LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
LUT School of Energy Systems
LUT Mechanical Engineering

Niko Niemi

COMPARISON OF OPEN DYNAMICS ENGINE, CHRONO AND MEVEA IN SIMPLE MULTIBODY APPLICATIONS

Examiners: Professor Aki Mikkola
D. Sc. (Tech.) Grzegorz Orzechowski

ABSTRACT

Lappeenranta University of Technology
LUT School of Energy Systems
LUT Mechanical Engineering

Niko Niemi

Comparison of Open Dynamics Engine, Chrono and MeVEA in simple multibody applications

Master's thesis

2017

80 pages, 24 figures, 11 tables and 1 appendix

Examiners: Professor Aki Mikkola
D. Sc. (Tech.) Grzegorz Orzechowski

Keywords: multibody dynamics, physics engine, Open Dynamics Engine, MeVEA, Chrono

In this thesis, simulation speed and accuracy of Open Dynamics Engine (ODE), Chrono and MeVEA are compared. To this end, a simple multibody model is created for benchmarking purposes in each of the software. The feasibility of ODE in biomechanical modeling is also studied by creating two hands in the software environment. The theory behind multibody dynamics and the physics engines is reviewed to gain a better understanding of the issues affecting performance. Emphasis is on Chrono and ODE as they are open-source software.

Each software is benchmarked using the same multibody system, which is a simple slider-crank mechanism consisting of three bodies. A torque is applied to the crank, which moves the slider back and forth and this is simulated for a set amount of simulation steps corresponding to two seconds in real-time. In the speed comparison, the simulation is run to test how long it takes to solve it. In the accuracy benchmark, the speed of the slider at each step is compared to a widely used multibody simulation software, Adams. Both benchmarks test different simulation step sizes. Results show that in terms of speed MeVEA is the fastest, ODE is second and Chrono is the slowest. ODE is the most accurate and Chrono and MeVEA yield similar results. However, ODE is not accurate until the step size is very small, which limits the usability in an environment that requires accurate and fast performance. The hand model in ODE works well, however the modeling is quite tedious as there is no graphical user interface. Future improvements to physics engines might include parallel computing.

This thesis was part of a collaboration project with Lappeenranta University of Technology and Aalto University, which aims to solve a problem of combining movement artificial intelligence and multibody dynamics simulation. High-level goal of the project is to provide a Virtual Coach, which can help a human in a movement-related task in real-time.

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
LUT Energiajärjestelmät
LUT Kone

Niko Niemi

Open Dynamics Enginen, Chrono ja MeVEAn vertailu yksinkertaisissa monikappaledynamiikan sovelluskohteissa

Diplomityö

2017

80 sivua, 24 kuvaa, 11 taulukkoa ja 1 liite

Tarkastajat: Professori Aki Mikkola
TkT Grzegorz Orzechowski

Hakusanat: monikappaledynamiikka, fysiikkamoottori, Open Dynamics Engine, MeVEA, Chrono

Tässä diplomityössä Open Dynamics Enginen (ODE), Chronon ja MeVEAn nopeutta ja tarkkuutta vertaillaan. Vertailua varten luodaan edellä mainituissa ohjelmistoissa yksinkertainen monikappaledynaaminen malli. Myös biomekaanisen mallinnuksen toteutettavuutta tutkitaan ODE:ssä luomalla kaksi kättä ohjelmistoympäristössä. Teoriaa monikappaledynamiikan ja fysiikkamoottoreiden taustalta tutkitaan, jotta saataisiin kattava näkemys suorituskyvyn haittatekijöistä. Erityisesti Chronoa ja ODE:ä tutkitaan, sillä ne ovat avoimen lähdekoodin ohjelmistoja.

Kutakin ohjelmistoa vertaillaan käyttäen samaa monikappalesysteemiä, joka on yksinkertainen kolmen kappaleen kampimekanismi. Kampeen asetetaan vääntömomentti, joka tällöin liikuttaa lohkoa edestakaisin ja tätä simuloidaan suorittamalla tietty määrä simulaatioaskelia, mikä vastaa kahta sekuntia reaaliajassa. Nopeusvertailussa selvitetään simulaation suoritukseen kuluva aika. Tarkkuusvertailussa liikkuvan lohkon nopeutta jokaisella simulaatioaskeleella verrataan laajalti käytettyyn monikappalesimulointi ohjelmistoon, Adamsiin. Molemmat vertailut kokeilevat eri simulaatioaskelia. Tulosten mukaan nopein vertailussa on MeVEA, toisena on ODE ja viimeisenä Chrono. ODE on puolestaan tarkin ja Chrono sekä MeVEA tuottivat tarkkuudeltaan samankaltaisia tuloksia. ODE on tosin tarkka vain erittäin pienillä simulaatioaskeleen arvoilla, mikä rajoittaa sen käytettävyyttä, kun vaaditaan tarkkuutta ja nopeutta. Käsimalli ODE:ssä toimii hyvin, vaikkakin mallinnus on varsin hidasta, koska graafista käyttöliittymää ei ole. Tulevaisuudessa fysiikkamoottorit saattavat hyödyntää rinnakkaislaskentaa.

Tämä diplomityö on osa Lappeenrannan teknillisen yliopiston sekä Aalto yliopiston yhteistä projektia, jonka päämääränä on yhdistää liikkumisen tekoäly sekä monikappaledynaaminen simulointi. Päämääränä on luoda virtuaalinen valmentaja, joka auttaisi ihmistä liikkumispainotteisessa tehtävässä reaaliajassa.

ACKNOWLEDGEMENTS

I would like to thank Professor Aki Mikkola and postdoctoral researcher Grzegorz Orzechowski for providing me an intersting topic of research and guiding me throughout the thesis process. I would also like to thank the good people from the research group of Machine Design for tirelessly helping me with problems I faced.

I cannot thank my family enough for the boundless support that I've got throughout my studies and life. Thank you to Kristiina for enduring and helping me during stressful times. As my time in Lappeenranta draws to a close, I cannot but reminisce the good times I spent with wonderful people. Hopefully a new chapter in my life will bring equal prosperity.

Niko Niemi

Lappeenranta 1.9.2017

TABLE OF CONTENTS

ABSTRACT

TIIVISTELMÄ

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

1 INTRODUCTION 9

1.1 Backgrounds 9

1.2 Research problem and scope of the study 11

2 LITERATURE REVIEW 12

2.1 Multibody system theory – Kinematic analysis 12

2.1.1 Description of rotation 15

2.1.2 Kinematic constraints 24

2.2 Multibody system theory – Dynamic analysis 26

2.2.1 Newtonian mechanics and Newton-Euler 26

2.2.2 Lagrangian mechanics and Lagrange multipliers 28

2.3 Physics engines' core components 35

2.3.1 Integrator 35

2.3.2 Simulator paradigm 37

2.3.3 Object representation 38

2.3.4 Material properties 38

2.3.5 Constraint implementation 39

2.4 Open Dynamics Engine 39

2.4.1 ODE integrator 39

2.4.2 Simulator paradigm 39

2.4.3 Object representation and collision 40

2.4.4 Material properties 41

2.4.5 Constraint implementation 41

2.5 Project Chrono 43

2.5.1 Chrono integrator 45

2.5.2 Simulator paradigm 46
2.5.3 Material properties 47
2.5.4 Object representation and collision 47
2.5.5 Constraint implementation 47
3 RESEARCH METHODS 48
3.1 Literature review 48
3.2 Computer simulation 48
3.3 Third-party benchmarks 54
4 RESULTS AND ANALYSIS 57
4.1 Spatial slider-crank modeling 57
4.2 Hand modeling 62
4.3 Simulation speeds 64
4.4 Simulation accuracy 66
4.5 Analysis 69
4.6 Reliability, validity and error analysis of the study 72
4.7 Further research 73
5 CONCLUSIONS 74
REFERENCES 76
APPENDICES
Appendix I: The source codes for benchmarks and hand model.

LIST OF SYMBOLS AND ABBREVIATIONS

$\boldsymbol{A}^i$	Rotation matrix
a	Acceleration
$\boldsymbol{C}$	Set of constraint equations
$\boldsymbol{C}_q$	Jacobian matrix of constraint equation
$\boldsymbol{C}_t$	Time differentiation of constraint equation
c_i	Runge-Kutta step size multiplier
CFM	Constraint force mixing parameter
CPU	Central Processing Unit
ERP	Error-reducing parameter
F	Force
f_N	Normal force
f_T	Tangential force
GPU	Graphics Processing Unit
$\boldsymbol{H}$	Matrix consisting of mass and Jacobian
h	Size of time step
$\boldsymbol{I}$	Identity matrix
$\boldsymbol{I}_{cr}$, $\boldsymbol{I}_{co}$, $\boldsymbol{I}_s$	Matrices of inertia for crank, connecting rod and slider
i, j, k	Imaginary units
$\boldsymbol{J}$	Matrix of inertia
k_i	Runge-Kutta weighted mean
$\boldsymbol{L}$	Lagrange's function
LCP	Linear complementarity problem
m	Mass
n_c	Number of constraint equations
n	Number of coordinates in a multibody system
ODE	Open Dynamics Engine
O(n)	Big O notation, solution time proportional to system size
OPCODE	Optimized Collision Detection
$\dot{p}^i$	Rate of change of momentum
q	Quaternion

$\boldsymbol{q}$	Vector of generalized coordinates
$\boldsymbol{Q}$	Vector of generalized forces
$\boldsymbol{Q}_c$	Vector of constraint force
$\boldsymbol{Q}_d$	Vector of constraint equation differentiated twice over time
$\boldsymbol{Q}_e$	Vector of externally applied force
$\boldsymbol{r}_p^i$	Position of particle *p* in global coordinates
$\boldsymbol{R}^i$	Position vector of the body origin
δr^i	Virtual displacement
$\boldsymbol{r}, \bar{\boldsymbol{r}}, \Delta\boldsymbol{r}, \boldsymbol{b}_1, \boldsymbol{b}_2$	Vector between two points in Rodriguez' method
SOR	Successive Over-Relaxation
T	Torque
$\boldsymbol{T}$	Kinetic energy
t	Time
$\bar{\boldsymbol{u}}^{\boldsymbol{i}}$	Distance of a point from body reference
μ	Friction coefficient
$\boldsymbol{U}$	Potential energy
$\boldsymbol{v}$	Axis of rotation
$\boldsymbol{X}, \boldsymbol{\xi}, \boldsymbol{\eta}, \boldsymbol{\zeta}$	Vector of coordinate system
φ, θ, ψ	Rotation angle about an axis, Euler angles
$\boldsymbol{\gamma}$	Rodriguez parameter
$\boldsymbol{\lambda}$	Lagrange multiplier
$\boldsymbol{\theta}$	Generalized rotational coordinates

1 INTRODUCTION

In this thesis, mechanical models created using Chrono, MeVEA and Open Dynamics Engine environment are presented. Purpose of this is to compare speeds of these physics engines and eventually combine two research objectives. The goal of the first research project is to simulate biomechanically sufficiently accurate multibody dynamics and the second research project utilizes artificial intelligence to optimize humanoid movement in computer animation. Tying together these two topics of research will have the potential to revolutionize movement-centric research and its applications.

1.1 Backgrounds

Perttu Hämäläinen is a professor of computer games at the Aalto University and is participating in efforts to develop digitally augmented sports and so called exergames. In addition to movement design in animation for games or movies, this technology would have the potential to solve problems in fields of medicine, human-computer interaction and sports et cetera. Past research by Hämäläinen has resulted in many demonstrations such as the Augmented Climbing Wall. Flagship demonstrator of these combined studies is related to climbing. (Kajastila, Holsti & Hämäläinen 2016, p. 758.)

Multibody dynamics deals with multiple bodies which can be connected to each other through joints, has been studied quite heavily in the last decades. Even though the mathematics and dynamics behind rigid bodies have been known since Newton, the ever-increasing need for computational efficiency still drives the research. (Erleben 2005, p. 7.) Motivation to this study should be therefore quite clear since combining artificial intelligence with computationally heavy algorithms, such as Monte Carlo for testing several iterations, results in an increased need in faster computational speed. Therefore, it is crucial to determine which physics engine is the most suitable to perform fast multibody dynamics simulations.

In addition to the speed and effectiveness of the simulation method, the models of the human need to be more efficient. Modelling the human body, which is a musculoskeletal system, will dispose of the need to capture motion data, which has prohibited some fields of research

where motion capture is not a possibility or is very costly. An example of such a field is safety related issues, where motion capture is obviously not an option.

High level goals of this combined project aim at solving multiple difficult problems, including:

- Sport and exercise, for example how to optimize an athlete's movement.
- Human-computer interaction, for example how does the AI react to human player's unanticipated movement?
- Movement design, for example how to dodge a hazard.
- Medical, for example how should a person with a specific injury move or lift a load.

(Hämäläinen 2016, p. 4.)

As stated earlier, the goal of this project is to tie together two separate subprojects. The first one concentrating on movement artificial intelligence to devise a control strategy of a humanoid by forward simulating the resulting system states. This is achieved by using novel Monte Carlo sampling methods. Simulations within the Monte Carlo sampling must be faster than real-time since it tests several different alternatives before eventually finding out the optimal. In figure 1 the method is demonstrated. The object in the figure on the left is to find the optimal path to dodge the ball and on the right to move between the circles without touching them.

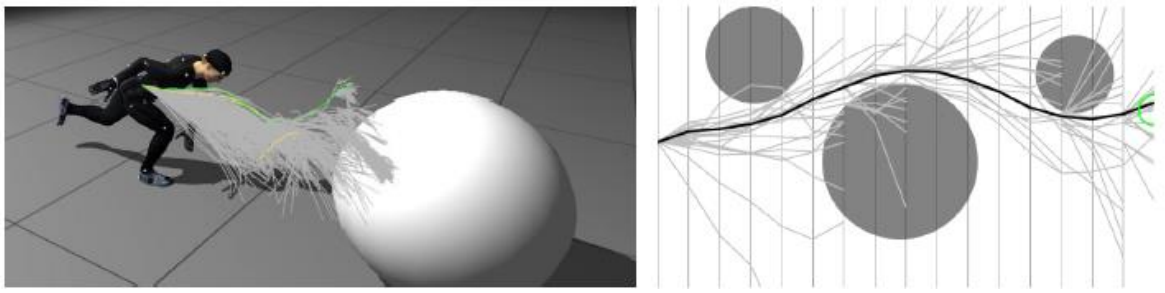

Figure 1. Method of testing several different paths (Hämäläinen 2016, p. 7).

The second subproject, which closely relates to the topic of this thesis, focuses on the efficiency of biomechanical modeling and simulation so that the artificial intelligence system can accurately simulate the optimal path. Second subproject can be split into two main parts, the first part simulates the humanoid on a lower-detail level, which is used in the motion AI

algorithm. In the first part, high accuracy is not a priority, rather the simulation speed is the crucial factor. In the second part, a higher-detail post-process simulation is performed, which will accurately calculate the humanoids movement based on the trajectory the motion AI calculated from the first part.

In the aforementioned first part of the second subproject, quite significant approximations have to be performed, while still maintaining plausible behavior. In terms of biomechanical detail this means neglecting individual muscle activation and instead considering joint torques. (Hämäläinen 2016, pp. 6-9.) It is worth mentioning that muscle and tendon models of the human body may prove to be too computationally heavy even for the second part of the subproject, though parallel computing could be the answer to this problem. Parallel computing utilizes the Graphics Processing Unit (GPU) in addition to the Central Processing Unit (CPU) giving a much-needed boost to computational resources at use.

1.2 Research problem and scope of the study

The objective of this thesis is to compare computational efficiency of the Open Dynamics Engine, Chrono::Engine and MeVEA. To this end, a simple mechanical system is modeled and analyzed using above mentioned three software. Based on the premise of the entire collaboration project, a simple mechanical system will resemble a slider-crank system. In addition to these topics of research, the feasibility of simplified biomechanical modeling in Open Dynamics Engine environment is investigated. Scope of the study is limited to one simple multibody benchmark study involving the aforementioned physics engines and the simplified biomechanical model is realized only in Open Dynamics Engine (ODE).

