calculated with the step of 10 mm, it means that every 10 mm the errors should equal to zero. The norm between two successive steps is shown in figures 3.26 and 3.27.

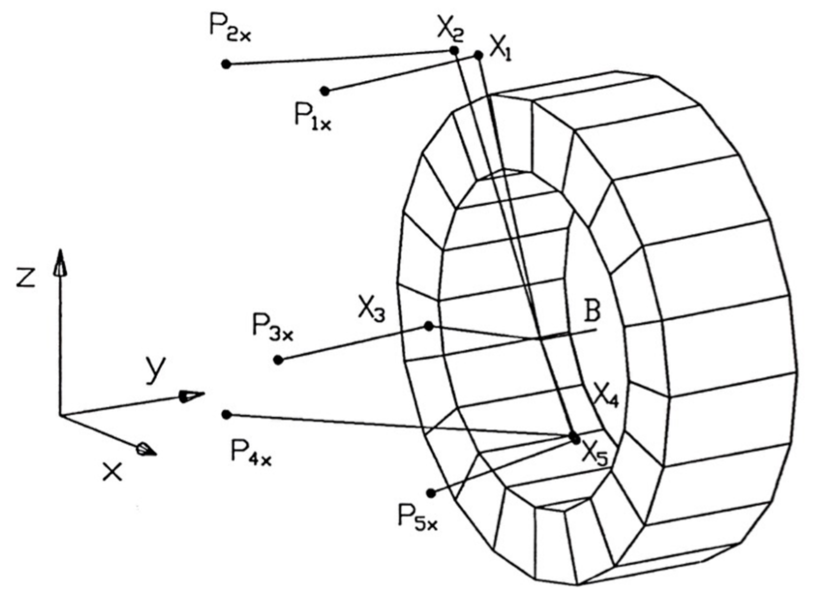


Fig. 3.21: McPherson suspension example geometry data [21]

Description		x	y	z
end of rod 1	P_{1x}	-64.0	413.0	327.0
end of rod 2	P_{2x}	-303.0	432.0	295.0
end of rod 3	P_{3x}	-93.0	366.0	4.0
end of rod 4	P_{4x}	-236.0	388.0	-109.0
end of rod 5	P_{5x}	211.5	384.5	-100.0
end of the spring	F_E	90.0	525.0	170.0
end of the damper	D_E	90	525.0	170.0

Table 3.1: Points fixed to the chassis

Description		x	y	z
end of rod 1	X_1	-64.0	636.0	345.0
end of rod 2	X_2	-117.0	636.0	338.0
end of rod 3	X_3	-188.0	647.0	-23.0
end of rod 4	X_4	-5.0	737.0	-130.0
end of rod 5	X_5	2.5	737.0	-134.0
end of the spring	M_1	-103.0	463.0	87.0
end of the damper	B	0	786.0	0
	B_S	0	686.0	-1.0

Table 3.2: Points fixed to the chassis

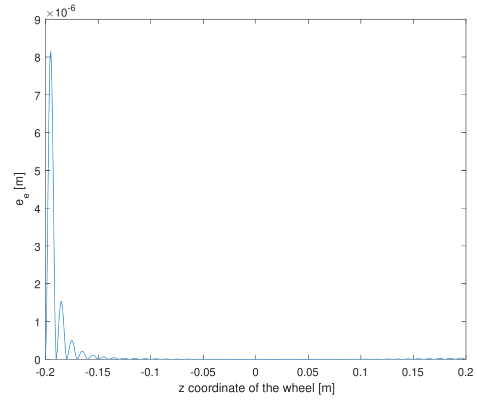


Fig. 3.22: Error in position

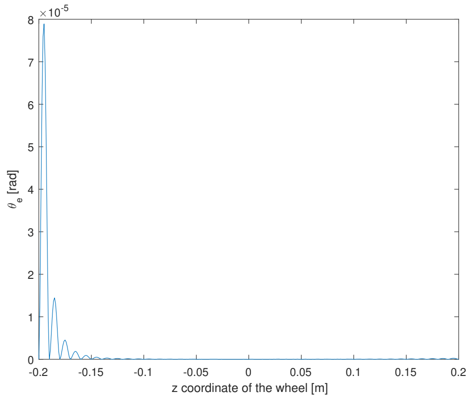


Fig. 3.23: Error in rotation

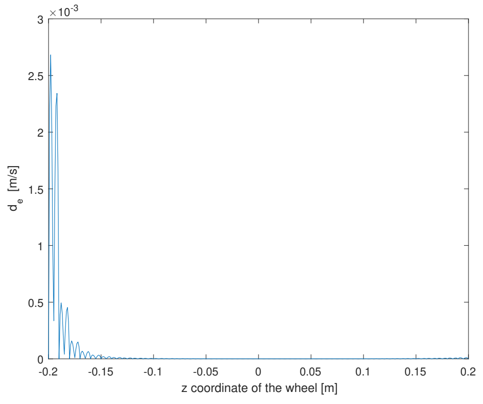


Fig. 3.24: Error of $\mathbf{d}_{A,u}$

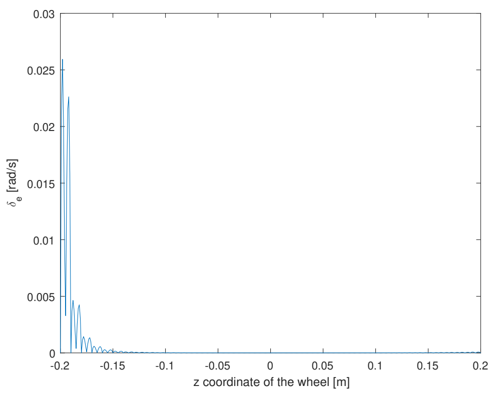


Fig. 3.25: Error of $\boldsymbol{\delta}_{A,u}$

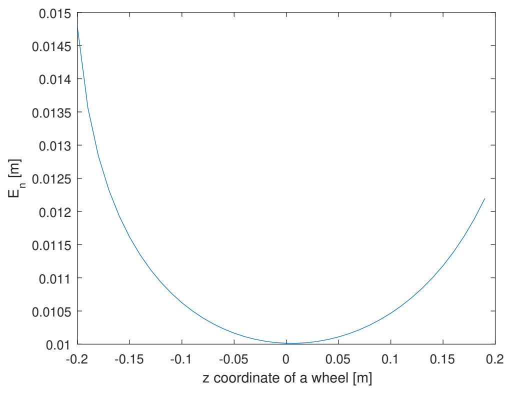


Fig. 3.26: Norm of displacement between two successive steps

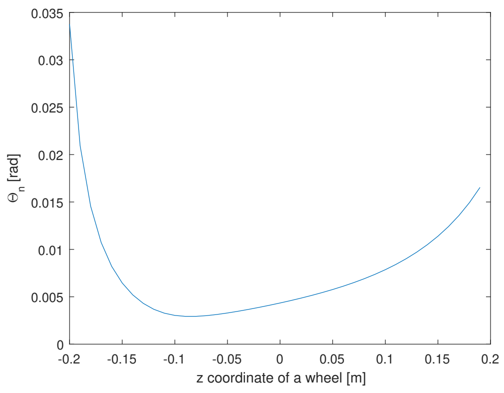


Fig. 3.27: Norm of rotation between two successive steps

3.6 Conclusion and recommendations

The presented results show that the accuracy depends strongly on the magnitude of E_n − the norm in translation between two steps. Next, we discuss the behaviour of e_e − the error in translation on the step size (figure 3.28). After some research on the suspensions we find out that the maximum of e_e depends on the step value with exponential growth (figure 3.29). In conclusion, it is essential to monitor not only step size but also the norm in translation between two successive steps − E_n, because bigger step size does not necessarily lead to bigger E_n and vice versa. Besides, the biggest error is usually in the middle of the interpolated interval, which can be seen in figures 3.3 and 3.5. Errors plotted in figures 3.4 and 3.6 are much lower because the given parameter is angle and the motion is circular, which is easy to interpolate from the view of angles. The reader may notice, the errors in suspension position and $\mathbf{d}$ vectors are lower than the errors in rotation angles and $\boldsymbol{\delta}$ vectors. It is probably because the motion of the suspension is more sliding motion than rotary motion.

We have found that the dependence between the step size and the interpolation error is exponential. Therefore, we recommend performing an error analysis before the proposed interpolation method is implemented because the accuracy of the method can be poor if the step size is large.

The times needed to compute the exact solution and the interpolated one from the precomputed table are approx. 1000 s and approx. 3 s. This difference is clear proof of the main advantage of the proposed interpolation method.

In the next figures we can see the dependence of value e_e on the step size and the z coordinate of the wheel support − figure 3.28 and the maximum value of e_e on the step value − 3.29.

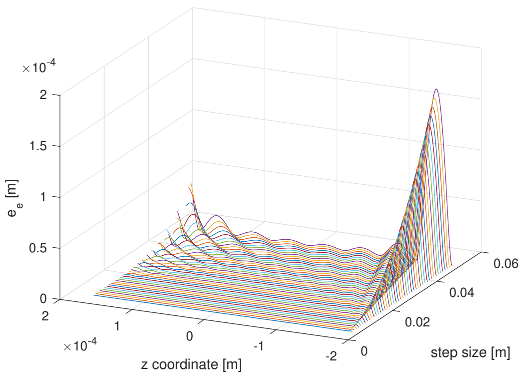


Fig. 3.28: Translation error on interpolation step size - double wishbone

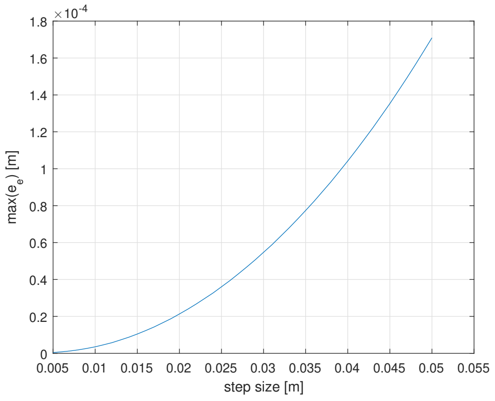


Fig. 3.29: Translation error on step size - double wishbone

Chapter 4

Implementation of kinematic interpolation under EasyDyn

The purpose of this chapter is to reproduce in `C++` the interpolation performed under Matlab in the previous chapter at position, velocity and acceleration levels. The next goal of this chapter is to compare all results computed in Matlab with results obtained by C++.
EasyDyn [24] is a `C++` library for the simulation of multibody systems and problems represented by the first-order or second-order differential equations. The library is organized in 4 modules [24]

- the *vec* module, introducing classes related to vector calculus: vectors, rotation tensors, inertia tensors and homogeneous transformation matrices;
- the *sim* module to integrate second-order differential equations;
- the *mbs* module, a fronted to *sim* which automatically builds the differential equations of motion of a multibody systems from the kinematics and the applied forces;
- the *visu* module, allowing to build object-oriented scenes composed of moving objects.

4.1 Approach

First, we rewrote Matlab codes for the interpolation at position, velocity and acceleration levels in `C++` (example in appendix A.3. So that the `C++` code does not have to include a kinematics solver, the computed data from Matlab were stored into the table in terms of series of values $u_0,\ u_1,\ u_2,\ ...,u_N$.

$$\begin{array}{llll}
N & & & \\
u_0 & \mathbf{T}_{0,A}(u_0) & \{\mathbf{d}_{A,u}(u_0)\}_0 & \{\boldsymbol{\delta}_{A,u}(u_0)\}_0 \\
u_1 & \mathbf{T}_{0,A}(u_1) & \{\mathbf{d}_{A,u}(u_1)\}_0 & \{\boldsymbol{\delta}_{A,u}(u_1)\}_0 \\
u_2 & \mathbf{T}_{0,A}(u_2) & \{\mathbf{d}_{A,u}(u_2)\}_0 & \{\boldsymbol{\delta}_{A,u}(u_2)\}_0 \\
\vdots & \vdots & \vdots & \vdots \\
u_N & \mathbf{T}_{0,A}(u_N) & \{\mathbf{d}_{A,u}(u_N)\}_0 & \{\boldsymbol{\delta}_{A,u}(u_N)\}_0,
\end{array}$$

where N is a number of rows and in which each matrix $\mathbf{T}_{0,A}(u_i)$ is divided into three parts

$$\mathbf{T}_{0,A}(u_i)(1,1:4) \quad \mathbf{T}_{0,A}(u_i)(2,1:4) \quad \mathbf{T}_{0,A}(u_i)(3,1:4)$$

to make writing and reading of the data easier. The values from `C++` were exported with the precision of 23 decimal places.

4.2 Results

In this section we compared interpolated data from Matlab file and from `C++` file for McPherson, 5-link and double wishbone (DW) suspensions. It means the vectors given by Matlab:

$\mathbf{e}_M(u_i)$, $\mathbf{d}_M(u_i)$, $\boldsymbol{\delta}_M(u_i)$, $\{d\mathbf{d}/du\}_M(u_i)$, $\{d\boldsymbol{\delta}/du\}_M(u_i)$

were subtracted from the vectors given by `C++`:

$\mathbf{e}_C(u_i)$, $\mathbf{d}_C(u_i)$, $\boldsymbol{\delta}_C(u_i)$, $\{d\mathbf{d}/du\}_C(u_i)$, $\{d\boldsymbol{\delta}/du\}_C(u_i)$

and the result (error) was normalized for each value of u. The quantities depicted in figures 4.1 − 4.15 are defined as:

$$\mathbf{e}_d(u_i) = \|\mathbf{e}_C(u_i) - \mathbf{e}_M(u_i)\|, \tag{4.1}$$
$$\mathbf{d}_d(u_i) = \|\mathbf{d}_C(u_i) - \mathbf{d}_M(u_i)\|, \tag{4.2}$$
$$\boldsymbol{\delta}_d(u_i) = \|\boldsymbol{\delta}_C(u_i) - \boldsymbol{\delta}_M(u_i)\|, \tag{4.3}$$
$$\{d\mathbf{d}/du\}_d(u_i) = \|\{d\mathbf{d}/du\}_C(u_i) - \{d\mathbf{d}/du\}_M(u_i)\|, \tag{4.4}$$
$$\{d\boldsymbol{\delta}/du\}_d(u_i) = \|\{d\boldsymbol{\delta}/du\}_C(u_i) - \{d\boldsymbol{\delta}/du\}_M(u_i)\|. \tag{4.5}$$

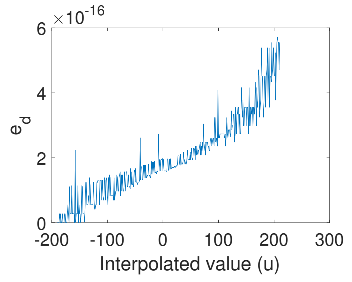


Fig. 4.1: Error of $\mathbf{e}(u_i)$ of DW suspension - [m]

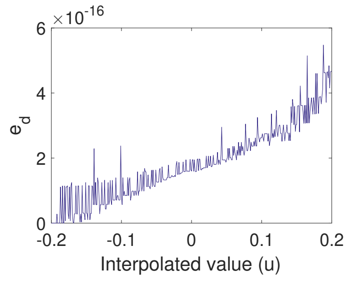


Fig. 4.2: Error of $\mathbf{e}(u_i)$ of McPherson suspension - [m]

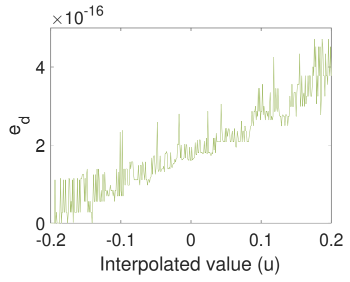


Fig. 4.3: Difference of vectors $\mathbf{e}(u_i)$ of 5 link suspension - [m]

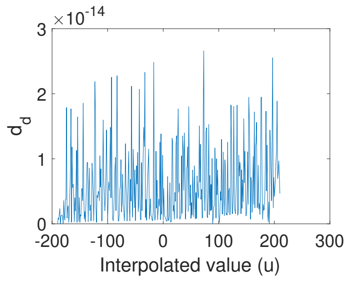


Fig. 4.4: Difference of vectors $\mathbf{d}(u_i)$ of DW suspension - [m]

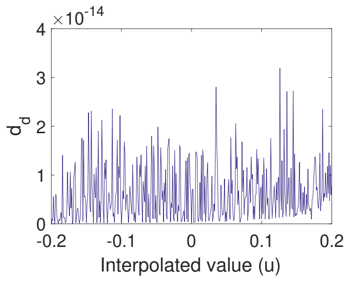


Fig. 4.5: Difference of vectors $\mathbf{d}(u_i)$ of McPherson suspension - [m]

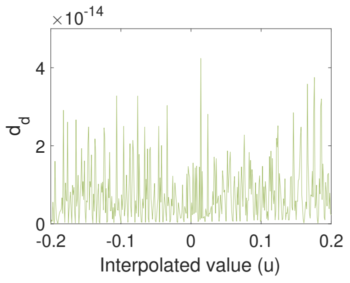


Fig. 4.6: Difference of vectors $\mathbf{d}(u_i)$ of 5 link suspension - [m]

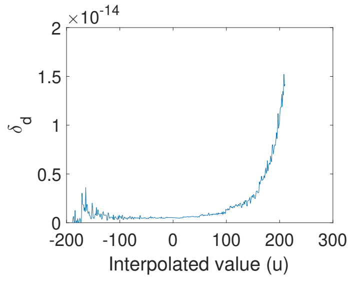


Fig. 4.7: Error of $\boldsymbol{\delta}(u_i)$ of DW suspension - [m]

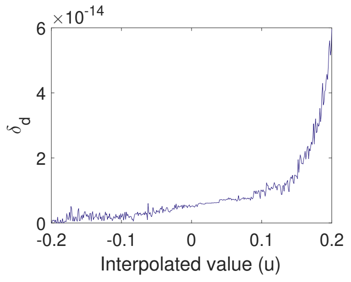


Fig. 4.8: Error of $\boldsymbol{\delta}(u_i)$ of McPherson suspension - [m]