The research should answer the following questions:

- How fast is the Open Dynamics Engine compared to other physics engines, e.g. Chrono::Engine or MeVEA? How about the accuracy?
- How does the future development of physics engines look like in terms of computational speed?

2 LITERATURE REVIEW

Literature review in this thesis will be split into three main parts. One of the main themes in this thesis is constrained rigid multibody dynamics and it is covered in the first part. Topics for the first part include description of rotation, constraints, analytical techniques for solving dynamical systems and brief introduction to contact problems. In the second part, we define the crucial properties which form a base for a physics engine. Third part will concentrate on three different physics engines and their performance in multibody dynamics based on previous two parts of the literature review. These physics engines are:

- Open Dynamics Engine,
- Project Chrono,
- MeVEA.

Open Dynamics Engine was originally developed by Russell Smith and it is an open-source physics software development kit. It is used for simulating rigid bodies with focus on fast simulation along with robust and stable behavior. (Smith 2006, p. 1.)

Alike ODE, Project Chrono is open-source as well. Chrono is being developed in the University of Wisconsin-Madison and University of Parma, Italy. It comprises of several different modules, which each serve a special function. In the heart of it is the Chrono::Engine. (Project Chrono – An Open Source Physics Engine 2016.)

MeVEA is a Finnish company providing simulation solutions. If was founded in 2005 as a Lappeenranta University of Technology spin-off company. Their software is used for real-time dynamics simulation. (Software for Real-Time Simulation | MeVEA 2016.)

2.1 Multibody system theory – Kinematic analysis

A multibody system consists of an arbitrary number of bodies, which may have been connected to each other and/or the world around them via connection elements, which are from here on referred to as constraints. Constraints are joints and they limit the relative motion of pairs of bodies, but they do not have a mass. Multibody system can be either dynamic or kinematic. Dynamic multibody system consists of bodies which have a certain

mass and the bodies can be subject to forces. Usually dynamic multibody problems are also categorized based on the known and wanted values and these categories are called forward and inverse problem. The latter subgroup knows where to go, but does not know the forces and torques needed to do it. Meanwhile the forward problem knows the forces and torques subjected to the body but does not know the end position of the body. (Chaudhary & Saha 2009, pp. 1-3.) In this thesis constrained rigid multibody dynamics with mainly forward oriented problems are applied. Unlike dynamic system, kinematic system studies motion without considering masses or forces (Wittenburg 2008, p. 13). In figure 2 below a comprehensive demonstration of a multibody system is depicted. It includes external forces, springs, dampers and joints.

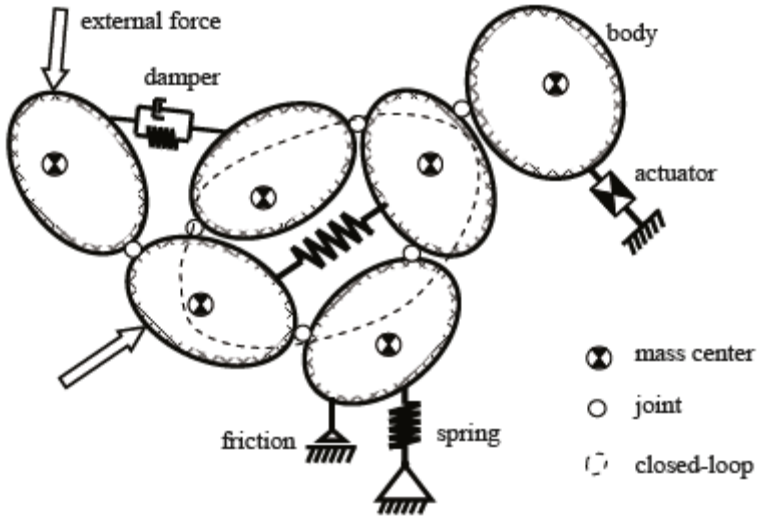

Figure 2. A demonstration of a multibody system with joints, springs, dampers and forces (Chaudhary & Saha 2009, p. 2).

Even nowadays multibody dynamics is a somewhat researched topic even though the basic mathematics behind it have been known for almost 400 years since the days of Newton. The motivation for its research has been stated in the introduction; the ever-increasing need for computational efficiency and growing proportions of simulation call for more advanced algorithms capable of solving problems fast and reliably while being tolerant for faulty configurations. (Erleben 2005, pp. 5-7.)

Principles of motion of a multibody system were characterized by Euler and D'Alembert in the eighteenth century. Governing equations were formed based on Newton laws of linear

motion as well as Euler's equations of rotational motion. Combining the efforts of Newton and Euler formed the famous and even nowadays widely used dynamic formulation known as Newton-Euler equations of motion. Lagrange contributed by forming a method of systematic analysis of constrained multibody systems. Using the energy concept, he analytically derived generalized equations of motion. Nowadays many more formulations exist which are based on the governing equations. However, the variety of dynamic formulations is a result of mixing and combining different basic principles, such as selection of description of coordinates and principles of mechanics (i.e. Lagrange's equations, Newton-Euler equations, virtual work or D'Alembert's principle). (Chauhary & Saha 2009, p. 3.)

Now if we start of by defining a kinematic system, i.e. not considering mass or forces exerted to a body, in three-dimensional space, we need to use six independent coordinates, three translational and three rotational. Once these coordinates are identified, the global position of an arbitrary point on the body can be deduced in terms of these coordinates. In figure 3, a rigid body is depicted and point P on the body can be written in Cartesian coordinates as (Shabana 1998, p. 11):

$$\boldsymbol{r}_p^i = \boldsymbol{R}^i + \boldsymbol{A}^i \bar{\boldsymbol{u}}^i \tag{1}$$

In equation 1, $\boldsymbol{R}^i$ is the position of the body *i's* origin, $\boldsymbol{A}^i$ is the transformation matrix, which transforms the body coordinate system to the global coordinate system (this will be discussed in the following chapters) and $\bar{\boldsymbol{u}}^i$ is the position of point P from the body reference origin with respect to the body coordinate system (Shabana 1998, p. 11).

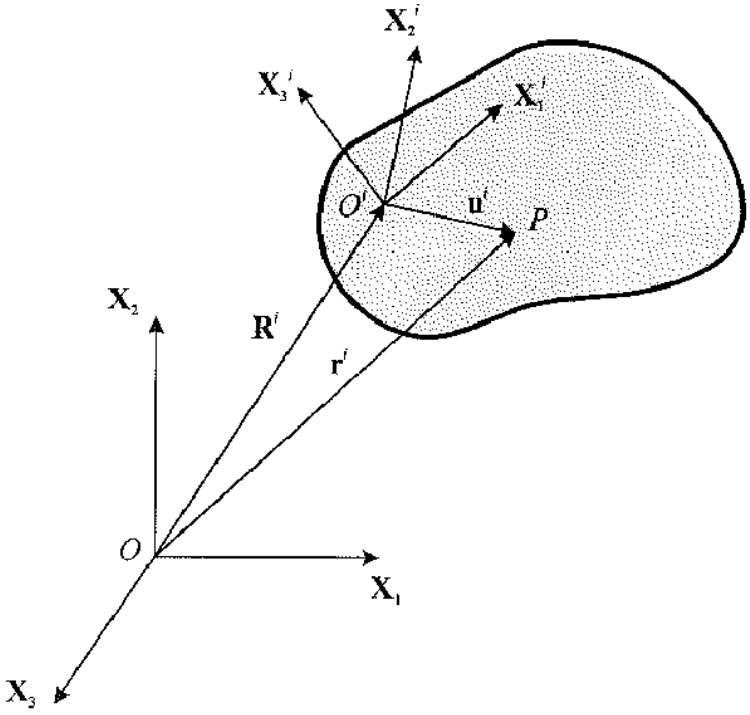

Figure 3. An arbitrary shaped body in three-dimensional space (Shabana 1998, p. 11).

2.1.1 Description of rotation

As stated earlier, the transformation matrix $\boldsymbol{A}^i$ transforms from body coordinate system, denoted as $X_1^i\, X_2^i\, X_3^i$, to global coordinate system (also known as absolute coordinate system), denoted as $X_1X_2X_3$, and thus generality is not lost. Subscripts denote the axes and they are equivalent to X, Y and Z commonly used in literature. (Shabana 1998, pp. 29-30.) There are numerous ways to form the transformation matrix such as:

- Rodriquez formula and parameters
- Euler angles
- Quaternions

(Wittenburg 2008, p. 9, 20.)

These methods are now discussed. If we take a body in three-dimensional space and rotate it about an axis, we can portray it as in figure 4 below.

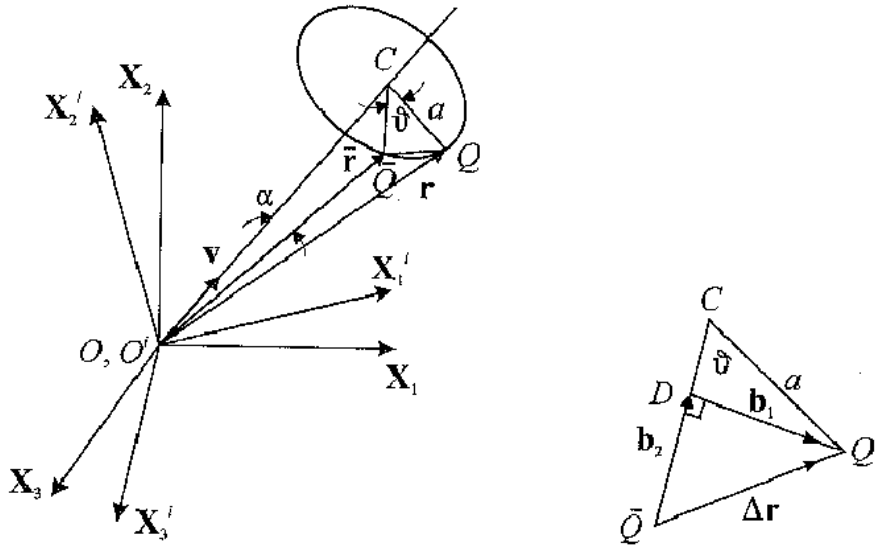

Figure 4. Formulating rotation matrix for a body rotated about an axis in three-dimensional space (Shabana 1998, p. 30).

In figure 4, the body has been rotated at an angle θ about the axis OC and as a result, point $\bar{Q}$ moves to point *Q*. $\Delta\boldsymbol{r}$ denotes the position vector from $\bar{Q}$ to *Q*. Vector $\boldsymbol{r}$ can then be written as (Shabana 1998, p. 30):

$$\boldsymbol{r} = \bar{\boldsymbol{r}} + \Delta\boldsymbol{r} \tag{2}$$

In the equation 2 above $\bar{\boldsymbol{r}}$ is the original vector to point $\bar{Q}$ before rotating the body at an angle θ. According to the figure 4, $\Delta\boldsymbol{r}$ can also be written as a sum of vectors $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$:

$$\Delta\boldsymbol{r} = \boldsymbol{b}_1 + \boldsymbol{b}_2 \tag{3}$$

In equation 3, $\boldsymbol{b}_1$ is perpendicular to the plane $OC\bar{Q}$ and therefore its direction can be written as a cross product of the vector $\boldsymbol{v}$ and $\bar{\boldsymbol{r}}$, $(\boldsymbol{v} \times \bar{\boldsymbol{r}})$. $\boldsymbol{v}$ describes the axis of rotation and is a unit vector. $\boldsymbol{b}_1$ therefore has a magnitude of:

$$|\boldsymbol{b}_1| = a \sin(\theta) \tag{4}$$

Based on figure 4, it can be shown that:

$$a = |\bar{\boldsymbol{r}}| \sin(\alpha) = |\boldsymbol{v} \times \bar{\boldsymbol{r}}| \tag{5}$$

Thus, $\boldsymbol{b}_1$ is:

$$b_1 = a \sin(\theta) \frac{v \times \bar{r}}{|v \times \bar{r}|} = (v \times \bar{r}) \sin(\theta) \tag{6}$$

The magnitude of the vector $\boldsymbol{b}_2$ can be written as:

$$|\boldsymbol{b}_2| = a - a\cos(\theta) = (1 - \cos(\theta))a = 2a \sin^2 \frac{\theta}{2} \tag{7}$$

Noting that $\boldsymbol{b}_2$ is perpendicular to unit vector $\boldsymbol{v}$ and DQ, $\boldsymbol{b}_2$ is the vector (Shabana 1998, p. 31):

$$\boldsymbol{b}_2 = 2a \sin^2 \frac{\theta}{2} \frac{\boldsymbol{v} \times (\boldsymbol{v} \times \bar{\boldsymbol{r}})}{a} = 2[\boldsymbol{v} \times (\boldsymbol{v} \times \boldsymbol{r})] \sin^2 \frac{\theta}{2} \tag{8}$$

Vectors $\boldsymbol{v}$ and $\boldsymbol{r}$ can be written in terms of their unit components using identity. This produces skew symmetric matrices. Thus letting:

$$\boldsymbol{v} \times \bar{\boldsymbol{r}} = \tilde{\boldsymbol{v}}\bar{\boldsymbol{r}} = -\tilde{\bar{\boldsymbol{r}}}\boldsymbol{v} \tag{9}$$

leads to:

$$\tilde{\boldsymbol{v}} = \begin{bmatrix} 0 & -\boldsymbol{v}_3 & \boldsymbol{v}_2 \\ \boldsymbol{v}_3 & 0 & -\boldsymbol{v}_1 \\ -\boldsymbol{v}_2 & \boldsymbol{v}_1 & 0 \end{bmatrix}, \tilde{\bar{\boldsymbol{r}}} = \begin{bmatrix} 0 & -\bar{\boldsymbol{r}}_3 & \bar{\boldsymbol{r}}_2 \\ \bar{\boldsymbol{r}}_3 & 0 & -\bar{\boldsymbol{r}}_1 \\ -\bar{\boldsymbol{r}}_2 & \bar{\boldsymbol{r}}_1 & 0 \end{bmatrix} \tag{10}$$

In equation 10 the subscripts denote the component of the unit vector. After this the equation 2 can be written as:

$$\boldsymbol{r} = \left[\boldsymbol{I} + \tilde{\boldsymbol{v}} \sin\theta + 2(\tilde{\boldsymbol{v}})^2 \sin^2 \frac{\theta}{2}\right] \bar{\boldsymbol{r}} \tag{11}$$

In equation 11, $\boldsymbol{I}$ is a 3*3 identity matrix. Further simplifying the equation leads to:

$$\boldsymbol{r} = \boldsymbol{A}\bar{\boldsymbol{r}} \tag{12}$$

Thus, the rotation matrix $\boldsymbol{A}$ can be written as:

$$\boldsymbol{A} = \left[\boldsymbol{I} + \tilde{\boldsymbol{v}}\sin\theta + 2(\tilde{\boldsymbol{v}})^2\sin^2\frac{\theta}{2}\right] \tag{13}$$

This formula in equation 13 for the rotation matrix is known as the Rodriquez formula. Determining the rotation therefore requires the knowledge of the axis and angle of rotation. For that reason, the number of variables is four, three variables for the components of the unit vector and one variable for the angle of rotation around the unit vector. (Shabana 1998, p. 31.)

Similarly, one of the more commonly used descriptions of rotation in multibody dynamics, the quaternion approach, consists of four parts. Quaternions were invented by Hamilton in 1844 as an extension of complex numbers and it was discovered soon after that that they could be used to describe rotation. Quaternions consist of three imaginary and one real part, therefore by definition quaternions are hyper-complex numbers of rank 4. (Coutinho 2013, p. 306.) The core rule to forming the quaternions is (Coutinho 2013, pp. 307-308):

$$i^2 = j^2 = k^2 = ijk = -1 \tag{14}$$

In equation 14 *i, j* and *k* denote imaginary numbers. Using this equation quaternions can be formed:

$$q = q_0 + q_1 i + q_2 j + q_3 k \tag{15}$$

Quaternion in equation 15 consists of a real part, q_0, and three imaginary parts, $q_1 i$, $q_2 j$, $q_3 k$. To help visualize how and why quaternions work, one can imagine a 2-D complex plane case commonly used in many fields of engineering and mathematics, where the

vector's angle is expressed using imaginary units and the magnitude is expressed with a real part. In quaternion approach, this is expanded to a three-dimensional space.