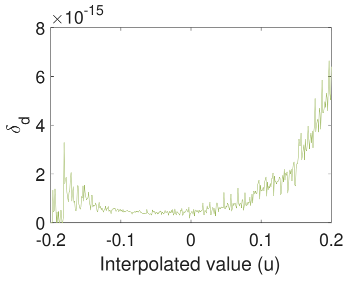


Fig. 4.9: Error of $\boldsymbol{\delta}(u_i)$ of 5 link suspension - [m]

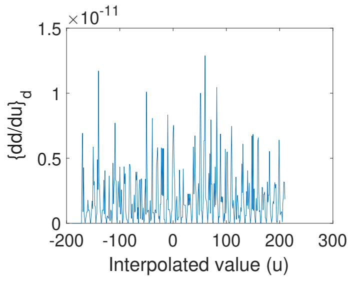


Fig. 4.10: Error of $\{d\mathbf{d}/du\}(u_i)$ of DW suspension - [m]

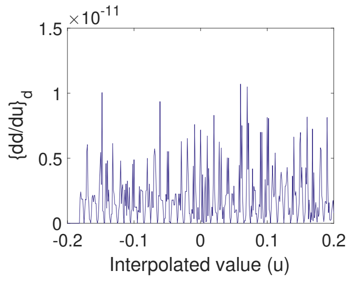


Fig. 4.11: Error of $\{d\mathbf{d}/du\}(u_i)$ of McPherson suspension - [m]

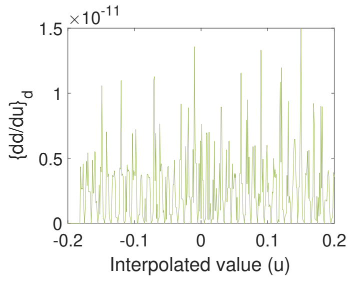


Fig. 4.12: Error of $\{d\mathbf{d}/du\}(u_i)$ of 5 link suspension - [m]

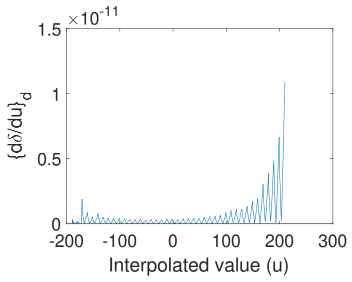


Fig. 4.13: Error of $\{d\boldsymbol{\delta}/du\}(u_i)$ of DW suspension - [m]

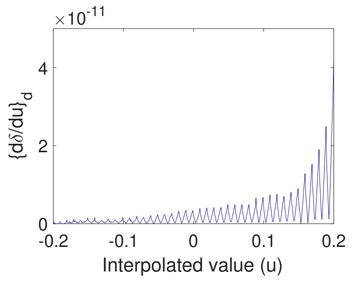


Fig. 4.14: Error of $\{d\boldsymbol{\delta}/du\}(u_i)$ of McPherson suspension - [m]

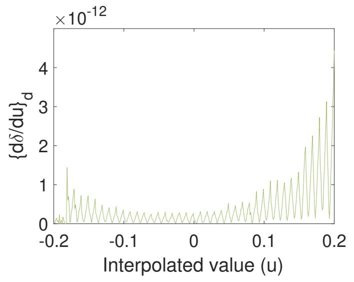


Fig. 4.15: Error of $\{d\boldsymbol{\delta}/du\}(u_i)$ of 5 link suspension - [m]

4.3 Conclusion

The largest difference over all compared quantities given by `C++` and Matlab was approximately 10^{-10} which is negligible value despite the fact that input values were exported with the precision of 10^{-23}.
The errors should reach 0 value at each value u_i, because it should be the exact value, but it does not. On the other hand errors are low in comparison with real values of the quantities. I also tried to offset compared values but results of errors were higher, so the right values of u_i are compared.

We can conclude that the errors are negligible and are caused by different computer number formats in Matlab and C++, so the computation in meters is precise enough and can be used for the next work.

To prove that the errors are caused by different computer number formats in Matlab and C++, we proceed the computation in millimetres. We obtain following results (figure 4.16 − figure 4.30), which are more accurate and even reach zero at each value u_i.

To sum up we proved that the errors obtained from the computation in meters are numerical errors and are still low enough for the next usage.

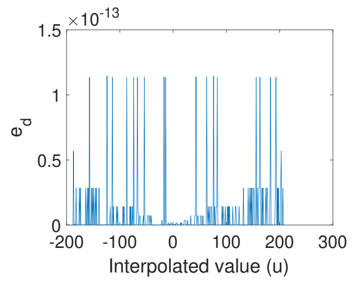


Fig. 4.16: Error of $\mathbf{e}(u_i)$ of DW suspension - [mm]

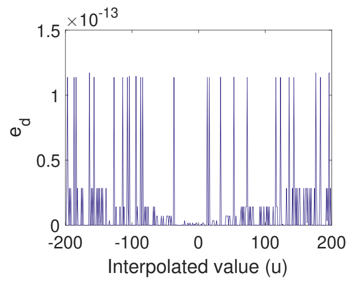


Fig. 4.17: Error of $\mathbf{e}(u_i)$ of McPherson suspension - [mm]

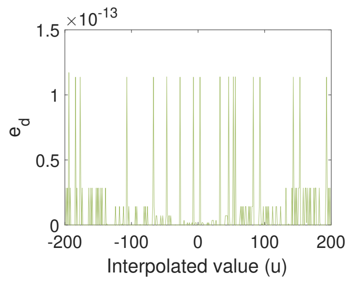


Fig. 4.18: Error of $\mathbf{e}(u_i)$ of 5 link suspension - [mm]

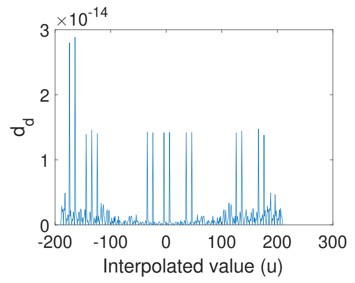


Fig. 4.19: Difference of vectors $\mathbf{d}(u_i)$ of DW suspension - [mm]

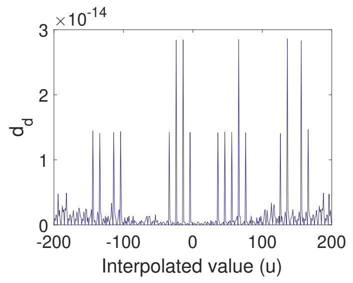


Fig. 4.20: Difference of vectors $\mathbf{d}(u_i)$ of McPherson suspension - [mm]

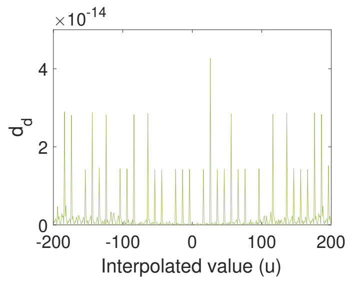


Fig. 4.21: Difference of vectors $\mathbf{d}(u_i)$ of 5 link suspension - [mm]

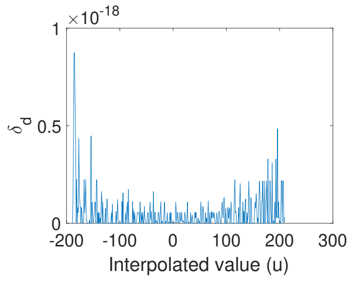


Fig. 4.22: Difference of vectors $\boldsymbol{\delta}(u_i)$ of DW suspension - [mm]

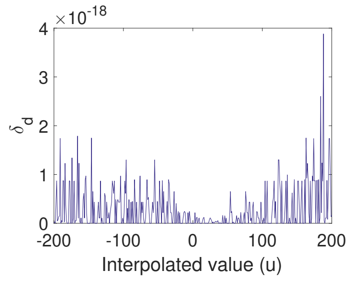


Fig. 4.23: Difference of vectors $\boldsymbol{\delta}(u_i)$ of McPherson suspension - [mm]

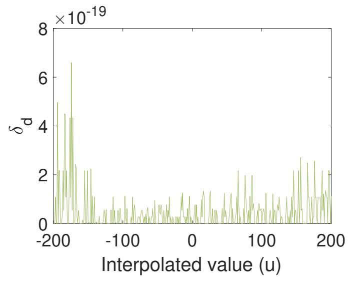


Fig. 4.24: Error of $\boldsymbol{\delta}(u_i)$ of 5 link suspension - [mm]

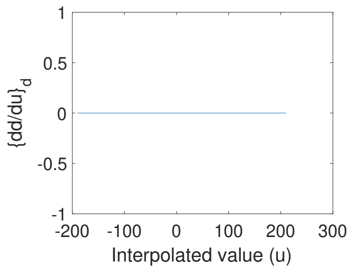


Fig. 4.25: Error of $\{d\mathbf{d}/du\}(u_i)$ of DW suspension - [mm]

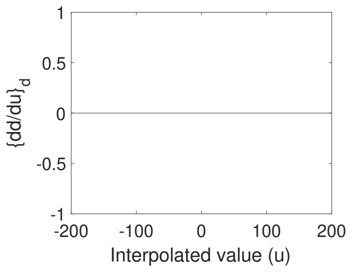


Fig. 4.26: Error of $\{d\mathbf{d}/du\}(u_i)$ of McPherson suspension - [mm]

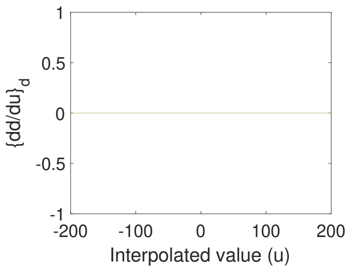


Fig. 4.27: Error of $\{d\mathbf{d}/du\}(u_i)$ of 5 link suspension - [mm]

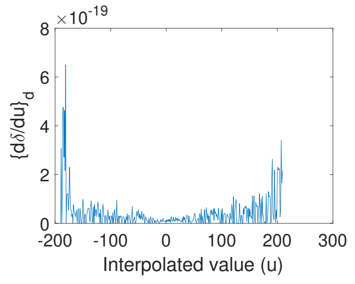


Fig. 4.28: Error of $\{d\boldsymbol{\delta}/du\}(u_i)$ of DW suspension - [mm]

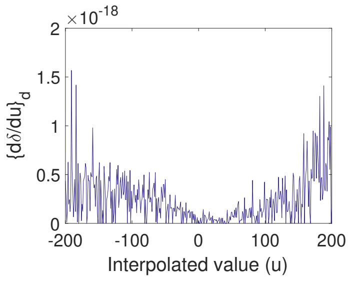


Fig. 4.29: Error of $\{d\boldsymbol{\delta}/du\}(u_i)$ of McPherson suspension - [mm]

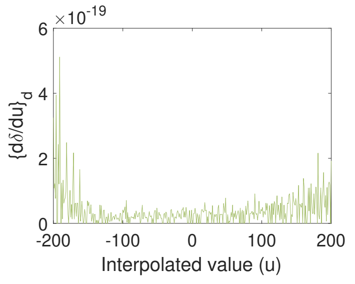


Fig. 4.30: Error of $\{d\boldsymbol{\delta}/du\}(u_i)$ of 5 link suspension - [mm]

Chapter 5

EasyDyn model verification and application on real example

The proposed interpolation method is verified in this chapter. The verification is proceeded on the level of dynamics. For the verification, three approaches to model suspension from the formula car of the UWB Racing are chosen:

- the proposed approach implemented in EasyDyn,
- a model in Ansys Mechanical 2020 r1 [19], and
- a model implemented in Matlab in accordance with Shabana [18].

First, a model of the suspension used in a formula car of the UWB Racing Team Pilsen is developed in EasyDyn. Then we compare the results of the proposed approach executed in EasyDyn with the results obtained in commercial software Ansys Mechanical 2020 r1 [19]. We further implemented a model based on the approach proposed by Shabana [18] and used the results for the verification. We verify the kinematics and dynamic by using two states:

- The model where the external force is only gravity.

  This model verifies accuracy of kinematics and dynamic.

- The model where a spring and damper are included.

  EasyDyn allows using springs and dampers as well as Ansys Mechanical. By this simulation we can compare the equilibrium states.

All components of all models are massless to achieve similarity, only one mass point is placed into the centre of wheel support (point K - figure 5.2), where is also placed interpolated coordinate system in EasyDyn.

Point	x [m]	y [m]	z [m]
A	0.9650	-6.8969e-02	0.46817
B	0.9650	-0.23987	0.46236
C	0.9650	-0.2350	0.3800
D	0.9650	-0.26802	0.4301
E	0.77129	-0.25058	0.15827
F	1.0589	-0.24167	0.18498
G	0.76463	-0.25083	6.0576e-02

Point	x [m]	y [m]	z [m]
H	1.0404	-0.22648	5.2443e-02
I	1.0923	-0.24551	7.00e-02
J	0.9850	-0.5500	8.25e-02
K	0.98503	-0.56317	0.16499
L	1.0330	-0.56672	0.10572
M	0.96643	-0.52541	0.2575
N	0.9659	-0.48461	0.26819

Fig. 5.1: Coordinates of points where joints or coordinate systems are placed

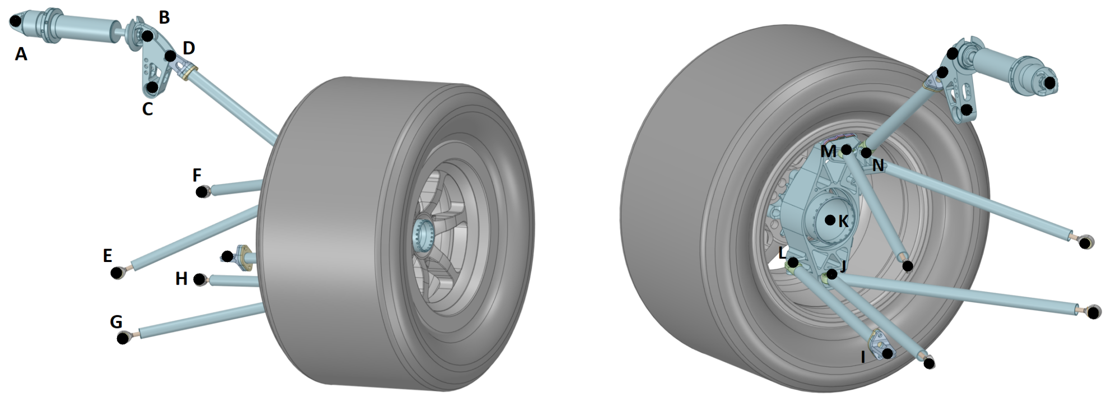


Fig. 5.2: Detail of the modelled suspension

5.1 Comparison of motion in gravity field

In this section, we compare the simulation results of the proposed model in EasyDyn with results from Ansys Mechanical and from the method presented in [18]. We let the wheel support fall in a gravity field with gravitational acceleration $\mathbf{g} = (0, 0, -9.81)\ \mathrm{m} \cdot \mathrm{s}^2$.

We compare position components for all three models and the angular velocity components only between Ansys Mechanical model and EasyDyn model. We do not

compare orientation of the wheel support because Ansys Mechanical does not support that.
EasyDyn model consists of two bodies. The interpolated one which represents the wheel support and which weights 10 kg, and chassis. The initial condition of z coordinate of wheel support is set to $z = 0.1$ m. Velocity initial conditions are set to zero.

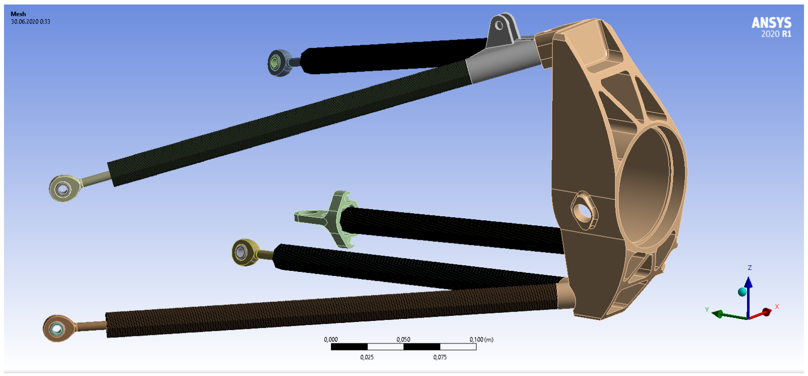


Fig. 5.3: Model visualisation in Ansys Mechanical

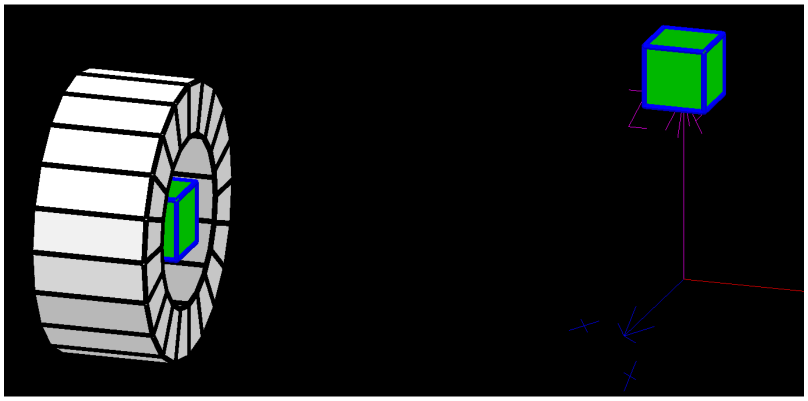


Fig. 5.4: Model visualisation with tyre in EasyDyn

Ansys Mechanical model consists of the upper and lower arm, steering rod and the wheel support. The initial condition is set the same as to EasyDyn model. The mass point is placed into the centre of wheel support, and its mass is 10 kg.
Model implemented in accordance with [18] has three bodies: upper and lower arm and wheel support. Initial conditions and weight properties are set the same as for

the previous examples. The following figures show models implemented in Ansys Mechanical and EasyDyn. The wheel support centre of Ansys Mechanical model has the same coordinates as the wheel centre of EasyDyn model.