An extension of the quaternion approach in description of rotation is commonly referred to as Euler parameters, which can be derived based on the transformation matrix on equation 13 using the trigonometric identity (Shabana 1998, p. 31):

$$\sin(\theta) = 2\sin\frac{\theta}{2}\cos\frac{\theta}{2} \tag{16}$$

Using the following four Euler parameters, equation 13 can be rewritten (Shabana 1998, p. 32):

$$\begin{cases} \theta_0 = \cos\dfrac{\theta}{2}, & \theta_1 = v_1\sin\dfrac{\theta}{2} \\ \theta_2 = v_2\sin\dfrac{\theta}{2}, & \theta_3 = v_3\sin\dfrac{\theta}{2} \end{cases} \tag{17}$$

Using this relation, transformation matrix in terms of Euler parameters can be written as:

$$\boldsymbol{A} = \begin{bmatrix} 1-2(\theta_2)^2-2(\theta_3)^2 & 2(\theta_1\theta_2-\theta_0\theta_3) & 2(\theta_1\theta_3+\theta_0\theta_2) \\ 2(\theta_1\theta_2+\theta_0\theta_3) & 1-2(\theta_1)^2-2(\theta_3)^2 & 2(\theta_2\theta_3-\theta_0\theta_1) \\ 2(\theta_1\theta_3-\theta_0\theta_2) & 2(\theta_2\theta_3+\theta_0\theta_1) & 1-2(\theta_1)^2-2(\theta_2)^2 \end{bmatrix} \tag{18}$$

In the previous explanations of rotation matrices only one rotation about one axis was considered. However, in a case of successive rotations about two different axes that are not parallel, measures need to be taken to portray the spatial position and rotation of a body correctly. Therefore, it is critical to understand that the order of successive rotations is very important. If we have two transformation matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ it can be demonstrated that:

$$\boldsymbol{A}_2\boldsymbol{A}_1 \neq \boldsymbol{A}_1\boldsymbol{A}_2 \tag{19}$$

Mathematical standpoint of the equation 19 is neglected but instead it is proved in figure 5, which demonstrates that the rotation sequence cannot be arbitrary.

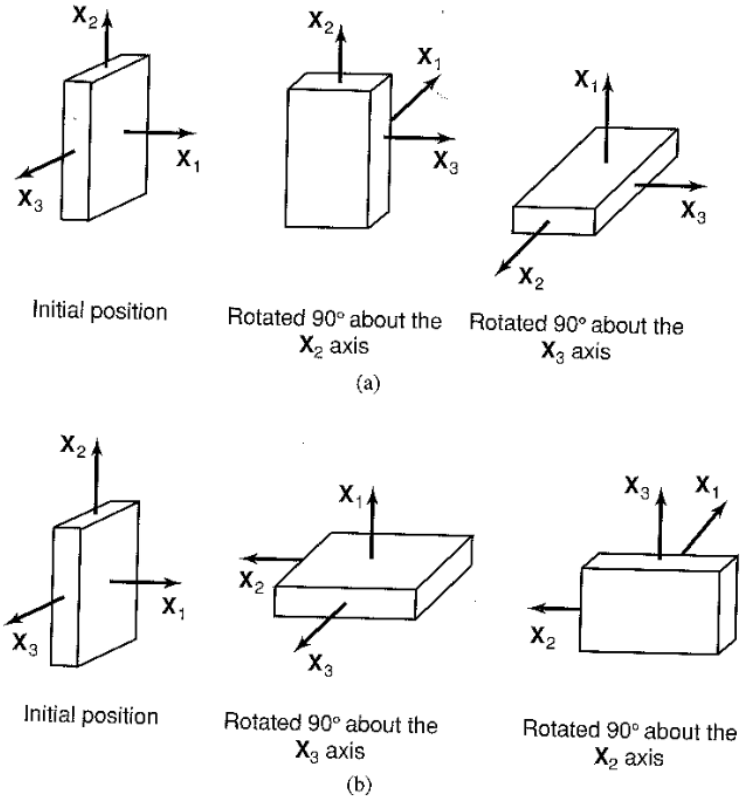

Figure 5. Demonstrating equation 19 – effect of rotation sequence (Shabana 1998, p. 44).

As it can be witnessed in figure 5, the order of rotation affects the outcome, unless the axes of rotation are parallel. There exist two methods to describe successive rotations and these are single-frame method and multi-frame method. Basic difference between these two is that in single-frame method one fixed coordinate system is used and after each rotation the vectors on the rigid body are defined in the previously determined coordinate system, which usually is the global coordinate system. In the multi-frame method, there is no need to transform the axes of rotation to the global coordinate system after rotations. This means that the transformation can be portrayed in terms of body fixed coordinate system. (Shabana 1998, pp. 42-47.)

Previously introduced formulations used four variables to describe the rotation. However, there are methods that only require three parameters. The ones that are discussed in the following section are Rodriguez parameters and Euler angles.

Rodriguez parameters will be derived based on the equation 13 using a definition of the vector **γ** of Rodriguez parameters (Shabana 1998, p. 63):

$$\boldsymbol{\gamma} = \boldsymbol{v}\tan(\frac{\theta}{2}) \tag{20}$$

where $\boldsymbol{v}$ is the unit vector the body is rotated about and θ is the angle of rotation. By writing **γ** in terms of its components:

$$\begin{cases} \boldsymbol{\gamma}_1 = \boldsymbol{v}_1\tan(\frac{\theta}{2}) \\ \boldsymbol{\gamma}_2 = \boldsymbol{v}_2\tan(\frac{\theta}{2}) \\ \boldsymbol{\gamma}_1 = \boldsymbol{v}_2\tan(\frac{\theta}{2}) \end{cases} \tag{21}$$

Now using trigonometric identities and further adjusting the equation (detailed mathematical representation can be found from referenced literature) *sin* terms in equation 13 can be written as:

$$\sin(\theta) = \frac{2\tan(\frac{\theta}{2})}{1+(\boldsymbol{\gamma})^2} \tag{22}$$

And:

$$\sin^2\frac{\theta}{2} = \frac{\tan^2\frac{\theta}{2}}{1+\tan^2\frac{\theta}{2}} = \frac{\tan^2\frac{\theta}{2}}{1+(\boldsymbol{\gamma})^2} \tag{23}$$

Therefore, in its explicit form, the transformation matrix is written as (Shabana 1998, p. 64):

$$\boldsymbol{A} = \frac{1}{1+(\boldsymbol{\gamma})^2} \times \begin{bmatrix} 1+(\gamma_1)^2-(\gamma_2)^2-(\gamma_3)^2 & 2(\gamma_1\gamma_2-\gamma_3) & 2(\gamma_1\gamma_3+\gamma_2) \\ 2(\gamma_1\gamma_2+\gamma_3) & 1-(\gamma_1)^2+(\gamma_2)^2-(\gamma_3)^2 & 2(\gamma_2\gamma_3-\gamma_1) \\ 2(\gamma_1\gamma_3+\gamma_2) & 2(\gamma_2\gamma_3+\gamma_1) & 1-(\gamma_1)^2-(\gamma_2)^2+(\gamma_3)^2 \end{bmatrix} \tag{24}$$

In equation 24 the transformation matrix is written in terms of the Rodriguez parameters γ_1, γ_2 and γ_3 (Shabana 1998, pp. 63-64).

Euler angles is the most widely used method for determining the orientation of a body. It too relies on three independent parameters, Euler angles. In the method three successive rotations are performed about three non-orthogonal axes. While there are many ways to perform the rotations, the most widely used will be introduced here. First let's consider a fixed coordinate system X_1, X_2, X_3 and a reference coordinate system ξ_1, ξ_2, ξ_3 which coincide at first. (Wittenburg 2008, pp. 9-11.) The first rotation is about X_3 axis at an angle ϕ. Since the rotation is in X_1X_2 plane, the equation for the rotation can be written as (Wittenburg 2008, p. 9):

$$\boldsymbol{\xi} = \boldsymbol{A}_1\boldsymbol{X} \tag{25}$$

In equation 25, $\boldsymbol{A}_1$ is the transformation matrix and $\boldsymbol{X}$ is the coordinate system the rotation is about. The transformation matrix $\boldsymbol{A}_1$ can be written as (Wittenburg 2008, p. 10):

$$\boldsymbol{A}_1 = \begin{bmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{26}$$

Then another coordinate system η_1, η_2, η_3 is considered to coincide with the rotated ξ_1, ξ_2, ξ_3 coordinate system. The system is then rotated about the axis ξ_1 at an angle of θ and again the rotation is in ξ_1, ξ_2 plane so the equation for the rotation becomes:

$$\boldsymbol{\eta} = \boldsymbol{A}_2\boldsymbol{\xi} \tag{27}$$

In equation 27 $\boldsymbol{A}_2$ is the transformation matrix. And it is defined as:

$$\boldsymbol{A}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix} \tag{28}$$

The last step is to consider the coordinate system $\zeta_1, \zeta_2, \zeta_3$ which is initially coincident with η_1, η_2, η_3. The final rotation is about η_3 at an angle of ψ. Rotation can be written as:

$$\boldsymbol{\zeta} = \boldsymbol{A}_3\boldsymbol{\eta} \tag{29}$$

Likewise, equation 29 conforms to equations 25 and 27. Therefore A_3 is the transformation matrix which can be written as:

$$\boldsymbol{A}_3 = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{30}$$

After forming these three transformation matrices, the final transformation between the original coordinate system X_1, X_2, X_3 and the final coordinate system can be formulated:

$$\boldsymbol{\zeta} = \boldsymbol{A}_3\boldsymbol{A}_2\boldsymbol{A}_1\boldsymbol{X} \tag{31}$$

This can be written in a more understandable form containing the ultimate transformation matrix using Euler angles:

$$\boldsymbol{X} = \boldsymbol{A}\boldsymbol{\zeta} \tag{32}$$

In equation 32 $\boldsymbol{A}$ is the transformation matrix which using Euler angles takes the ultimate form of (Wittenburg 2008, p. 10):

$$\boldsymbol{A} = \begin{bmatrix} \cos(\psi)\cos(\phi) - \cos(\theta)\sin(\phi)\sin(\psi) & \sin(\psi)\cos(\phi) + \cos(\theta)\sin(\phi)\cos(\psi) & \sin(\theta)\sin(\phi) \\ -\cos(\psi)\sin(\phi) + \cos(\theta)\cos(\phi)\sin(\psi) & -\sin(\psi)\sin(\phi) + \cos(\theta)\cos(\phi)\cos(\psi) & \sin(\theta)\cos(\phi) \\ \sin(\theta)\sin(\psi) & -\sin(\theta)\cos(\psi) & \cos(\theta) \end{bmatrix} \tag{33}$$

The three parameters ϕ,θ and ψ shown in equation 33 are called Euler angles (Wittenburg 2008, p. 10).

There are also other similar successive methods to formulate transformation matrix as Euler angles. However, the 3, 1, 3 sequence is the most commonly used. Another somewhat

common is Bryan angles, also known as Cardan angles. Compared to Euler angles it has only one significant difference, it uses a sequence of 1, 2, 3. Compared to Euler angles, Bryan angles can be linearized for small angles. (Wittenburg 2008, pp. 12-14.)

Now what method should one choose to use in a multibody dynamic analysis? Each method surely has their pros and cons. Euler angles, which is probably the most common method, and Rodriguez parameters require 3 x 3 matrix. Also, one of the main flaws these two methods have is that at certain angles singularities may occur. This happens, when two of the three rotation axes coincide and the phenomenon is called gimbal lock. This is not the case with quaternion approach or Rodriguez formula and they only use four variables to describe the orientation. (Coutinho 2013, p. 313.) However, quaternions are regarded as a more complex concept to fully grasp but in many cases, they are the ideal way to calculate rotations.

2.1.2 Kinematic constraints

A multibody system has a certain number of degrees of freedom, which are commonly referred to as DOF. A single body moving freely in space has 6 degrees of freedom, 3 for position and 3 for orientation. (Coutinho 2013 p. 5.) By first defining generalized coordinates of the body reference and referring to figure 3 they can be written as (Shabana 1998, p. 91):

$$\boldsymbol{q}_r^i = \left[R_1^i \;\; R_2^i \;\; R_3^i \;\; \boldsymbol{\theta^{iT}}\right] \tag{34}$$

In the equation 34 R_1^i, R_2^i and R_3^i define the position of the body origin in global position and $\boldsymbol{\theta^i}$ defines the rotation that one of the methods introduced in the previous chapter yielded, therefore it can be either 3 or four variables. For simplicity, generalized coordinates can be denoted by the vector $\boldsymbol{q} = [q_1\ q_2\ q_3\ \ldots\ q_n]^T$ where n is the total number of coordinates. Number of generalized coordinates in a multibody system is related to the number of constraint equations n_c and the rule $n_c \leq n$ applies. Commonly the constraint equations are written in the following form (Shabana 1998, p. 92):

$$\boldsymbol{C}(q_1\ q_2\ q_3\ \ldots\ q_n, t) = \boldsymbol{C}(q, t) = 0 \tag{35}$$

In the equation 35 $\boldsymbol{C}$ is the set of constraint equations. In this case the constraint equations are holonomic. There are two types of holonomic constraints: scleronomic and rheonomic. Scleronomic constraints don't describe the constraints explicitly as a function of time while rheonomic do. There are also non-holonomic constraint equations, which are differential constraints that cannot be integrated, for example they can contain a time derivative of a generalized coordinate (i.e. speed). (Erleben 2005, pp. 60-64, Shabana 1998, pp. 91-93.)

As stated earlier, a multibody system has a certain number of degrees of freedom. In the case of holonomic constraints, minimum number of independent coordinates needed to describe the system can be subtracted by the number of constraint equations, $n - n_c$. For example, the Peaucellier mechanism shown in figure 6 has nine links, including the two links fixed to the ground. In a two-dimensional case and using Cartesian coordinates, the mechanism has 27 coordinates to describe the configuration of each link. However, noting that there are 10 revolute joints between links and 2 fixed links, the number of degrees of freedom can be calculated to be just one (3 reduced from the number of DOFs for every fixed link and 2 for every revolute joint).

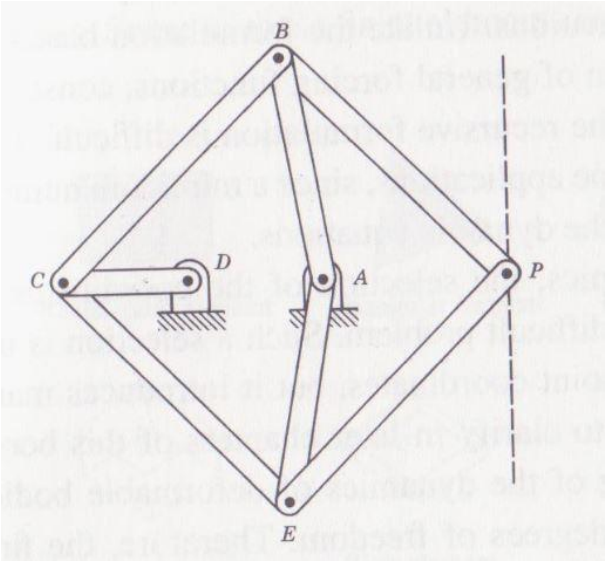

Figure 6. Peaucellier mechanism (Shabana 1998, p. 21).

A commonly used concept in constrained multibody dynamics is virtual displacement, which refers to making infinitesimal changes to some arbitrarily chosen coordinate $\boldsymbol{q}$ with respect to the constraints and forces imposed on the system. Constraint equation given by equation

35 can be reformulated using the concept of virtual displacement and Taylor's expansion as (Shabana 1998, p. 100):

$$\boldsymbol{C}_{q1}\delta q_1 + C_{q2}\delta q_2 + \cdots + C_{qn}\delta q_n = 0 \tag{36}$$

In the equation 36, C_{qi} can be written as (Simeon 2013, p. 14):

$$\boldsymbol{C_q} = \frac{\delta \boldsymbol{C}(q)}{\delta q_i} \tag{37}$$

which can be written in a matrix form as a $n_c \times n$ matrix. $\boldsymbol{C_q}$ is called the Jacobian. Jacobian is square matrix for a kinematically driven system, since $n_c = n$ (Simeon 2013, p.14, Shabana 1998, p. 100).