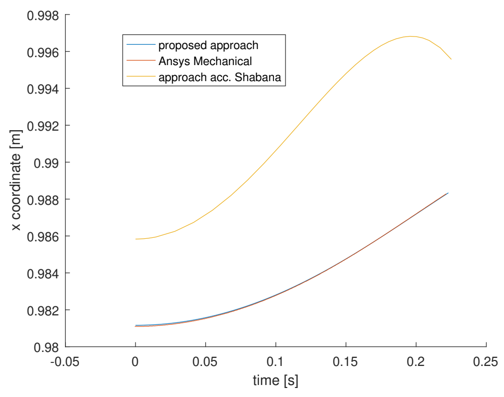


Fig. 5.5: Comparison of x component of position vector

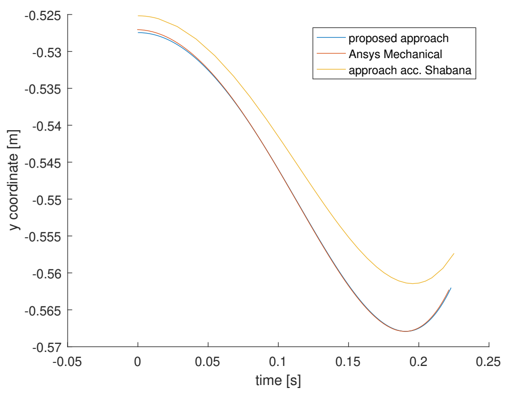


Fig. 5.6: Comparison of y component of position vector

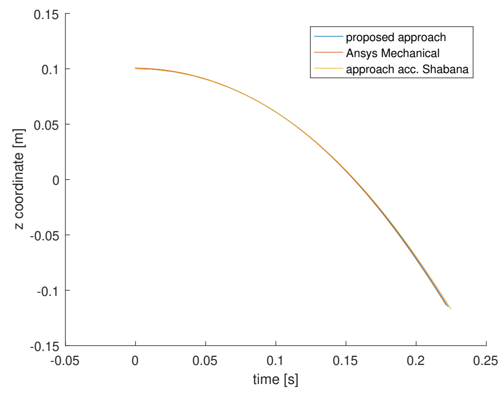


Fig. 5.7: Comparison of z component of position vector

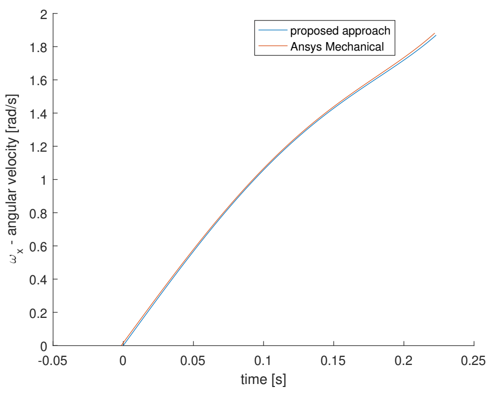


Fig. 5.8: Comparison of x component of angular velocity vector

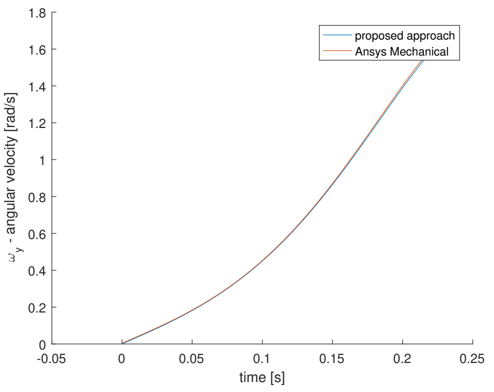


Fig. 5.9: Comparison of y component of angular velocity vector

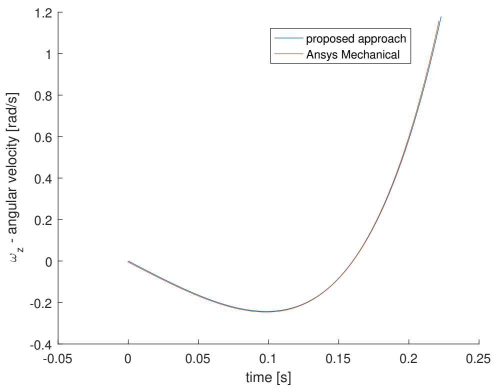


Fig. 5.10: Comparison of z component of angular velocity vector

5.2 Comparison of motion with external forces

In this section, we compare the behaviour of the Formula student suspension where forces from spring and damper are included. Spring and damper locate between points A and B, see figure 5.2. EasyDyn model consists of three bodies. The interpolated one which represents the wheel support and which weights 10 kg, chassis and crank. The crank body is defined by points B,C and D (figure 5.2). The initial condition of z coordinate of wheel support is set to $z = 0.0\ m$. Velocity initial conditions are set to zero.

Ansys Mechanical model consists of the upper and lower arm, steering rod, the wheel support, crank, rod between crank and wheel support, damper and spring. The initial condition is set the same as to EasyDyn model. The mass point is placed into the centre of wheel support, its mass is 10 kg. It is important to define the right joint at point N (figure 5.2), because in reality there is the spherical joint between the upper arm and the rod. But in EasyDyn model there are only three bodies: wheel support, chassis and crank. It means that spherical joint is defined at point N, but it is defined between the rod and wheel support.

Additional parameters:

- spring stiffness = 3500 N/m,
- damping = 50 N·s/m,
- weight of mass point/ weight of wheel support in EasyDyn = 10 kg,

- integration step = 0.0005 s,
- integration type EasyDyn – Newmark Integration; Ansys Mechanical – Runge-Kutta 4.