In the preceding chapters, many concepts were introduced which relate to kinematically driven systems. While the analysis of kinematically driven systems leads to algebraic equations which are used to find the coordinates, velocities and accelerations, it does not require any force analysis. The upcoming description of dynamic analysis does require this.

2.2 Multibody system theory – Dynamic analysis

Multibody system can be unconstrained or constrained. With regards to unconstrained motion, the most commonly used and simple form of equations of motion is the Newton-Euler equations. Selection of system coordinates, is not an issue in the case of unconstrained motion, but in the case of constrained motion, it becomes a topic of discussion. The choice matters because on the opposite ends there are a maximum set of equations in terms of redundant coordinates and minimum set of equations in terms of degrees of freedom. (Shabana 2001, pp. 188-189.)

2.2.1 Newtonian mechanics and Newton-Euler

In dynamical systems with classical mechanics, Newton's laws of motion apply. Thus, bodies have a mass and commonly there are forces and torques exerted on bodies. According to Newton's second law of motion force can be portrayed as:

$$F = ma \tag{38}$$

In the equation 38 a is the acceleration (second time derivative of position). Equation 38 applies to translational movement of a given body.

As mentioned previously, equation 38 is Newton's second law of motion in its well-known form. Newton's first law states that if there are no forces acting on a particle, the particle is at rest or in a steady motion in a straight line. Newton's third law, which is also known as the law of action and reaction states that when two particles exert forces on one another, the reaction forces are equal in magnitude and opposite in direction. The study of these laws is called Newtonian mechanics and while it serves as a basis for many theories, it alone cannot handle rigid bodies comprehensively due to rigid bodies having distributed masses (Shabana 2001, p. 10-11).

In rotational movement torque is defined as the product of inertia and rotational acceleration. Therefore, torque can be written as:

$$T = J * \frac{d^2\theta^i}{dt^2} \tag{39}$$

Torque is denoted with T, inertia J and $\frac{d^2\theta^i}{dt^2}$ is angular acceleration.

These rotational equations above are called Euler equations. In rigid body dynamics, it can be demonstrated that the unconstrained motion of a rigid body in three-dimensional space can be described using six equations, three describing the translational motion and three describing the rotational motion. Therefore, one can synthetically derive the equations of motion of an isolated body based on the afore introduced framework by Newton and Euler, this formulation is also called Newton-Euler. (Wittenburg 2008, p. 105.) Newton equations account for the translational equations and Euler equations for the rotational equations:

$$\begin{cases} m^i a^i = F^i \\ J^i \frac{d^2\theta^i}{dt^2} = T^i \end{cases} \tag{40}$$

Based on the Newton-Euler and using the principal of virtual displacement one can formulate D'Alembert's principle, which states that the inertia forces and moments acting on the body are equal to the external forces applied to the body. This makes it possible to neglect the constraint forces (Shabana 1998, p. 113):

$$\sum_{i=1}^{n_p} \left(F^i - \dot{p}^i\right) * \delta r^i = 0 \tag{41}$$

In the equation 41 the sum covers n_p number of particles, F^i is the force exerted on particle i, which consists of a sum of external and constraint forces. $\dot{p}^i$ is the rate of change of momentum of the particle and δr^i is the virtual displacement. The equation above can be simplified if the constraints are workless constraints and the equation is written in terms of a vector of generalized coordinates. After these assumptions, the simplified equation takes the form of (Shabana 1998, p. 114):

$$\sum_{j=1}^{n} \sum_{i=1}^{n_p} \left(F_e^i - \dot{p}^i\right) * \frac{\partial r^i}{\partial q_j} \partial q_j = \bar{\boldsymbol{Q}}^T \partial q = 0 \tag{42}$$

In the equation 42 F_e^i denotes the external force of particle i and term $\frac{\partial r^i}{\partial q_j} \partial q_j$ is the generalized coordinate term of the system and the $\bar{\boldsymbol{Q}}^T$ is the vector of generalized forces and the last term can also be written as the virtual work of the system (Shabana 1998, pp. 110-114).

2.2.2 Lagrangian mechanics and Lagrange multipliers

Lagrangian mechanics, which is also known as Euler-Lagrange formulation, is an excellent method for obtaining the equations of motion of a multibody dynamic system. It utilizes energy based method, which considers the kinetic and potential energy of the system. Lagrange's function is defined as (Hairer, Norsett & Wanner 1993, p. 30):

$$\boldsymbol{L = T - U} \tag{43}$$

where $\boldsymbol{T}$ is the kinetic energy of the system and $\boldsymbol{U}$ is the potential energy (Hairer, Norsett & Wanner 1993, pp. 30-31). If the generalized coordinates in the dynamic system are independent and the method of virtual work is applied, the equations of motion take the form of (Shabana 2001, p. 276):

$$\frac{d}{dt}\left(\frac{\partial \boldsymbol{T}}{\partial \dot{q}_j}\right) - \frac{\partial \boldsymbol{T}}{\partial q_j} = \boldsymbol{Q}_j \quad j = 1,2,\dots,n \tag{44}$$

In the equation 44, j denotes the system's independent coordinates or DOFs and $\boldsymbol{Q}_j$ is the generalized applied force in terms of the independent coordinate q (Wittenburg 2008, p. 90, Shabana 2001, pp. 276-277).

There are also numerous other equations of motion formulations, which are also loosely based on the concept of virtual work or the Lagrange's frame of reference. In addition to the formulations already mentioned, the most commonly known are the Gibbs-Appel and Hamiltonian formulation. These will not be further discussed due to there being more important aspects than choosing the formulation. One of the more important aspects according to Baraff (1996, p. 137): "Forward simulation is a key problem in computer graphics." He argues that constraints of a system are typically sparse and furthermore that disregarding the variety of dynamics formulations and instead focusing on the basic choice of choosing either a system description with reduced number of coordinates or a description with additional forces added to the system is essential. Reduced-coordinate description reduces the set of constraints from the number of degrees of freedom to achieve the set of reduced coordinates, commonly referred to as generalized coordinates. Finding this parameterization from maximal system to generalized system has proven to be quite expensive computationally. (Baraff 1996, p.137.)

Using Lagrange multipliers, (not to be confused with Lagrangian mechanics introduced previously) the state of the system can be expressed using the maximal coordinate representation. Constraint forces are used to enforce the constraints of the system. Even though this method of adding forces may sound worse to the uninitiated, usage of Lagrange multipliers is a very efficient method in this case since it enables the combination of an arbitrary set of constraints, which is a difficult operation if using the reduced-coordinate

approach. There are cases when the reduced-coordinate approach is preferable to the Lagrange multiplier approach and vice versa. For example, if the number of generalized coordinates left to the system is much less than the degrees of freedom removed due to the constraints, then it is justified to use the reduced-coordinates method. There are also some disadvantages to using multiplier-based methods, e.g. numerical integration errors while computing the multipliers may lead to drifting problems with constraints. This is not an issue with reduced-coordinate approach. On the other hand, Lagrange multiplier based method offers modularity for the software and there are cases when constant parameterization of a body would lead to an exponential growth in computational effort required, e.g. deformable bodies. There are also numerous stabilization methods in place to counter-act the numerical integration inaccuracies. (Baraff 1996, pp .137-139.)

Ahmed A. Shabana refers to the two methods discussed above as embedding technique and augmented formulation. Embedding technique relates to the reduced-coordinate method and augmented formulation uses Lagrange multipliers to determine the constraint forces of the system. (Shabana 2001, pp. 196-199.) As mentioned earlier the embedding technique that leads to a minimum set of coupled equations suffer from problems that are related to identifying the dependent and independent coordinates of the system. When forming the constraint Jacobian matrix of a dynamically driven system, it can be seen to $n_c \times n$ non-square matrix. With the case of linearly independent constraints, Gaussian elimination can be used to identify the nonsingular sub-Jacobian which is associated with the dependent coordinates. This is where embedding technique faces challenges as it has to find the inverse of this sub-Jacobian matrix. (Shabana 2001, pp. 322-323.)

As stated earlier, the Lagrange multiplier based method requires constraint forces to be added to the system. As in the virtual work example shown previously, the constraint forces have to be workless. The definition of workless constraint force is difficult, namely because there isn't any physical quantity to describe it. However, the constraint should be formulated so that it does not add any energy to the system, i.e. it is as "lazy" as possible. (Baraff 1996, p. 140.) Mathematically this leads to:

$$\boldsymbol{Q}_c^i = -\boldsymbol{C}_{qi}{}^T \boldsymbol{\lambda}_i \tag{45}$$

In equation 45, $\boldsymbol{Q}_C^i$ denotes the constraint force that holds the i^{th} constraint and $\boldsymbol{C}_{qi}$ is the Jacobian matrix and $\boldsymbol{\lambda}$ is called the Lagrange multiplier. The dimension of the Lagrange multiplier vector is equal to the number of constraint equations. Before examining the use of Lagrange multipliers in equations of motion, they are derived using the Newton-Euler formulation (Shabana 2001, p. 309):

$$\boldsymbol{M}^i\ddot{\boldsymbol{q}}^i = \boldsymbol{Q}_e^i + \boldsymbol{Q}_c^i \tag{46}$$

In equation 46 the assumption of reference point being defined at center of mass is made. $\boldsymbol{Q}_e^i$ denotes the external force applied to the rigid body *i.* In its explicit form, the equation can be rewritten as (Shabana 2001, p. 309):

$$\begin{bmatrix} m^i\boldsymbol{I} & 0 \\ 0 & \boldsymbol{J}^i \end{bmatrix}\begin{bmatrix} \ddot{\boldsymbol{R}}^i \\ \ddot{\boldsymbol{\theta}}^i \end{bmatrix} = \begin{bmatrix} (\boldsymbol{Q}_e^i)_R \\ (\boldsymbol{Q}_e^i)_\theta \end{bmatrix} + \begin{bmatrix} (\boldsymbol{Q}_c^i)_R \\ (\boldsymbol{Q}_c^i)_\theta \end{bmatrix} \tag{47}$$

Here mass of the body is denoted by m^i and mass moment of inertia by $\boldsymbol{J}^i$. Subscript *R* means that the element of the matrix in question is related to translational coordinate and subscript θ refers to rotational. The constraint forces that were previously formulated can be directly replaced to equation 46. Using the same notation equation 46 can be written as (Baraff 1996, p. 141):

$$\boldsymbol{M}^i\ddot{\boldsymbol{q}}^i = \boldsymbol{Q}_e^i + \boldsymbol{C}_{qi}{}^T\boldsymbol{\lambda}_i \tag{48}$$

This can be solved for acceleration $\ddot{\boldsymbol{q}}$ (for simplicity, the superscripts are left off):

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}\boldsymbol{C}_q{}^T\boldsymbol{\lambda} + \boldsymbol{M}^{-1}\boldsymbol{Q}_e \tag{49}$$

Differentiating constraint equation displayed in equation 35 with respect to time once and twice leads to equations (Shabana 2001, p. 336):

$$\begin{cases} \boldsymbol{C}_q\dot{\boldsymbol{q}} = -\boldsymbol{C}_t \\ \boldsymbol{C}_q\ddot{\boldsymbol{q}} = \boldsymbol{Q}_d \end{cases} \tag{50}$$

In equation 50 the subscript *t* denotes the differentiation with respect to time and $\boldsymbol{Q}_d$ is the combined term that results from two successive partial differentiations. The second partial differentiation can also be displayed as:

$$\boldsymbol{C}_q\ddot{\boldsymbol{q}} + \boldsymbol{Q}_d = 0 \tag{51}$$

The sign of Q_d can be ignored since the term bears no physical meaning as is. Equations 49 and 51 can now be combined to form:

$$\boldsymbol{C}_q\left(\boldsymbol{M}^{-1}\boldsymbol{C}_q{}^T\boldsymbol{\lambda} + \boldsymbol{M}^{-1}\boldsymbol{Q}_e\right) + \boldsymbol{Q}_d = 0 \tag{52}$$

Equation 52 can now be formulated in a way to solve for the Lagrange multipliers, which is of the form:

$$A\boldsymbol{\lambda} = \boldsymbol{b} \tag{53}$$

Based on equation 52, the A and **b** are:

$$\begin{cases} A = \boldsymbol{C}_q\boldsymbol{M}^{-1}\boldsymbol{J}^T \\ \boldsymbol{b} = -(\boldsymbol{C}_q\boldsymbol{M}^{-1}\boldsymbol{Q}_e + \boldsymbol{Q}_d) \end{cases} \tag{54}$$

This formulation is referred to as $\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^T$ by Baraff. It is quite commonly used to compute the **λ**-value. It has quite a few advantages, for example if *A* isn't too large and nonsingular, Cholesky decomposition can be used to for calculating **λ**. As *A* grows, iterative methods start to outperform direct methods. These iterative methods expect *A* to be a sparse matrix. For example, in figure 7a there is a serial chain that can easily be solved using a banded solution method like banded Cholesky decomposition. Using the big O notation, time it takes to solve such a system takes *O(n)* time, meaning that the time it takes to reach the solution depends linearly on the size of the system. However, in figure 7b the links are all connected to the one link to form a branching structure. While the problem seems trivial and very much similar to the 7a, this is not true. This is due to the matrix *A* not being a sparse but being dense. This prohibits the use of $\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^T$ method. (Baraff 1996, pp. 140-142.)

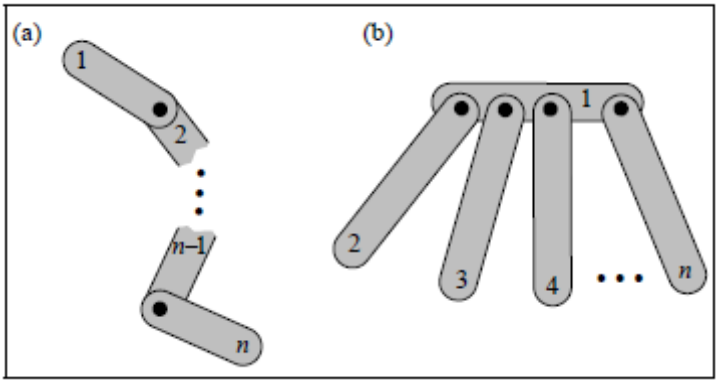

Figure 7. (a) Serial chain, (b) Branched structure (Baraff 1996, p. 141).

Using equations 48 and 50 and combining them to one matrix equation as (Shabana 2001, p. 336):

$$\begin{bmatrix} \boldsymbol{M} & \boldsymbol{C}_q^T \\ \boldsymbol{C}_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q}_e \\ \boldsymbol{Q}_d \end{bmatrix} \tag{55}$$

Accelerations and constraint forces can be solved from the equation above by solving:

$$\begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M} & \boldsymbol{C}_q^T \\ \boldsymbol{C}_q & 0 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{Q}_e \\ \boldsymbol{Q}_d \end{bmatrix} \tag{56}$$

Denoting the matrix containing mass and Jacobian as $\boldsymbol{H}$ it can be written as:

$$\begin{bmatrix} \boldsymbol{M} & \boldsymbol{C}_q^T \\ \boldsymbol{C}_q & 0 \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{H}_{qq} & \boldsymbol{H}_{q\lambda} \\ \boldsymbol{H}_{\lambda q} & \boldsymbol{H}_{\lambda\lambda} \end{bmatrix} = \boldsymbol{H} \tag{57}$$

In equation 57 the values for each individual item in the $\boldsymbol{H}$ matrix are:

$$\begin{cases} \boldsymbol{H}_{\lambda\lambda} = \left(\boldsymbol{C}_q \boldsymbol{M}^{-1} \boldsymbol{C}_q{}^T\right)^{-1} \\ \boldsymbol{H}_{qq} = \boldsymbol{M}^{-1} + \boldsymbol{M}^{-1} \boldsymbol{C}_q^T \boldsymbol{H}_{\lambda\lambda} \boldsymbol{C}_q \boldsymbol{M}^{-1} \\ \boldsymbol{H}_{q\lambda} = \boldsymbol{H}_{\lambda q}{}^T = -\boldsymbol{M}^{-1} \boldsymbol{C}_q^T \boldsymbol{H}_{\lambda\lambda} \end{cases} \tag{58}$$

Substituting equation 57 into 56 and multiplying the matrix with the force vector, obtained equation is:

$$\begin{cases} \ddot{\boldsymbol{q}} = \boldsymbol{H}_{qq}\boldsymbol{Q}_e + \boldsymbol{H}_{q\lambda}\boldsymbol{Q}_d \\ \boldsymbol{\lambda} = \boldsymbol{H}_{\lambda q}\boldsymbol{Q}_e + \boldsymbol{H}_{\lambda\lambda}\boldsymbol{Q}_d \end{cases} \tag{59}$$

These equations can be solved for the constraint forces and accelerations using direct integration methods. Also, $\boldsymbol{H}$ matrix can be solved in *O(n)* time using sparse matrix solution methods, e.g. factorizing $\boldsymbol{H} = LDL^T$ can be used. (Baraff 1996, p. 142.)