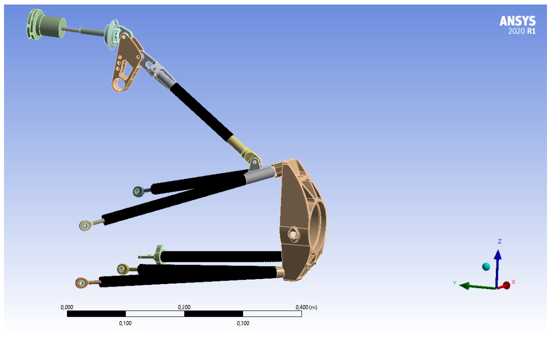


Fig. 5.11: Model visualisation in Ansys Mechanical, see here

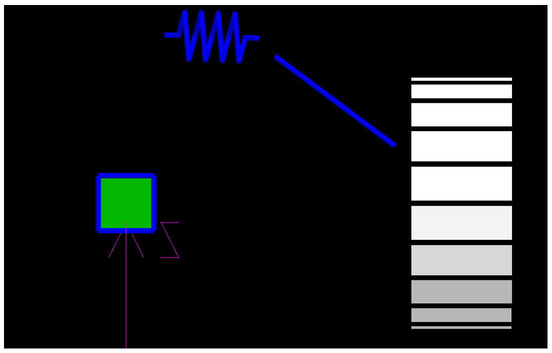


Fig. 5.12: Model visualisation with tyre in EasyDyn, see here

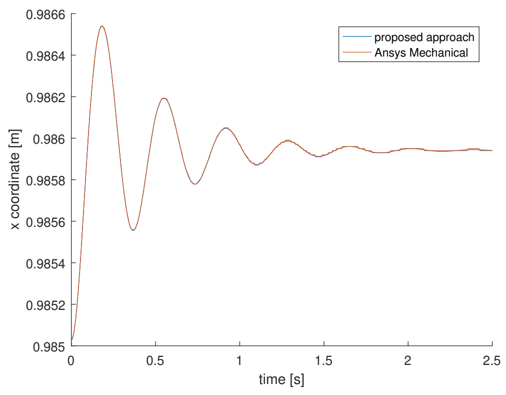


Fig. 5.13: Comparison of x component of position vector

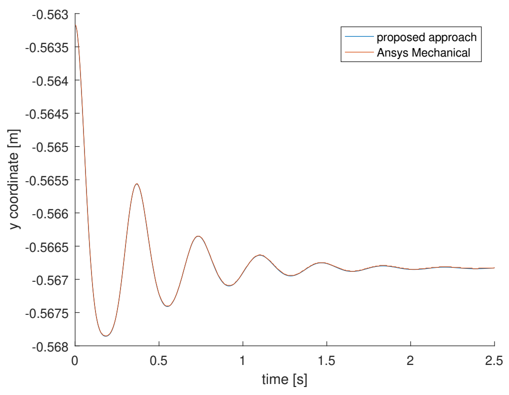


Fig. 5.14: Comparison of y component of position vector

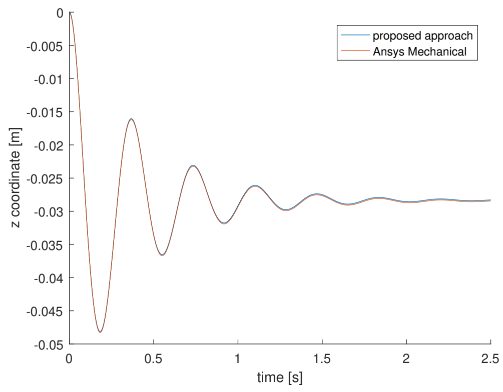


Fig. 5.15: Comparison of z component of position vector

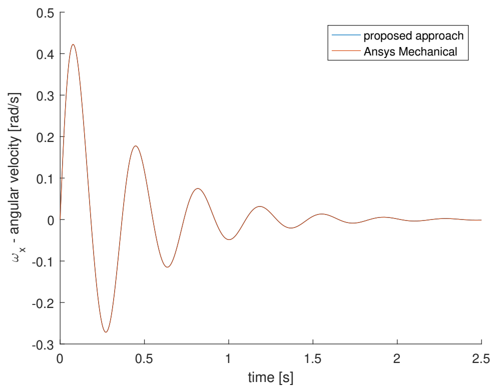


Fig. 5.16: Comparison of x component of angular velocity vector

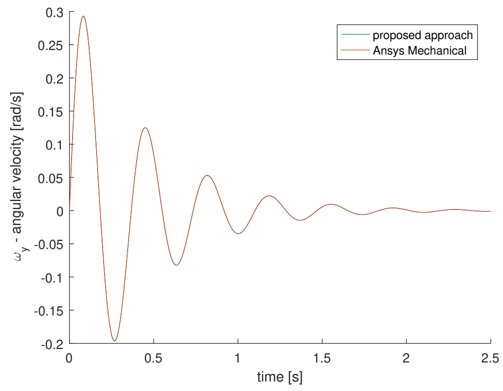


Fig. 5.17: Comparison of y component of angular velocity vector

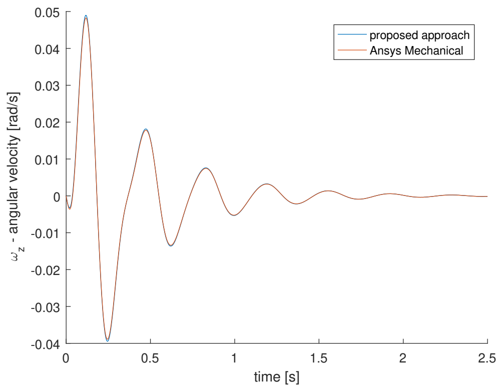


Fig. 5.18: Comparison of z component of angular velocity vector

5.3 Discussion

In the case of gravity field Ansys Mechanical and EasyDyn model are almost the same. The maximum difference is $7 \cdot 10^{-4}$ mm and 10^{-2} rad $\cdot$ s^{-1} (figures 5.19 – 5.20) which is negligible in comparison with the values of position and angular velocity. The model based on prof. Shabana approach [18] is not included in the figures, because the differences between this model and Ansys Mechanical model or EasyDyn model were much higher – around 0.7 mm.

In the case of the example with external forces, the differences between results obtained with EasyDyn and Ansys Mechanical model are even lower. This behaviour was given by uncertainty of initial conditions because it is not possible to set remote displacement at a particular time in Ansys Mechanical. The differences between position and angular velocity are in figures 5.21 − 5.22. The most significant difference in position can be seen in z coordinate. It is probably caused by the biggest motion in this direction. The z coordinate difference value tend to $1.5 \cdot 10^{-4}$ m as the system tends to its equilibrium position. The differences in angular velocities are similar to each other, and their values reach 10^{-3} rad $\cdot$ s^{-1} and tend to zero because the motion is damped.
To conclude the weakness of EasyDyn model is the inability to connect the spring or damper to a body other than the wheel support. This connection is possible in case of McPherson or 5 link suspension where the spring and the damper are usually connected directly to the wheel support. However, if we assume a spherical joint between the upper arm and the rod at point N, we get different results shown in figure 5.23. The discrepancy is caused by the different setting of the kinematics. On the other hand, EasyDyn model gives very similar results as the same model implemented in commercial software Ansys Mechanical.

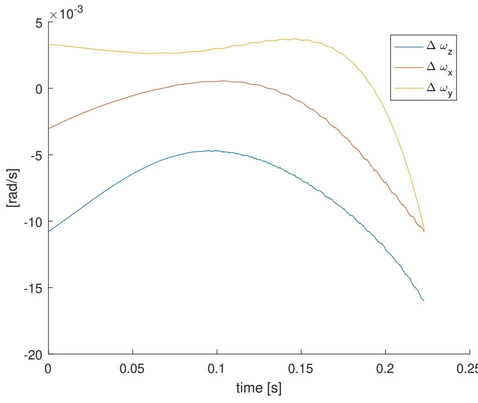


Fig. 5.19: Gravity field case - differences in components of angular velocity

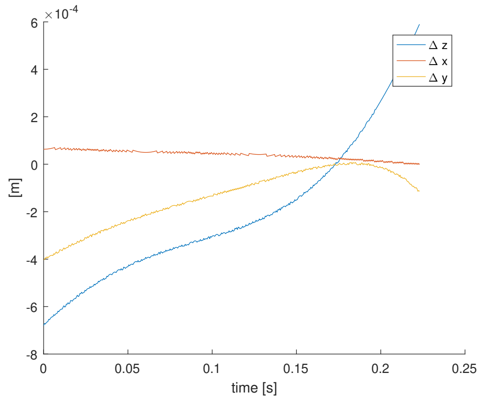


Fig. 5.20: Gravity field case - differences in components of position

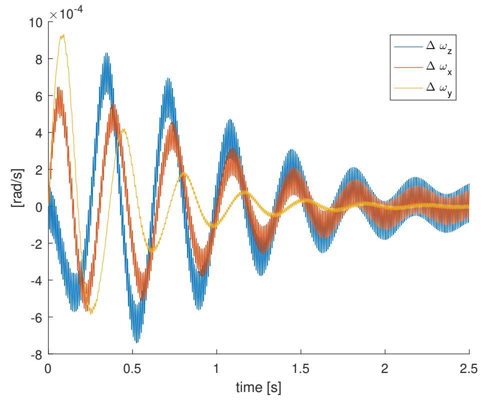


Fig. 5.21: Spring and damper case - in components of angular velocity

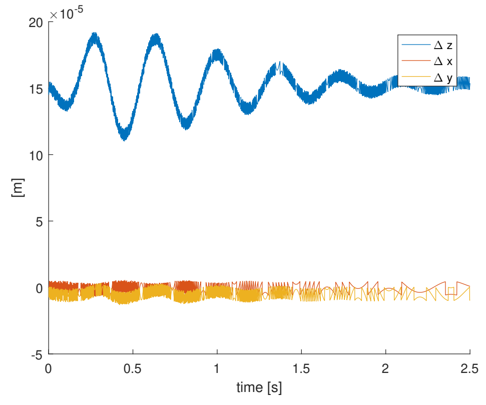


Fig. 5.22: Spring and damper case - differences in components of position

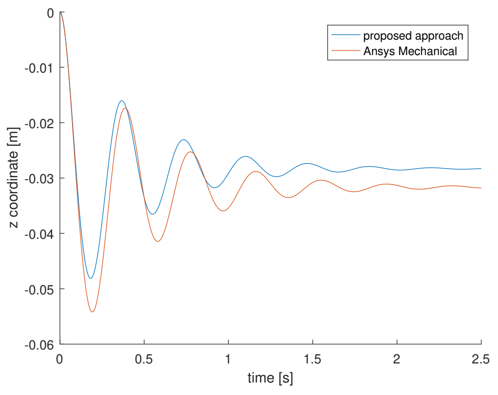


Fig. 5.23: Difference when spherical joint is at point N

Chapter 6

Conclusion

This work has provided an overview of kinematic interpolation in multibody systems with perfect joints for multibody dynamics simulation. The kinematics interpolation has a significant advantage of saving computational time, primarily when a kinematic chain consists of many parts and has a limited number of end effectors. It is because the interpolation process with searching in a lookup table is far less demanding than solving nonlinear constraint equations at each integration step during the dynamic solution. On the other side, the precision of the interpolated motion is limited, and the lookup table has to be evaluated beforehand.

The work is focused mainly on the kinematics of automotive suspension systems and the interpolation employing cubic Hermite splines. A thorough search of the relevant literature yielded only a limited number of works focused on the kinematic interpolation using the cubic Hermite splines [26] with no work dealing with the automotive suspension systems. An interpolation method, which gives the body configuration on the level of position/orientation, velocity and acceleration, has been introduced. The method is based on the cubic Hermite splines mentioned earlier and provides smoothness on the position/orientation and the velocity levels and continuity on the acceleration level. These characteristics are present because the second derivative of a cubic polynomial is a linear function.

The evaluation of the input lookup tables has been demonstrated for the most common types of suspension systems including a double wishbone, McPherson and five link suspension. The lookup tables have been evaluated in the Cartesian coordinates using a code written in Matlab. The interpolation for all three suspension systems was firstly implemented under Matlab and then successfully reproduced in C++ with using EasyDyn library. It has been shown that the data transfer between

Matlab and C++ can cause numerical errors due to different computer number formats in these two programming languages. However, the numerical errors due to the finite precision of the digital number are usually negligible in comparison with the numerical errors due to the interpolation.

At the end of the work, the proposed method has been applied to the real example of a Formula SAE student suspension. The results from EasyDyn have been compared with the results obtained using Ansys Mechanical and using a code implemented in accordance with reference [18]. Firstly the models have been analysed only in a gravity field and then with acting external forces in spring and damper. The results obtained with EasyDyn and Ansys Mechanical are almost identical in the gravity field, and the differences are low also in a case with the external forces. The maximum difference in position is less than $2 \cdot 10^{-4}$ m, and the difference in angular velocity reached 10^{-3} rad.s^{-1}, the relative errors are less than 0.4 % and 0.3 %, respectively. Therefore, the proposed method is suitable for computationally-demanding problems, including optimization. The proposed method allows to build the whole car in EasyDyn and optimize, e.g. time of cornering or even time of one race lap because EasyDyn allows using torques and forces needed for power train and cornering, or compute eigenvectors and eigenvalues of the whole drivetrain.