The drawbacks of using acceleration-based method described above are the incapability to handle collisions and many times they are subject to inconsistency and indeterminacy. The drawbacks stem from the transformation from Differential Algebraic Equations (DAEs) to Ordinary Differential Equations (ODEs). DAEs are transformed to ODEs because the methods of solving ODEs are quite simple and well understood. The problems arise when the constraint equations are differentiated. This method does yield the required ODEs, but the newly found state variables, generalized coordinates and velocity, do not completely satisfy the constraint equations because of numerical error in the integration. To overcome these constraint violation and drifting issues methods, such as the Baumgarte's stabilization, are used. Baumgarte's constraint violation stabilization method aims to stabilize the violations by adding additional terms in the acceleration constraint equations. There is, however, the possibility for this method to affect the dynamic equations since it lacks positive error control, which may have detrimental effects to the simulation. Similar method as Baumgarte's one is based on the use of penalty forces that considers the numerical error. (Erleben 2005, p. 46, Yu & Chen 2000, p. 52.)

Another common method to stabilize constraints is based on stabilization by projection. This method is usable with practically any integration method and the way it works is by utilizing Lagrange multipliers to solve a minimization problem that arises after each time step when constraints drift. The problem with iterative method is that it calls for solving multiple numerical integration steps, such as Newton's method, to reduce error in the constraints. However, this is not optimal in real-time applications and therefore the number of steps has to be reduced. Thus, real-time applications can apply a non-iterative projection method,

which limits the error so that it will not accumulate over time. (Burgermeister, Arnold & Esterl 2006, pp. 767-771.)

2.3 Physics engines' core components

In this chapter, the main physics engine characteristics and properties are introduced. These properties and their effect to the output variables of the simulation, such as speed, robustness, stability and accuracy, are discussed. Then the afore introduced physics engines are examined based on these properties and values after which a conclusion can be formed on what sort of tasks these engines are most suited for. Due to the nature of closed source programs, some features are not up for reviewing as they have not been disclosed to the public. This is the case with MeVEA.

The following study is split into five main categories that are used to define the properties of physics engines (Boeing & Bräunl 2007, p. 281):

- Integrator
- Simulator paradigm
- Object representation, collision detection and contact determination
- Material and friction properties
- Constraint implementation

2.3.1 Integrator

Integrator or numerical time integration method plays a key role when determining the accuracy and the speed of the simulation. Euler's method is probably the simplest numerical time integration method. Explicit Euler's method is formulated as (Boeing & Bräunl 2007, p. 282):

$$x(t_0 + h) = x_0 + h * \dot{x}(t_0) \tag{60}$$

In equation 60, *x* denotes the function to be numerically integrated which is a function of time. t_0 is the initial time and *h* being the size of the step. First order Euler's method in simulation has proven to be quite inaccurate depending on the problem (Hairer, Norsett & Wanner 1993, p. 132), therefore it is not used as is. Instead most physics engines employ a symplectic Euler integrator, also known as semi-implicit Euler, which is somewhat similar

to the explicit Euler integrator with the exception of first using the updated velocity and then calculating the position. Thus, symplectic Euler integrator is formulated as (Boeing & Bräunl 2007, p. 282):

$$\begin{cases} \dot{x}(t_0 + h) = \dot{x}_0 + h * \ddot{x}(t_0) \\ x(t_0 + h) = x_0 + h * h\dot{x}(t_0) \end{cases} \quad (61)$$

Equation 61 therefore has one additional clause. Accents indicate the differentiation with respect to time.

Other commonly used time integration method is the Runge-Kutta method. This method utilizes Euler's method too. Method for integrating using explicit Runge-Kutta goes as follows (Haataja et al. 1999, p. 197):

1. First an Euler step from point (t_k, x_k) is taken with the length of $c_2 h$.
2. Derivative in the afore calculated point is noted as k_2.
3. Then one must return to the starting point and take another Euler step with the length of $c_3 h$.
4. Using the weighted mean of k_1 and k_2, one gets k_3. k_1 is the function value at (t_k, x_k)
5. Using this weighted mean value, the final Euler step is taken from point (t_k, x_k) to point $(t_k + 1, x_k + 1)$. Runge-Kutta method can be understood to probe the value of the derivative before taking the final step.

This is the second order explicit Runge-Kutta, however the most commonly used Runge-Kutta method is the fourth order Runge-Kutta. This method has four *k* values before determining the weighed mean value k_5. The method is the most popular because it yields the best results while at the same time not requiring any additional derivatives. (Haataja et al. 1999, p. 198.) In the table 1 the order of the Runge-Kutta is compared with the number of operations required to solve the equation. Although it is worth noting that this does not generally apply to differential equation systems, but is just one reason why the 4th Runge-Kutta is the most commonly used.

Table 1. Operations per order of the method.

Order	1	2	3	4	5	6	7	8
Operations	1	2	3	4	6	7	9	11

It is worth noting that for stiff problems explicit Runge-Kutta is not feasible but implicit Runge-Kutta must be utilized. Implicit Runge-Kutta can also be shown to have an order, and therefore accuracy, of *2s* when the number of operations is *s*. Though this does demand that nonlinear equations are solved at each step, which requires a lot of computational effort. (Haataja et al. 1999, p. 199.)

2.3.2 Simulator paradigm

The simulator paradigm of a physics engine affects the accuracy of different aspects. These paradigms are commonly divided to three groups: penalty-based method, constraint-based method and impulse-based method. These paradigms relate to how contact is detected and handled and therefore have a significant impact on computational effort required as well as achieved accuracy. Constraint-based method is regarded as a rather complex paradigm and it usually deals with differential algebraic equations. Complementary form of the equations of motion is quite commonly formulated with constraint-based methods. Many techniques have been developed for solving these Linear Complementarity Problems (LCP) that arise when using constraint-based methods. Penalty-based method is a simpler method, however it has quite a few drawbacks. Penalty-method allows for penetration on contact, which is then used to calculate the penalty forces, however fixed time-stepping introduces numerical instability and erroneous penetrations if the step size is not small enough. It is quite difficult, even impossible, to accurately model two infinitely rigid objects penetrating one another with a finite outcome. As a result, very small timesteps are require. Impulse-based method simulates contacts as impulses between the objects. The advantage is computational efficiency in a case with multiple moving objects. The drawback is that static object representation is lacking. Contact for two static objects is described using high-frequency micro-impulses. (Erleben 2005, pp. 20-21, Baraff 1989, p. 224.)

Recently the uses of recursive algorithms have also gained traction. Recursion is based on a method to solve forward dynamics problem by Featherstone and they can be realized to be

tree-like systems such as a humanoid model. The method is also referred to as reduced coordinate method. One proposed method to handle recursive algorithm is to first introduce open-loop recursive differential equations and then turning it to closed-loop using position penalty terms. While they excel in branched structures, they struggle to perform in an environment of multiple disjointed bodies. (Callejo et al. 2014, p. 36, Erleben 2004, p. 69.)

2.3.3 Object representation

Main task of object representation is to help detect collision and contact. While it might seem trivial to detect contact between two objects, discrete time integration and contact point determination raise a few issues. Is the contact determined in the future, past or present? Furthermore, can the objects penetrate or are the rigid contact conditions strictly enforced? Choosing the correct simulator paradigm can answer these questions, but there is more to object representation than that.

Commonly objects in multibody environment have a certain volume surrounding them, which is used to help determine the contact. This is because some objects are quite complex in shape and it is beneficial to simplify the contact determination with a simpler bounding volume. Most commonly used ones are axis-independent volumes (e.g. spheres), axis-aligned volumes and object-oriented volumes. Difference between the last two is that axis-aligned is in global coordinates while the object-oriented is in object's local coordinates. It is noteworthy that usually contact detection hierarchical procedure; when one contact is detected in the bounding volumes, the procedure moves to more detailed bounding volume. (Volino 2004, pp. 22-23.)

2.3.4 Material properties

Material properties define which physical models can be simulated. Friction model, for example, can be categorized under material properties. Friction model is usually according to Coulomb friction, but approximations are quite common as well. Usually friction is formulated as an LCP problem. Historically significant framework was done by, Cottle and Dantzig, who mathematically formulated principal pivoting method to solve LCP. Their algorithm is known today as Dantzig's algorithm. Carlton Lemke suggested a different method to solve the linear complementarity problem more applicable to rigid-body contact

simulation, which is introduced in detail in Cottle, Pang & Stone 1992. (Cottle & Dantzig 1968, Cottle, Pang & Stone 1992.)

2.3.5 Constraint implementation

How many different constraints is the engine able to simulate? Most common constraints are universal, spherical, revolute, prismatic, fixed and generic constraint but there are also some more complex constraints such as corkscrew and distance or non-holonomic constraints. Constraint implementation should also resolve the problems with constraint instability, which is especially troublesome with constraint-based methods.

2.4 Open Dynamics Engine

As stated before ODE is a physics engine. As such it is used in various application, for example in video games or robotic simulators etc. Original developer Russell Smith is no longer leading the developing process, but it still has active developers working on it. The interface is in C even though the engine is mostly written in C++.

2.4.1 ODE integrator

ODE has been designed to be fast, robust and stable. Therefore, the integrator had to be carefully chosen as to keep the analysis times as low as possible and in the same time not compromising the stability. Fast, robust and stable does not however translate to 'accurate' as ODE has been mainly designed to be used in games and other virtual reality environments. It is not a recommended tool for quantitative engineering due to its inaccuracy, which is a result of it having only a first order integrator. (Smith 2006, pp. 1-2.)

2.4.2 Simulator paradigm

Simulation method in ODE is based on the impulse-based model by Trinkle & Stewart and Anitescu & Potra. Stewart & Trinkle suggested use of an impulse-based method. The main advantages related to the method are that all contact forces are treated equally (whether finite or instantaneously infinite), which results in that there is no need to determine the time of impact explicitly. Whereas in acceleration based methods when contacts form and break, they require a switch between impulse-momentum and force-acceleration methods. Moreover the method by Stewart & Trinkle does not require the step size to be too small. Anitescu & Potra extended on the work of Stewart & Trinkle and introduced a method for

time-stepping with acceleration-level LCP solver while maintaining velocities and impulses as unknowns. This method guarantees a solution regardless of the number of contacts or configuration of the system (Stewart & Trinkle 1996, Anitescu & Potra 1997).

The issue with contact is that it requires the engine to solve a Linear Complementarity Problem. There are two algorithms in ODE that solve the LCP. Dantzig's algorithm, which is an extension of the commonly used Lemke's algorithm, is used in the *dWorldStep()* function which tries to yield a numerically exact solution. Often it is not the fastest method but it yields more accurate results. The method itself uses a LDL^T factorization to solve the constraints. There is also an iterative algorithm that is called *dWorldQuickStep()* in ODE API which utilizes a parameter called Successive Over-Relaxation (SOR) and the LCP solver is called Projected Gauss-Seidel. However, this iterative algorithm is not recommended to structures that are near-singular due to poor accuracy. Legged robot that sits on the ground can be near-singular. Also, high-friction contacts cause problems. (Smith 2006, p. 17.)

2.4.3 Object representation and collision

Open Dynamics Engine has its own collision detection engine, but it also a couple of libraries that are used to detect collisions. These libraries are OPCODE (acronym for Optimized Collision Detection) and GIMPACT. OPCODE uses Axis-Aligned Bounding Box trees in its current implementation (version 1.3).

Collision handling in ODE works in following order:

1. Call to collision detection is made to determine what each body is touching. The functions that handle this return a list of points of contact. Information about position of these points, surface normal vector and depth of penetration are contained within this list.
2. Contact joints are created in these contact points. There are multiple properties that can be defined for the contact joints, such as friction coefficient for rolling, spinning etc. or bounciness, constraint force mixing parameter etc.
3. Contact joint group is created with the purpose of easy addition of joints and clean-up.
4. A simulation step is performed next.

5. After stepping the simulation, the contact joint group is cleaned-up.

(Smith 2006, p. 10.)

2.4.4 Material properties

ODE uses an approximated Coulomb friction model. Coulomb friction is simply described as being a relation between tangential and normal forces acting on a contact point. This can be written in an equation form as (Smith 2006, p. 11):

$$|f_T| \leq \mu * |f_N| \tag{62}$$

In equation 62 f_T denotes tangential force acting on the contact point and f_N the normal force accordingly. Friction coefficient is described by μ. This equation is usually depicted as a friction cone although ODE uses a friction pyramid shaped approximation to achieve better efficiency. ODE approximates friction coefficient by taking the maximum μ value which ever tangential direction it is. (Smith 2006, p. 11.) Friction model is based on the work of David Baraff, who extended on the work of Cottle and Dantzig and Lemke. The model by Baraff includes both static and dynamic friction and a model of Coulomb friction, which as stated earlier, is approximated in ODE. (Baraff 1994, p. 23.)

2.4.5 Constraint implementation

ODE is capable of simulating spherical, hinge, sliding, contact, universal, double hinge, fixed and angular motor joint. In addition, several joint limits can be applied to each joint, such as friction and restitution coefficients, angle and angular speed limitations, maximum forces etc.

As was described previously in chapter 2.2.2, constraint-based methods suffer from joint instability, i.e. drifting. In ODE, these can happen because the orientation of bodies was not set correctly or simply because numerical approximations will yield these cumulating drifting issues during the simulation. In figure 8 this constraint instability is demonstrated as the ball has drifted away from the socket.

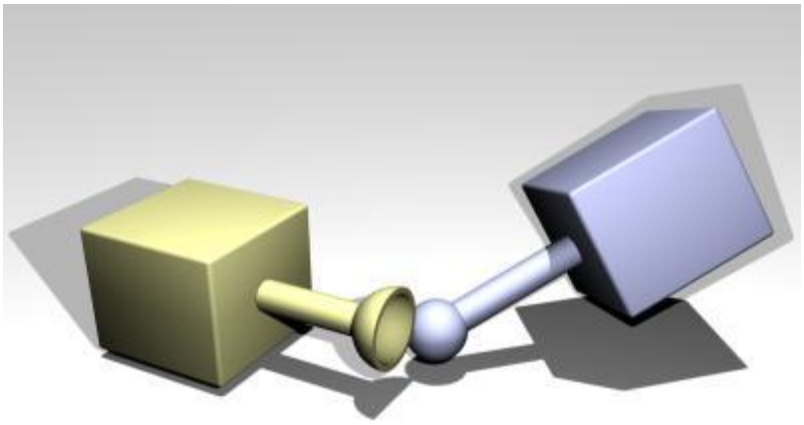

Figure 8. Constraint instability issues in a ball and socket joint (Smith 2006, p. 8).

ODE tackles the aforementioned problem using a special Error Reduction Parameter (ERP). This parameter creates an additional force during each simulation step to bring the jointed bodies back accordingly. Value of this is between 0 and 1, 0 means there is no force added during each step, while 1 means that complete joint correction is attempted. However, neither of the foregoing values are recommended as ERP = 0 would lead to drift and ERP = 1 cannot fully remove the error due to internal approximations. Values between 0.1 and 0.8 are recommended. One of the more interesting features of ODE is called Constraint Force Mixing (CFM). CFM parameter is implemented directly to constraint equation and what it does is it can set a "hard" contact (no penetration allowed) to a non-hard contact. CFM value of zero means the constraint is strictly enforced (hard), while the positive value will allow for some violation. (Smith 2006, pp. 7-8.) Together CFM and ERP can be used to simulate spring and damper and even more importantly, simulating spring and damper this way results in not needing to explicitly simulate them, rather implicitly obtaining them as a part of LCP system (Hsu & Peters 2014, p. 41).

Although ODE has not become the de facto standard in robotics simulation, there have been some contributions that address the issues that arise in this specific field when using ODE. Drumwright et al. introduces the following issues when using ODE to simulate mobile robot locomotion:

1. ODE requires computational power substantially, which makes real-time simulation quite slow to perform. This is a problem especially relevant to this thesis.
2. Joint-dampening is far from ideal model. Thus simulated robots cannot perform certain types of tasks. ODE is not the tool to perform qualitative research.
3. Friction cone is an approximation of the true Coulomb friction cone.
4. ODE applies a LCP solver to enforce constraints and ensure forces satisfy non-interpenetration. Yet this solver doesn't always find a solution which causes instable and non-physical behavior.

Suggested solutions to the aforementioned issues include parallelization of the ODE's quickstep algorithm that was discussed briefly. Parallelization was based on turning the Gauss-Seidel algorithm subsets into Jacobi iterations. System can be stabilized using CFM parameters. Non-iterative algorithms could provide even better performance. (Drumwright et al. 2010, pp. 39-43.)

2.5 Project Chrono

According to their website: "Chrono is a physics-based modelling and simulation infrastructure based on a platform-independent open-source design implemented in C++." It is a software development kit that includes several additional modules. Applications include robotics, vehicle dynamics, finite element analysis, virtual reality, collision detection etc. It is actively developed in the University of Wisconsin-Madison. Chrono features modules for interoperation with CAD files (CASCADE module), cosimulation using TCP/IP socket system (COSIMULATION module), FEA, runtime visualization using Irrlicht library (IRRLICH module), interoperation with Matlab (MATLAB module), Intel MKL library direct solver (MKL module), parallel computation using OpenMP, CUDA or Thrust (PARALLEL module), postprocessing for high quality rendering (POSTPROCESS module), Python parser (PYTHON module), vehicle modelling (VEHICLE module), OpenGL API and at the heart of Chrono is the Chrono::Engine, which includes the core functionalities. In the figure 9, components which are included in the Chrono::Engine are shown. (Project Chrono – An Open Source Physics Engine 2016, Brief Introduction 2016.)

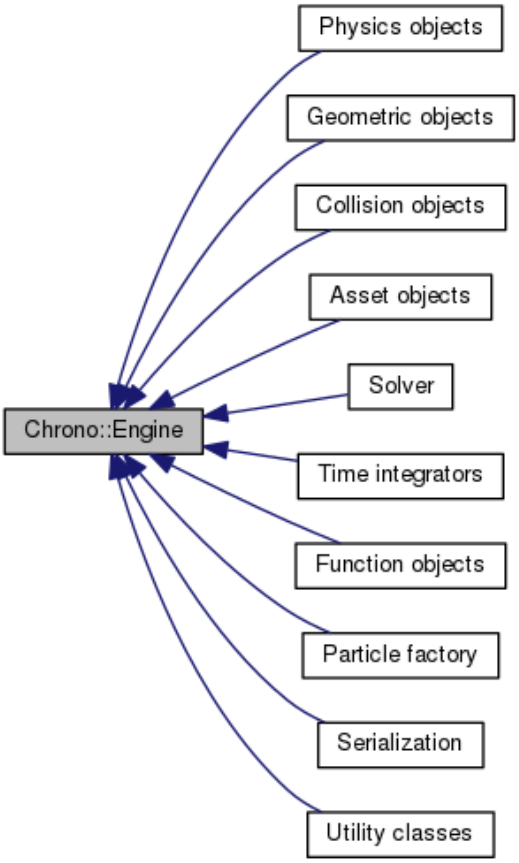

Figure 9. Chrono::Engine collaboration diagram (Chrono::Engine 2016).

Installation process for Chrono is somewhat similar as that of other physics engines described earlier. As with open-source software, it is recommended to use the Git-client to clone the Git repository. Other libraries have to be downloaded as well, as they are not included in the Chrono source files. For example, if one wishes to simulate a rigid body and visualize the simulation as well as use parallel computing to utilize GPU processing power, Irrlicht, Blaze, Boost and Thrust libraries need to be downloaded from their respective sites (or repositories). The three latter ones are related to the parallel computing module. The benefits of modularity are that firstly module dependent software are not required but can be used if the user so desires, secondly a project made of smaller components avoids producing a huge dynamic link library file (.dll), and lastly in the future more modules can be developed without interfering with the core components of Chrono. Figure 10 illustrates the dependencies of different components and modules of the Chrono and project environment.

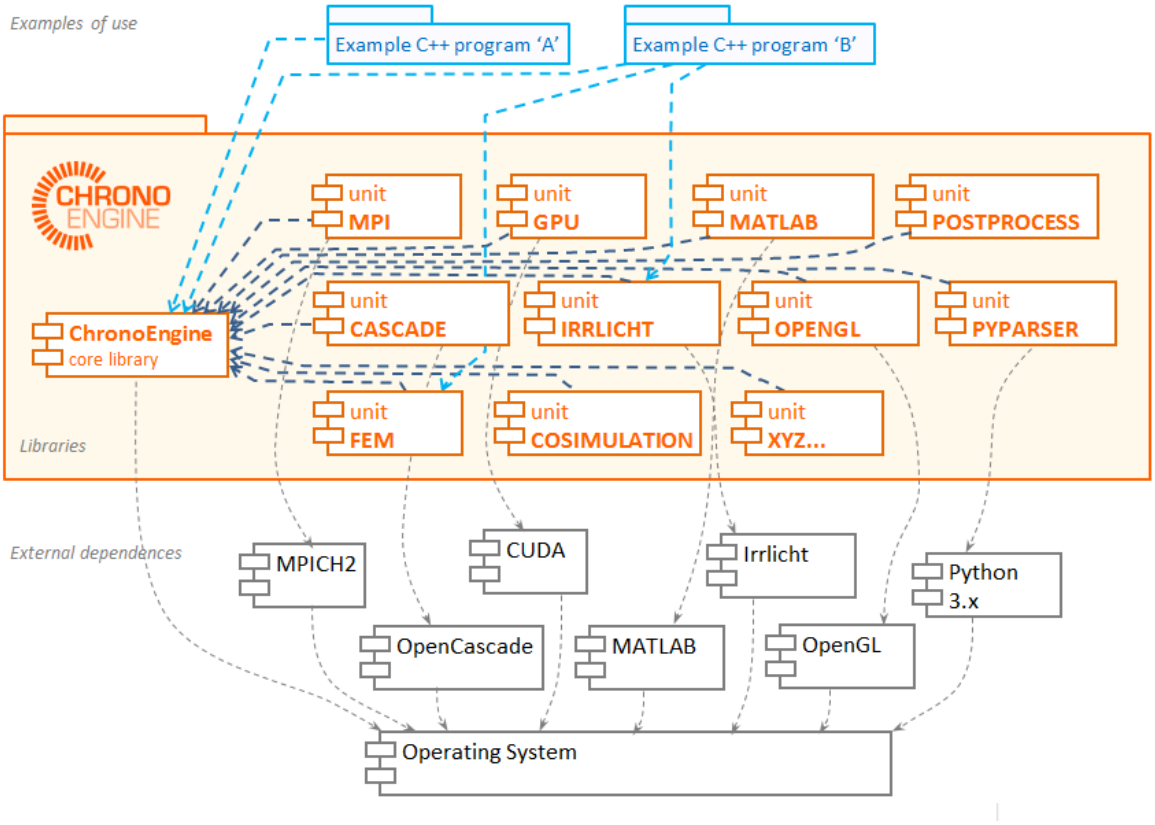

Figure 10. Relations between Chrono::Engine, its modules and external dependencies (Introduction to modules 2016).

Even though in terms of speed Chrono is not considered a contender for ODE, the most interesting feature with Chrono is the parallel computing module. Even though at the time of this thesis the module seems to be a work-in-progress, it seems promising and may prove to boost performance by including the GPU in the simulation process. Chrono does trade generality to speed, hence it may exclude some features from the standard Chrono::Engine.

2.5.1 Chrono integrator

Chrono has three main time stepping methods, like other introduced physics engines Chrono has a

- Symplectic first-order Euler integrator
    - Symplectic Euler is very useful due to its capability to solve problems of non-smooth dynamics (so-called Differential Variational Inequalities, DVI). Non-smooth problems arise when hard contacts are introduced to the system. DVI timestepper in Chrono API is called *ChTimestepperEulerImplicitLinearized.* (Tasora 2016, pp. 1-2.) There is also the possibility to use implicit Euler

integrator by calling *ChTimestepperEulerImplicit* function. Fast but offers first-order accuracy. Constraint stabilization is utilized. Chrono also includes a projected method of integration, which is called using *ChTimestepperEulerImplicitProjected.*

- Newmark integrator
    - Widely used within the finite element method community. Second-order integrator that delivers first-order accuracy (Tasora 2016, pp. 34-35).
- HHT integrator
    - Implicit integrator that is based on the formula by Hilber, Hughes & Taylor. Called with *chrono::ChTimestepperHHT*. It is commonly slower than symplectic Euler, but offers second-order accuracy when used in FEA. No need for constraint stabilization due to inner iterations keeping the constraints exact. Does not support hard contacts. (Tasora 2016, pp. 35-37).

In addition to these integrators, many more can be found from Chrono, but they seem to be under development and may not be stable. For example, Runge-Kutta 4 can be found as well as explicit Euler integrator.

2.5.2 Simulator paradigm

Chrono has multiple solvers, but there is limited amount of information regarding these solvers. Recommended solvers according are:

- SOLVER_SOR
    - Low precision but for small problems provides fast execution. Supports hard contacts.
- SOLVER_APGD
    - High accuracy and very good convergence. Supports hard contacts.
- SOLVER_BARZILAIBORWEIN
    - Good convergence and support for hard contacts. Similar to previous solver but may be more robust with large mass ratios
- SOLVER_MINRES
    - FEA solver with good convergence. No support for hard contacts.

(Simulation system 2016.)

Quaternion algebra forms the basis for descriptions of positions, speeds and accelerations. Quaternions are used in coordinate descriptions as well.

2.5.3 Material properties

Frictional contact description can be chosen from complementarity based approach or penalty based approach. Chrono includes Coulomb friction model to portray stick-slip phenomena. To add to that, rolling and spinning friction are also supported. Non-linear springs and dampers can be added.

2.5.4 Object representation and collision

Chrono supports compounds of cubes, spheres, triangle meshes, convex geometries etc. There are two main methods available to detect collision: Bullet collision detection engine, which is included in Chrono::Engine, or Chrono's own collision detection, which is comprised of three phases: broad phase, narrow phase and detail phase collision detection. Broad phase collision detection utilizes a method called sweep-and-prune SAT, the narrow phase uses AABB and/or OBB volume trees and the detail phase uses customized primitive-to-primitive fallbacks. Upon contact, relevant data, such as penetration depth and distance, are measured. Objects can be disabled so that they are not considered by collision detection, this way processing power is not allocated to objects that are nowhere near colliding with another object. (Brief Introduction 2016.)

2.5.5 Constraint implementation

Chrono's implementation of stabilization differs from ODE. Euler implicit projected time stepper utilizes projection, which formulates a speed problem followed by a position problem, thus keeping constraints from drifting. Euler implicit linearized resembles formulation by Anitescu/Stewart/Trinkle, but does not include similar ERP as ODE. (Simulation system 2016.)

3 RESEARCH METHODS

Research methods used in this thesis are literature review and computer simulation. The goal of the literature review was to provide necessary theory behind multibody dynamics as well as review recent studies related to the subject. Computer simulation was performed using different physics engines to see how their simulation times differ from one another. To achieve applicable results, the simulation model and basic paradigms had to be equal.

3.1 Literature review

The basic theory behind multibody dynamics was dictated by textbooks by Ahmed A. Shabana (Shabana 1998, Shabana 2001). This is due to not to confuse the reader by the multitude of different symbols for the same quantities. Literature regarding recent developments taken from peer-reviewed sources, mainly from Scopus database. Regarding the timeline of the theory presented in the literature, it is quite vast. Some of the principal theories were presented during 1700s. Even though the field of multibody dynamics is heavily researched, some of those topics of research have not yet found their way into the physics engines used for modeling, which is the main topic in this thesis. Moreover, the emphasis in multibody dynamic research has shifted from rigid materials towards flexible materials. Most new publications do not offer any new formulations but instead mix and match currently used ones or focus on improving computational efficiency through optimizing software's hardware usage. Finite element analysis of structures is a topic of heavy research.

3.2 Computer simulation

Computer simulation was performed on several physics engines listed in the beginning of section 2. The purpose of this was to achieve comparable results with one another. Therefore, it was crucial to perform the analyses on the same computer. The specifications of the used computer are displayed in the table 2 below.

Table 2. Specifications of the used computer.

Operating System	Windows 7 Enterprise 64-bit
Processor	Intel Core i7-3770 @ 3.40 GHz (8 CPUs)
Memory	16 GB of RAM
Graphical processing unit	NVIDIA Quadro 600, 4095 MB

There are also various methods to increase the speed of simulation, such as parallel computing, but these were disregarded as they would not shed light as to what physics engine performs fastest. Also, there is still quite a lot of research to be done about parallel computing. The model used in the benchmark is a simple slider-crank model, which is analyzed with and without friction. The case with no friction will therefore include a slider constraint while the contact condition is neglected. The source codes for each tested physics engine will be provided in the appendices.

The basic components of the two main physics engines, ODE and Chrono, were described earlier but the workflow when working with these software development kits was neglected. Latest version of ODE can be found from Bitbucket, which is a site for repositories. It is held there as it includes version control and therefore can be utilized by the developing community to suggest changes to the software. An older version of ODE source code can be found from its website. Once the source files have been downloaded (either directly from the website or via version software such as Mercurial or GIT) the installing can be done in several ways. Project files can be generated using premake4.exe, which is included in the source code. This is done by opening a command window in the location of the ODE "build" folder and running the command "premake4.exe" and including appropriate options. These options can be e.g. "--with-tests", which would include test project files or "--with-demos", which would include demo files. Using this configuration, the user would also need to clarify the Visual Studio version to be used in the command line prompt. Regardless of the installed Visual Studio version, it is recommended, even in the latest releases, to use "vs2008" option, although the "vs2010" option seems stable too. premake4 does not include anything later than vs2010. There are also other options that can be clarified, but in this thesis the command line command was "premake4.exe -vs2008". The project files are then created to the "build" folder with the options that were used in the command line prompt. Using the aforementioned command, only Microsoft Visual Studio Solution file and VC++ Project file

were created. When opening the solution file with Visual Studio, the files need to be migrated/upgraded to the version of Visual Studio running on the computer.

Working with ODE requires an Integrated Development Environment and compiler, hence Visual Studio is used in ODE project development. Thus, there is no built-in Graphical User Interface (GUI) in ODE and the development process is restricted to high-level programming language, which is C in the case of ODE. C is used rather than C++ due to C being easier and using C++ might lead to problems across multiple compilers. Compiling can be done in "debug" or "release" mode. While debug mode is slower, it ensures internal consistency by checking run-time arguments and performing run-time tests, release mode does not include these.

Typical simulation code in ODE resembles the following order:

1. Creating the dynamic world
2. Creating bodies in the world
3. Setting the state of the bodies (position, rotation)
4. Creating joints between bodies and/or the world and setting their parameters
5. Creating a collision world and collision objects
6. Creating a joint group to hold all the contact joints
7. Starting the simulation loop:
    a. Applying forces to bodies
    b. Adjusting the joint parameters
    c. Calling the collision detection module
    d. Creating a contact joint for every detected contact point
    e. Taking a simulation step
    f. Removing all joints in the joint group
8. Destroying the dynamic and collision world

(Smith 2006, pp. 10-11.)

Simulation loop is performed as many times as the user defines. As the coding is done using C, the code is partitioned accordingly. Moreover, the ODE comes with "drawstuff"-library, which is used to graphically demonstrate the simulation using Open Graphics Library (OpenGL) API. It is used in the demos. However, it is not used in the benchmarking since it will loop as real-time (instead of as fast as it could).