Recommendation for the next work is to achieve smoothness also on the level of acceleration which could be possible by using quartic splines or splines of even higher orders and to implement this method into EasyDyn.

Bibliography

[1] WRIGHT, Thomas Wallace. Elements of Mechanics Including Kinematics, Kinetics and Statics. E and FN Spon (1896). Chapter 1.

[2] SHABANA, Ahmed A. Dynamics of multibody systems. Cambridge: University Press, 1998. Print.

[3] PAPEGAY, Yves A., Jean-Pierre MERLET a David DANEY. Exact kinematics analysis of Car's suspension mechanisms using symbolic computation and interval analysis. Mechanism and Machine Theory [online]. 2005, 40(4), 395-413 [cit. 2020-07-03]. DOI: 10.1016/j.mechmachtheory.2003.07.003. ISSN 0094114X. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0094114X04001338

[4] CHEN, Zhen-Yu. Computer aided design in control systems 1988: selected papers from the 4th IFAC Symposium, Beijing, PRC, 23-25 August 1988. New York: Published for the International Federation of Automatic Control by Pergamon Press, 1989. ISBN 0080357385

[5] JAZAR, Reza N. Numerical Methods in Kinematics. JAZAR, Reza N. Theory of Applied Robotics [online]. Boston, MA: Springer US, 2007, 2007, s. 375-415 [cit. 2020-07-06]. DOI: 10.1007/978-0-387-68964-7_9. ISBN 978-0-387-32475-3. Dostupné z: http://link.springer.com/10.1007/978-0-387-68964-7_9

[6] MCNAMEE, Paul. Automated Memoization in C++ [online]. August, 1998 [cit. 2020-07-06]. Dostupné z: http://pmcnamee.net/c++-memoization.html

[7] SHABANA, Ahmed A. Computational dynamics. 2nd. ed. New York: John Wiley, 2001. ISBN 0471371440.

[8] WEIHAI CHEN, ZHAOJIN WEN, ZHIYUE XU a JINGMENG LIU. Implementation of 3-axis linear interpolation in a FPGA-based 4-axis motion controller. In: 2008 3rd IEEE Conference on Industrial Electronics

and Applications [online]. IEEE, 2008, 2008, s. 1308-1313 [cit. 2020-07-06]. DOI: 10.1109/ICIEA.2008.4582729. ISBN 978-1-4244-1717-9. Dostupné z: http://ieeexplore.ieee.org/document/4582729/

[9] MYOUNG-JUN KIM, MYUNG-SOO KIM a SUNG YONG SHIN. A C/sup 2/-continuous B-spline quaternion curve interpolating a given sequence of solid orientations. In: Proceedings Computer Animation'95 [online]. IEEE Comput. Soc. Press, 1995, s. 72-81 [cit. 2020-07-06]. DOI: 10.1109/CA.1995.393545. ISBN 0-8186-7062-2. Dostupné z: http://ieeexplore.ieee.org/document/393545/

[10] EBERLY, David. Geometric Tools: Rotation Representations [online]. Redmond WA 98052 [cit. 2020-07-06]. Dostupné z: https://www.geometrictools.com/Documentation/RotationRepresentations.pdf

[11] ESCALONA, José L. a Javier F. ACEITUNO. Multibody simulation of railway vehicles with contact lookup tables. International Journal of Mechanical Sciences [online]. 2019, 155, 571-582 [cit. 2020-07-06]. DOI: 10.1016/j.ijmecsci.2018.01.020. ISSN 00207403. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0020740317302485

[12] SCHIEHLEN, W. Multibody System Dynamics [online]. 1(2), 149-188 [cit. 2020-07-06]. DOI: 10.1023/A:1009745432698. ISSN 13845640. Dostupné z: http://link.springer.com/10.1023/A:1009745432698

[13] MULLINEUX, Glen, Robert J. CRIPPS a Ben CROSS. Fitting a planar quadratic slerp motion. Computer Aided Geometric Design [online]. 2020 [cit. 2020-07-06]. DOI: 10.1016/j.cagd.2020.101911. ISSN 01678396. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0167839620300984

[14] KIM, Myung-Soo a Kee-Won NAM. Interpolating solid orientations with circular blending quaternion curves. Computer-Aided Design [online]. 1995, 27(5), 385-398 [cit. 2020-07-06]. DOI: 10.1016/0010-4485(95)96802-S. ISSN 00104485. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/001044859596802S

[15] RAGHAVAN, M. a B. ROTH. Solving Polynomial Systems for the Kinematic Analysis and Synthesis of Mechanisms and Robot Manipulators. Journal of Vibration and Acoustics [online]. 1995, 117(B), 71-79 [cit. 2020-07-06]. DOI: 10.1115/1.2838679. ISSN 1048-9002. Dostupné z: https://asmedigitalcollection.asme.org/vibrationacoustics/article/117/B/71/439056/Solving-Polynomial-Systems-for-the-Kinematic

[16] VERLINDEN, Olivier. Computer-Aided Analysis of Mechanical Systems. 1. Faculté Polytechnique de Mons, Service de Mécanique Rationnelle, 31 Bd Dolez, B-7000 Mons (Belgium): Faculé Polytechnique de Mons, 2016.

[17] BLUNDELL, M. and HARTY, D. Multibody systems approach to vehicle dynamics. Oxford: Elsevier Butterworth-Heinemann, 2004.

[18] SHABANA, Ahmed A. Dynamics of multibody systems. Cambridge, United Kingdom New York, NY: Cambridge University Press, 2020. Print.

[19] Ansys® [Ansys Mechanical Premium], 2020 R1, ANSYS Help 2020 R1, , ANSYS, Inc.

[20] CORKE, Peter I. Robotics, vision and control: fundamental algorithms in MATLAB. Berlin: Springer, 2011. Springer tracts in advanced robotics, v. 73. ISBN 9783642201431.

[21] KORTÜM, W., R.S. SHARP a A.D. DE PATER. Application of Multibody Computer Codes to Vehicle System Dynamics: Progress Report to the 12th IAVSD Symposium on a workshop and Resulting Activities. Society for Engineering and Scientific Education. Oberpfaffenhofen, Cranfield, Delft, 1991.

[22] VERLINDEN, Olivier a Georges KOUROUSSIS. EasyDyn 1.2.4: C++ library for the easy simulation of dynamic problems. Boulevard Dolez 31, 7000 Mons, BELGIQUE, 2008.

[23] GOGU, Grigore. Mobility of mechanisms: a critical review. Mechanism and Machine Theory [online]. 2005, 40(9), 1068-1097 [cit. 2020-05-12]. DOI: 10.1016/j.mechmachtheory.2004.12.014. ISSN 0094114X.

[24] VERLINDEN, Olivier. EasyDyn: Easy Simulation of Dynamic Problems [online]. [cit. 2020-07-01]. https://hosting.umons.ac.be/html/mecara/EasyDyn/index.html

[25] MINAKER, B., n.d. Fundamentals Of Vehicle Dynamics And Modelling :B A Textbook For Engineers With Illustrations And Examples.

[26] WAGNER, Petr, Jan KORDAS, Viktor MICHNA a Jiri KOTZIAN. Motion control for robots based on cubic Hermite splines in real-time. IFAC Proceedings Volumes [online]. 2010, 43(24), 150-155 [cit. 2020-07-06]. DOI: 10.3182/20101006-2-PL-4019.00029. ISSN 14746670. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S1474667015310041

Appendix A

A.1 Derivation of constraints

In this section the following relation is proved:

$$\frac{\partial^i b}{\partial q_k} = \frac{\partial\, ^i\dot{b}}{\partial \dot{q}_k}. \tag{A.1}$$

A.1.1 Relation of $\mathbf{q}_k$ to body $\boldsymbol{i}$

It is considered that q_k relates to body i, then the following relations hold:

$$^1b = \mathbf{x}_{i.a} \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) \qquad / \frac{\partial}{\partial q_k}, \tag{A.2}$$

$$\frac{\partial^1 b}{\partial q_k} = \frac{\partial \mathbf{x}_{i.a}}{\partial q_k} \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) + \mathbf{x}_{i.a} \cdot \frac{\partial \mathbf{e}_{i.a}}{\partial q_k} - \underbrace{\mathbf{x}_{i.a} \cdot \frac{\partial \mathbf{e}_{j.b}}{\partial q_k}}_{=0},$$

$$= \frac{\partial \mathbf{x}_{i.a}}{\partial q_k} \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) + \mathbf{x}_{i.a} \cdot \frac{\partial \mathbf{e}_{i.a}}{\partial q_k}, \tag{A.3}$$

$$\tag{A.4}$$

From the chapter 1 equation (2.27), the following relation also holds:

$$\frac{\partial^1 \dot{b}}{\partial \dot{q}_k} = (\boldsymbol{\delta}_{i.a,k} \times \mathbf{x}_{i.a}) \cdot (\mathbf{e}_{i.a} - \mathbf{e}_{j.b}) + \mathbf{x}_{i.a} \cdot \mathbf{d}_{i.a,k}. \tag{A.5}$$