Chrono::Engine workflow is quite similar to the one in ODE, but overall it can be seen to be considerably larger due to the number of modules that can be included in a project. In the figure 11 a workflow is demonstrated that is certainly not applicable to every project but serves more as a demonstration of available features.

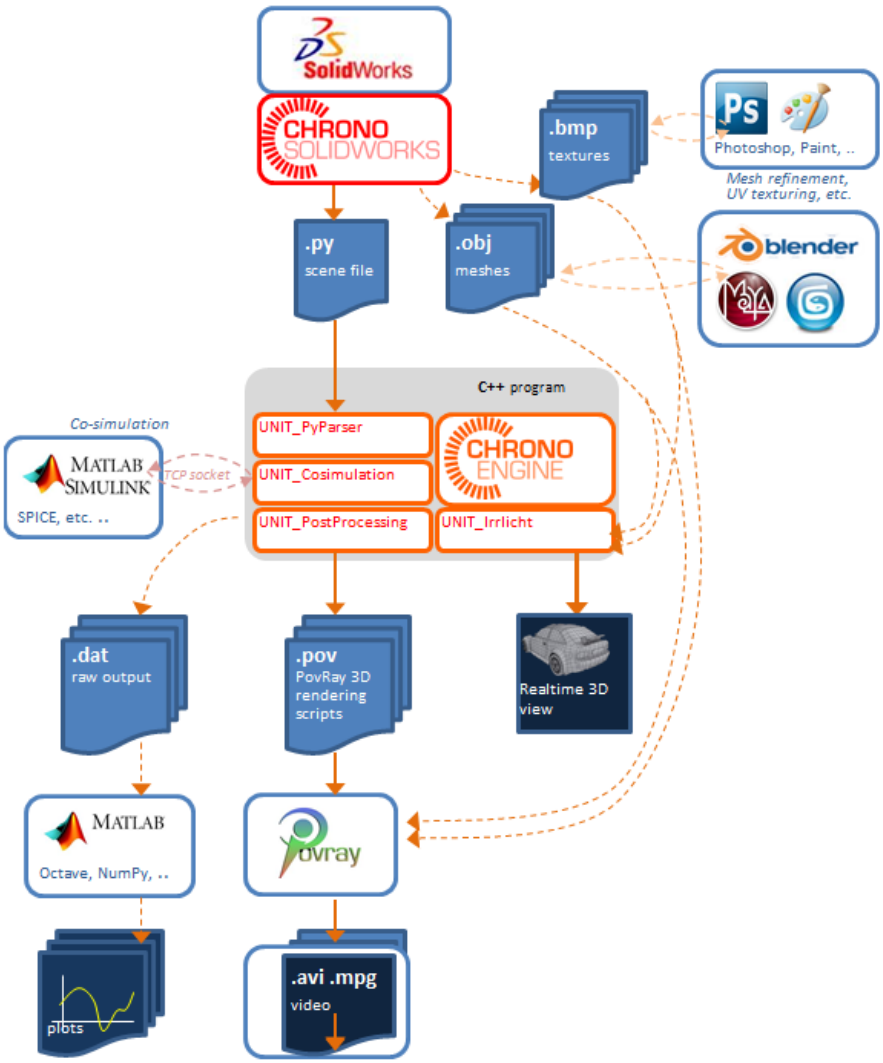

Figure 11. One workflow possibility in Chrono::Engine (Chrono Frequently Asked Questions 2016).

Although there is a rather substantial number of modules included in the figure above, there is no need to utilize them in this thesis. Only the Chrono::Engine component with Irrlicht is required to run the simulations and verify them.

MeVEA workflow follows an easier route, as it contains a graphical interface for easier model verification. MeVEA's simulation platform itself consists of two parts, MeVEA Modeller, which is used to create and modify the model, and MeVEA Solver, which solves the multibody dynamics simulation. MeVEA also contains external interface with Matlab/Simulink, which is also compatible with C/C++, thus being quite modular.

In this thesis, the simulation model for benchmarking was a 3-dimensional slider-crank mechanism shown in figure 12. The mechanism consists of three bodies: a crank, connecting rod and slider. The crank is connected to ground via hinge joint at point A and connected to connecting rod via spherical joint at point B. Slider is connected to ground via sliding joint, also known as prismatic joint, and there is a universal joint between connecting rod and slider at point C in such a way that the connecting rod can only rotate around body coordinates x and y shown in figure 12. The total number of degrees of freedom is therefore 6*3 – 5 - 3 - 4 – 5 = 1. A torque is applied to crank at hinge location A, which will lead to slider moving in x-axis direction. Inertia matrices and masses will be defined individually for each body. Crank has a mass of 0.12 kg and inertia matrix of:

$$\boldsymbol{I}_{cr} = \begin{pmatrix} 0.0001 & 0 & 0 \\ 0 & 0.00001 & 0 \\ 0 & 0 & 0.0001 \end{pmatrix}$$

Connecting rod has a mass of 0.5 kg and inertia matrix of:

$$\boldsymbol{I}_{co} = \begin{pmatrix} 0.004 & 0 & 0 \\ 0 & 0.0004 & 0 \\ 0 & 0 & 0.004 \end{pmatrix}$$

And the slider has a mass of 2.0 kg and inertia matrix of:

$$\boldsymbol{I}_{s} = \begin{pmatrix} 0.0001 & 0 & 0 \\ 0 & 0.0001 & 0 \\ 0 & 0 & 0.0001 \end{pmatrix}$$

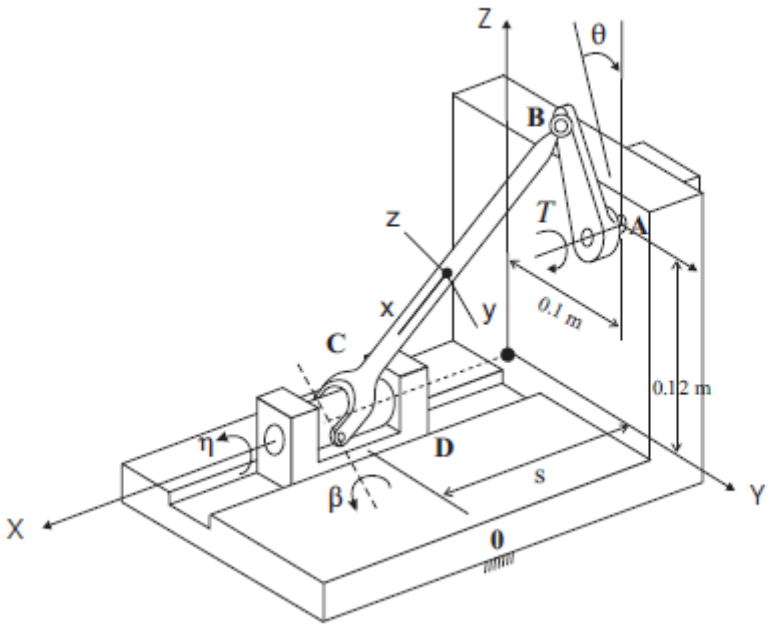

Figure 12. 3-dimensional slider-crank mechanism with dimensions for links (Masoudi, p. 1).

To verify the correct behavior of the model, the visual representation is first created in each physics engine. After model verification, the visual rendering is removed and the simulation is performed as fast as possible recording slider position in each loop and the time it takes to complete the loop. Simulation accuracy using different time steps is compared with the results from Adams. Adams is a widely adopted multibody dynamics simulation software and as such is a prime candidate to compare the results with. Unlike ODE and Chrono, it contains a graphical user interface and therefore modeling workflow contains fewer steps as model verification can be done simultaneously. It is developed by MSC Software and is available as a closed source product (Adams 2017).

Contact between the slider and ground is tested in ODE to see if there is a significant increase in loop times. The test is performed by inserting guiding blocks around the slider so that the slider moves between these blocks and the step time is compared to the one without with the assumption that adding the contact condition the step time will increase. Blocks are inserted because the sliding constraint needs to be removed in order that the contact between ground and the slider occurs and since there is not sliding constraint, the slider would move uncontrollably.

Since this thesis is a part of a larger project, the purpose of which is to develop a virtual movement coach, a hand is modeled in ODE. There are various aspects that need to be covered related to biomechanical hand model, such as the finger placements and lengths/widths, joint types and limits as well as contact determination. In the end, the hand should be able to perform any actions that the virtual coach might demand.

The hand model includes three bodies for each of the five fingers and a palm body to which each of the finger's first body is connected. Initially the joints are defined as hinge joints for index, middle, ring and pinkie and a universal joint for the first thumb body. The second and the third thumb body will have hinge joints connecting them to the previous body and the index's, middle's, ring's and pinkie's second and third body are constrained in the same manner. Since the fingers are not allowed to move beyond certain angles, the limiting angles have to be specified also. There will be both left and right hand created.

The geometry of the hand model is first created by a Matlab script (Hou 2017). Results are used in an ODE project via an implementation of a static library component which was created just to be used in this project. The hand itself is verified through simple simulations such as fixing the palm and letting fingers drop as they are affected by gravity. The flagship demonstrator, virtual coach, has to take contact into account, therefore the hand model has to work with contact.

3.3 Third-party benchmarks

There have been several third parties that have benchmarked physics engines in a multitude of ways. Among the most common features to be benchmarked are collision accuracy and performance, constraint stability, interpenetration of rigid bodies and naturally some task-related performance benchmarks, such as robotic grasping. In this chapter, noteworthy studies are reviewed briefly.

Qualitative physics engine evaluation methods presented by Boeing & Bräunl provided this thesis some subcategories to describe different physics engines. The template used in subsections of 2.3 – 2.5 is according to Boeing & Bräunl. They performed benchmarking on several physics engines, some of which are no longer relevant. They used their custom-built Physics Abstraction Layer, which provided API to perform identical simulations on each of

the physics engines of their choice and the simulations consisted of simple multibody dynamics, such as a ball dropping and measuring the height it reaches or measuring error in penetration in a collision. ODE excelled in integrator accuracy as well as constraint accuracy, but did require the most time to solve constraints. It is noteworthy that this research serves more as a basis for creating other benchmarks since the software and hardware used have evolved since these benchmarks were performed. (Boeing & Bräunl 2007, pp. 281-288.)

Hummel et al. focused mainly on the topics of restitution, collision performance, constraint accuracy and handling of many objects. The main motivation behind their study was to simulate on-orbit servicing tasks so that they could be used to train astronauts and develop robot control. Collision performance was tested by dropping a varying number of balls to the ground and the results showed that ODE was efficient, outperforming all but Havok physics engine. Accuracy of collision response also showed uplifting results for ODE, but fixed constraint stability was not up to par, although according to ODE source code comments, fixed constraint should be avoided if possible. They also attempted to simulate a screw and a nut to benchmark advanced collision and friction, but ODE could not perform this simulation along with many of the other tested physics engines. The conclusion of the research was that ODE was fast and accurate if collision and constraints were not overloaded, otherwise it resulted in unpredictable behavior. (Hummel et al. 2012, pp. 346-357.)

Erez, Tassa & Todorov came up with quantitative measures to benchmark performance of different physics engines related to specific robotic and gaming tasks in multibody dynamics. They highlight the recursive algorithms combined with modern velocity-stepping method and underline their usefulness in robotic simulations. ODE does not however include recursive algorithm. Experiments to determine physics engine speed and accuracy were robotic grasping, humanoid falling to the ground, planar chain and 27 capsules falling to the floor. Most interesting of the aforementioned experiments is the robotic grasping simulation, results of which are shown in figure 13. ODE seemed to perform up to bar with other well-known physics engines, but the physics engine created by the authors of the paper, MuJoCo, seemed to excel even when faster than real-time performance was required and the accuracy seemed to be orders of magnitude better. ODE did perform the fastest when the simulations consisted of many disconnected bodies. In figure 13 below, the first picture depicts the dynamical system, the second one shows results on how many evaluations per second the

respective physics engine makes and the third one demonstrates speed and accuracy comparison. (Erez, Tassa & Todorov 2015, pp. 4397-4403.)

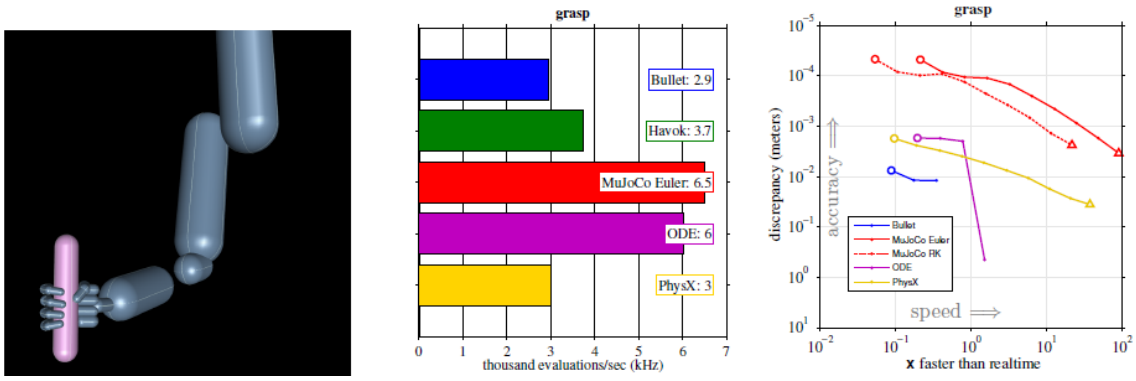

Figure 13. Simulation and comparison results in MuJoCo (Erez, Tassa & Todorov 2015, p. 4400).

In another study by the same group, they argued that MuJoCo could be used in many control optimization and model-predictive control schemes such as the flagship demonstartor introduced in this thesis. Taking into account that optimizing requires a great number of iterations within a timestep (significantly faster than real-time) they claimed that MuJoCo is able to simulate humanoid moving 5000 times faster than real-time with a time-step of 15 milliseconds. It seems that MuJoCo also has implemented some simplified biomechanical models such as a muscle-tendon model. (Todorov, Erez & Tassa 2012, p. 5026.)

4 RESULTS AND ANALYSIS

The results section is split into two main parts, modeling results and the simulation results. The modeling results part review the spatial slider-crank model and the model of hands. The spatial slider-crank was used to benchmark the three aforementioned physics engines whereas the hand-model is tightly related implementation of the flagship demonstrator, virtual movement coach. Simulation results are divided to accuracy and speed results. Speed is benchmarked using the previously introduced spatial slider-crank mechanism using selected integrators and solvers of each physics engines. Speed and accuracy are measured on the same simulation runs. Used closed source software and the corresponding versions are listed in the table 3 below. Latest versions of ODE and Chrono were pulled 12.4.2017 and 10.4.2017 respectively.

Table 3. Closed source software and their respective versions.

Microsoft Visual Studio Community 2015	Version 14.0.25431.01 Update 3 with .NET Framework version 4.7.02053
MeVEA Modeller	V2.2.598
MeVEA Solver library	7.70.2466
MathWorks Matlab	R2015b version 8.6.0.267246
MSCSoftware Adams	X64 2012

4.1 Spatial slider-crank modeling

The slider-crank model in ODE –environment is shown in figure 14. Referring to workflow displayed in 3.2, the graphical demonstration only serves as a mean to verify the model. Graphics are rendered using Open Graphics Library (OpenGL) interface in both ODE and Chrono models.

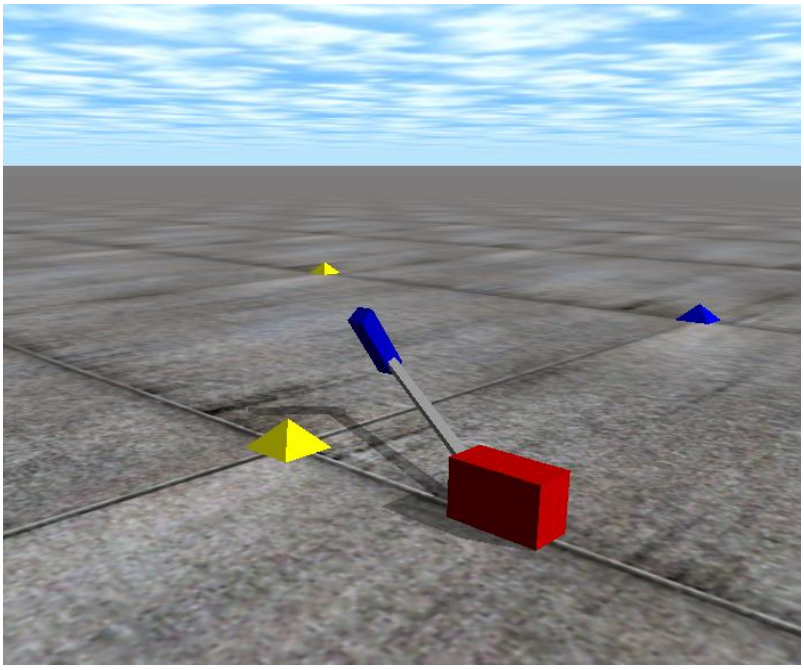

Figure 14. Spatial slider-crank in ODE.