Now it is a matter of proving two equations

$$\frac{\partial \mathbf{x}_{i.a}}{\partial q_k} = \left(\boldsymbol{\delta}_{i.a,k} \times \mathbf{x}_{i.a}\right), \tag{A.6}$$

$$\mathbf{d}_{i.a,k} = \frac{\partial \mathbf{e}_{i.a}}{\partial q_k}. \tag{A.7}$$

Equation (A.7) is directly fulfilled by (2.20). Equation (A.6) must be multiplied by $\frac{dq_k}{dt}$:

$$\frac{\partial \mathbf{x}_{i.a}}{\partial q_k} = \left(\boldsymbol{\delta}_{i.a,k} \times \mathbf{x}_{i.a}\right) \qquad / \cdot \frac{dq_k}{dt}, \tag{A.8}$$

$$\frac{\partial \mathbf{x}_{i.a}}{\partial q_k} \cdot \frac{dq_k}{dt} = \left(\boldsymbol{\delta}_{i.a,k} \cdot \frac{dq_k}{dt} \times \mathbf{x}_{i.a}\right). \tag{A.9}$$

Using equations (2.26) and (2.22) and assuming that $\mathbf{x}_{i.a}$ and $\boldsymbol{\omega}_{i.a}$ depend only on the orientation angles then it is possible to write

$$\frac{\partial \mathbf{x}_{i.a}}{\partial t} = \boldsymbol{\omega}_{i.a} \times \mathbf{x}_{i.a}, \tag{A.10}$$

$$\begin{pmatrix} \frac{\partial \mathbf{x}_{i.a}}{q_{6i-2}} & \frac{\partial \mathbf{x}_{i.a}}{q_{6i-1}} & \frac{\partial \mathbf{x}_{i.a}}{q_{6i}} \end{pmatrix} \cdot \begin{pmatrix} \dot{q}_{6i-2} \\ \dot{q}_{6i-1} \\ \dot{q}_{6i} \end{pmatrix} =$$

$$= \begin{pmatrix} \boldsymbol{\delta}_{i.a,\,6i-2} & \boldsymbol{\delta}_{i.a,\,6i-1} & \boldsymbol{\delta}_{i.a,\,6i} \end{pmatrix} \cdot \begin{pmatrix} \dot{q}_{6i-2} \\ \dot{q}_{6i-1} \\ \dot{q}_{6i} \end{pmatrix} \times \mathbf{x}_{i.a}. \tag{A.11}$$

The previous equation must be met for each row,

$$\frac{\partial \mathbf{x}_{i.a}}{\partial q_k} \cdot \frac{dq_k}{dt} = \left(\boldsymbol{\delta}_{i.a,k} \cdot \frac{dq_k}{dt} \times \mathbf{x}_{i.a}\right). \tag{A.12}$$

By the previous expression the prove is finished for constraints ${}^1b$, ${}^2b$ and ${}^3b$, because $\mathbf{x}_{i.a}$ can be substituted by $\mathbf{y}_{i.a}$ or $\mathbf{z}_{i.a}$.
It is still considered that q_k relates to body i, then the followng relations hold:

$${}^4b = \mathbf{y}_{i.a} \cdot \mathbf{z}_{j.b} \qquad / \frac{\partial}{\partial q_k}, \tag{A.13}$$

$$\frac{\partial {}^4b}{\partial q_k} = \frac{\partial \mathbf{y}_{i.a}}{\partial q_k} \cdot \mathbf{z}_{j.b} + \mathbf{y}_{i.a} \cdot \underbrace{\frac{\partial \mathbf{z}_{j.b}}{\partial q_k}}_{=0}, \tag{A.14}$$

which is with using the previous prove

$$\frac{\partial^4 b}{\partial q_k} = (\boldsymbol{\delta}_{i.a,k} \times \mathbf{y}_{i.a}) \cdot \mathbf{z}_{j.b}, \tag{A.15}$$

which is directly equal to $\frac{\partial\, ^4\dot{b}}{\partial \dot{q}_k}$.

A.2 Matrix $\mathbf{T}_{0,i}$

It is also possible to get this matrix by a dot product

$$\mathbf{T}_{0,i} = \overline{\mathbf{T}disp}_{0,i} \cdot \overline{\mathbf{T}rotz}_{0,i} \cdot \overline{\mathbf{T}roty}_{0,i} \cdot \overline{\mathbf{T}rotx}_{0,i}, \tag{A.16}$$

where matrices with overlines are defined as

$$\overline{\mathbf{T}disp}_{0,i} = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{A.17}$$

$$\tag{A.18}$$

$$\overline{\mathbf{T}rotz}_{0,i} = \begin{pmatrix} \cos\psi & -\sin\psi & 0 & 0 \\ \sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{A.19}$$

$$\tag{A.20}$$

$$\overline{\mathbf{T}roty}_{0,i} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{A.21}$$

$$\tag{A.22}$$

$$\overline{\mathbf{T}rotx}_{0,i} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{A.23}$$

A.3 Class and methods for interpolation

```
#ifndef TABMOTIONINTERPOLATE_H
#define TABMOTIONINTERPOLATE_H

#include <iostream>
#include <iomanip>
#include <math.h>
#include <stdlib.h>
#include <EasyDyn/vec.h>

using namespace std;

/**********************************************************/
/*                                                        */
/*          Definition of the class TabMotionInterpolate  */
/*                                                        */
/**********************************************************/

// Definition of class vec (3D vector)
class TabMotionInterpolate{
  private :
    int NPointsTab;
    double *uTab;
    mth *T0ATab;
    vec *dATab, *deltaATab;
  public :
    // Constructor: the table is initiated from data stored in text file
    // DataFile. Other types of constructors could be defined as well
    TabMotionInterpolate(const char* DataFileName);
    // Destructor: so as to release allocated memory
    ~TabMotionInterpolate();
    // Function to get interpolated quantities
    void GetInterpolatedQuantities(double u, mth &Tout, vec &dout,
     vec &deltaout, vec &ddoutdu, vec &ddeltaoutdu);
  };
/*********************************************************/
```

```
/* defining of
  double  ui, ui1;
  mth Ti, Ti1;
  vec di, di1;
  vec deltai, deltai1;
*/

/******************--- interpolation ---******************/
```

```
// interpolation of rotation, delta and ddelta/du

trot Ri  = Ti.R;
trot R1i = Ti1.R;

trot Rif = Ri.inv()*R1i; // transpose(Ri)*R1i;

double theta = acos((Rif.r11+Rif.r22+Rif.r33-1)/2);

   vec r;
   r.x = 1/(2*sin(theta))*(Rif.r32-Rif.r23);

   r.y = 1/(2*sin(theta))*(Rif.r13-Rif.r31);

   r.z = 1/(2*sin(theta))*(Rif.r21-Rif.r12);

   double ksi = (u-ui)/(ui1-ui);

    vec deltai_i   = Ri.inv()*deltai;
    vec deltai1_i1 = R1i.inv()*deltai1;

    trot Rrotaxis1, Rrotaxis2, Rrotaxis3;

    fRrotaxis(deltai_i/deltai_i.length(),
    deltai_i.length()*h10(ksi)*(ui1-ui),Rrotaxis1);
    trot R1 = Rrotaxis1;

    fRrotaxis(r,theta*h01(ksi),Rrotaxis2);
    trot R2 = Rrotaxis2;

    fRrotaxis(deltai1_i1/deltai1_i1.length(),
    deltai1_i1.length()*h11(ksi)*(ui1-ui),Rrotaxis3);
    trot R3 = Rrotaxis3;


    trot R = Ri*R1*R2*R3;
    Tout.R = R;

    vec delta10 = deltai;  //transpose(deltai) but it should be correct
    vec delta20 = Ri*R1*r;
    vec delta30 = Ri*R1*R2*deltai1_i1 ;

    deltaout = delta10*dh10(ksi)+delta20*dh01(ksi)*theta/(ui1-ui)
   +delta30*dh11(ksi);
```

```
vec ddelta10 = deltaout*ddh10(ksi)*1/(ui1-ui);
vec ddelta20 = Ri*R1*r*ddh01(ksi)*theta/((ui1-ui)*(ui1-ui))+
(delta10^delta20);
vec ddelta30 = Ri*R1*R2*deltai1_i1*ddh11(ksi)*1/(ui1-ui)+
((delta10+delta20)^delta30);

ddeltaoutdu = ddelta10+ddelta20+ddelta30;

//interpolation of position, d and dd/du

vec eAi = Ti.e;
vec eAi1 = Ti1.e;

vec e = eAi*h00(ksi)+eAi1*h01(ksi)+di*h10(ksi)*(ui1-ui)
+di1*h11(ksi)*(ui1-ui);
Tout.e = e;

dout  = eAi*dh00(ksi)*1/(ui1-ui)+eAi1*dh01(ksi)*1/(ui1-ui)+
di*dh10(ksi)+di1*dh11(ksi);

ddoutdu = eAi*ddh00(ksi)*1/((ui1-ui)*(ui1-ui))
+eAi1*ddh01(ksi)*1/((ui1-ui)*(ui1-ui))
+di*ddh10(ksi)*1/(ui1-ui)+di1*ddh11(ksi)*1/(ui1-ui);
```