In the figure 14, red block is the slider and blue block is the crank and grey block connecting them is the connecting rod. Yellow and blue pyramids simply display the simulation area and affect the simulation in no way. Comparison of the simulation times with and without contact condition is performed by inserting guide blocks with a high density around the slider block and removing the prismatic joint that guides the slider in Y-axis. This comparison does not address the precision of movement. The model with contact condition is displayed in figure 15 below.

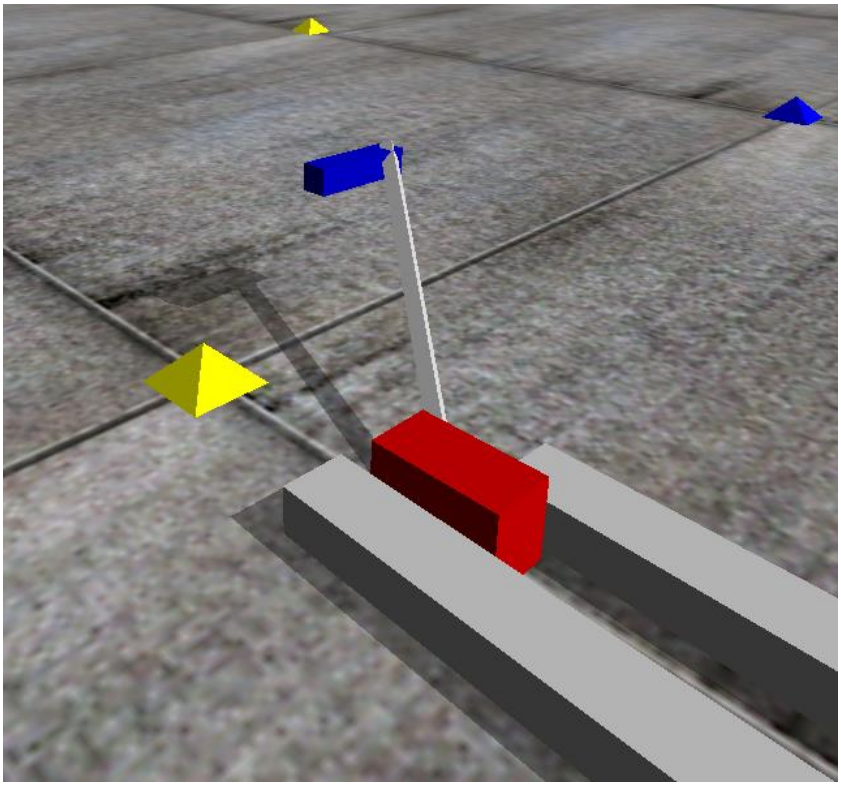

Figure 15. Contact condition simulation.

The two additional grey blocks are the supporting blocks used in the simulation. They are not on level to mitigate errors regarding the track slider is moving in.

As stated earlier, the workflow in Chrono can look quite different to the one in ODE, although in this project the workflow is fairly similar. The model is first verified graphically and after that the model is benchmarked. Graphical verification is shown in figure 16, colors are similar to the ones shown in earlier graphical representations.

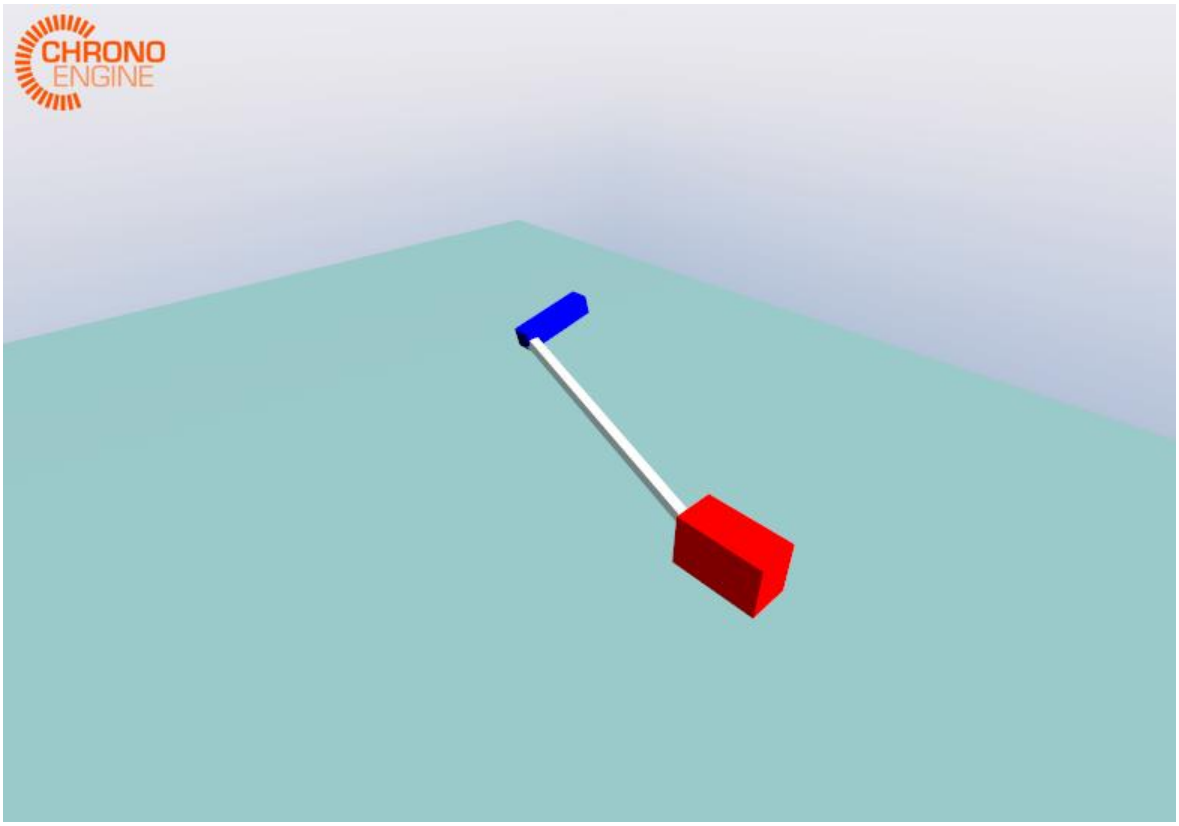

Figure 16. Spatial slider-crank in Chrono::Engine environment.

The MeVEA model is displayed in figure 17. MeVEA model was used in benchmarking similarly as ODE and Chrono.

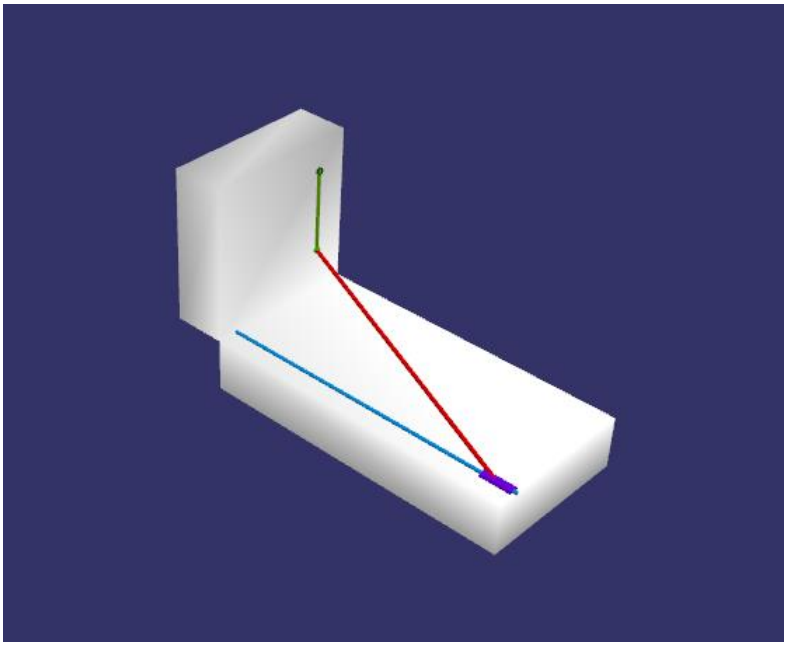

Figure 17. Spatial slider-crank in MeVEA Modeller software.

In the MeVEA model the green link represents the crank, red link the connecting rod and the purple box the slider, while the blue line illustrates the prismatic constraint. The grey blocks play no part in the simulation.

For benchmarking purposes, a model in Adams is created as well. In the figure 18 below, the spatial slider-crank is shown in the Adams environment with the bodies highlighted. Model verification in Adams is shown in figure 19 which states that there is only one degree of freedom and there are no redundant constraints.

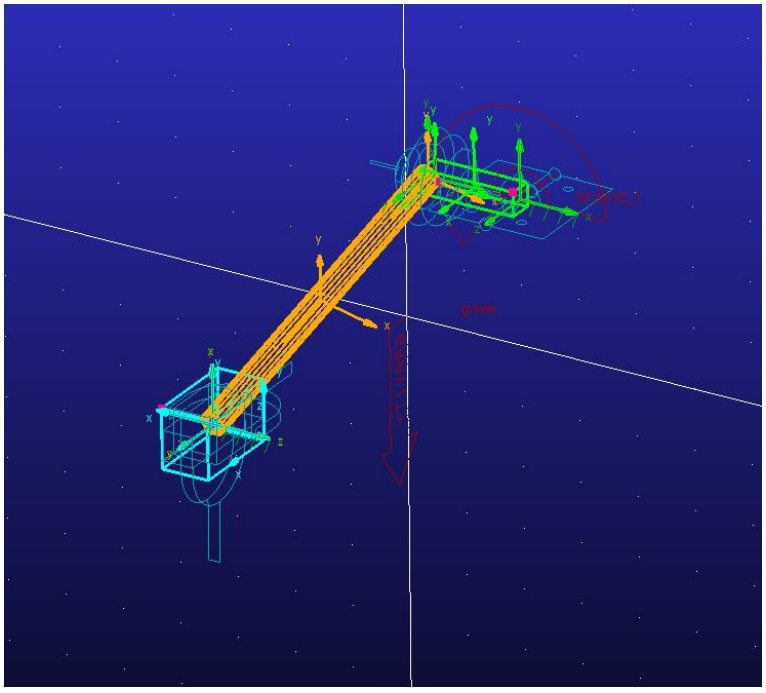

Figure 18. Spatial slider-crank in Adams.

```
Model verified successfully
VERIFY MODEL: .spatial_slider_crank

   1 Gruebler Count (approximate degrees of freedom)
   3 Moving Parts (not including ground)
   1 Revolute Joints
   1 Spherical Joints
   1 Translational Joints
   1 Universal Joints

   1 Degrees of Freedom for .spatial_slider_crank

There are no redundant constraint equations.


Model verified successfully
```

Figure 19. Model verification in Adams.

4.2 Hand modeling

Not accounting the CAD-models, hand model was created exclusively in the ODE environment. It consisted of a palm body and fingers, which in turn include 3 bodies. Hand model was not benchmarked, it rather served as a simple demonstration. Naturally both hands were modeled. The hands in ODE are depicted in figure 20.

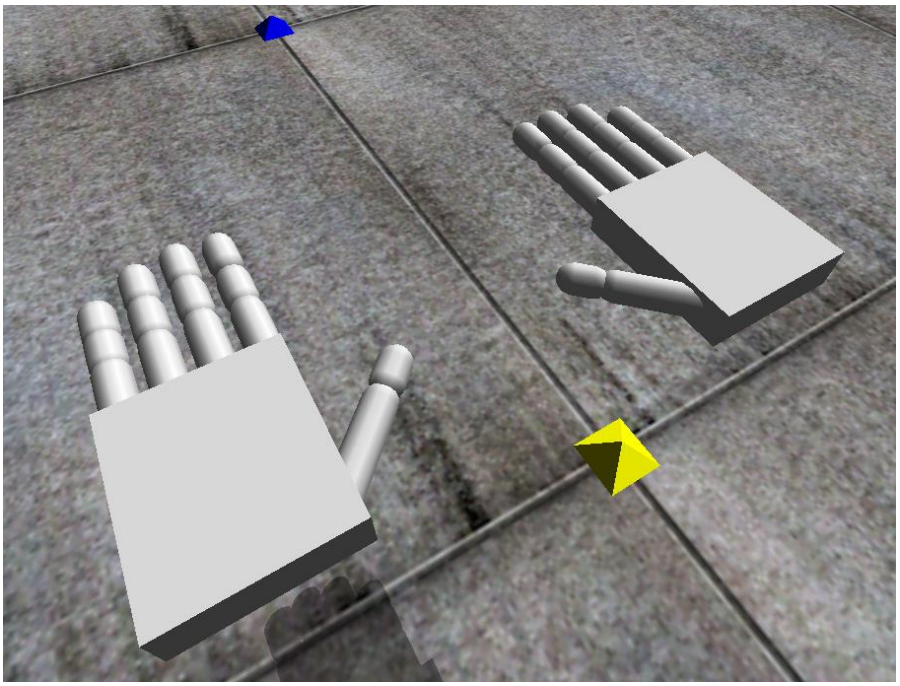

Figure 20. Hands modeled in ODE –environment.

While the index, middle, ring and pinkie fingers consist of hinge joints, the first thumb joint is a universal joint. However, it is possible to model each joint as universal if that is required. Since ODE does not recommend using fixed joint type, the palm is attached to the world using a hinge joint. Position of hands can be changed by altering globally defined X, Y and Z position values and the rotation is determined by rotation axis vector and angle, which are used to form a rotation matrix, which are likewise globally defined.

Control scheme for the hands was also created. It is based on a PID-controller that controls the angle of finger joints by setting torques to the joints according to the input parameter. The purpose of the controller was to use it as a verification tool rather than actually control the hand in a specific way. The problems that arose with the hand model were related to instability of the joints, LCP errors were frequent and could be credited to joint limits, especially thumb part proved to be quite problematic.

4.3 Simulation speeds

The speeds of the simulations are displayed in the tables below. The speeds are compared to real-time performance, so that the numerical value indicates how much faster than real-time the simulation was. For each physics engines the tested simulation time steps were 10, 5, 2, 1, 0.5 and 0.1 milliseconds, except MeVEA, which couldn't output results at 0.1 millisecond time step. For ODE, the tested solvers were the dWorldStep and dWorldQuickStep and in addition the model was benchmarked with contact condition. After initial tests with Chrono solvers in graphical environment, it quickly became apparent that only SOR and its alternatives and Barzilaiborwein were the only solvers that displayed reasonable results, however simulation speeds could be compared with other solver as well. For Chrono the integrators tested were Euler implicit linearized and Euler implicit projected, which have a basis on the same theory as the ones in ODE. MeVEA used the Lagrange penalty formulation with a Gear stiff integrator and an error tolerance value of 10E-9. Gear stiff integrator with SI1 formulation was also utilized in the Adams reference result simulation.

In table 4, ODE speed comparison is displayed, which is scaled to how much faster than real-time the simulation was. The comparison includes both solvers with and without contact condition. Worldstep and WorldQuickstep solvers were introduced in previous chapters.

Table 4. ODE speed comparison (times faster than real-time).

Stepsize [ms]	Worldstep	Worldstep with contact	WorldQuickstep	WorldQuickstep with contact
10	46,76	22,20	49,76	22,49
5	26,21	11,66	22,26	11,51
2	9,78	4,55	8,94	4,67
1	4,77	2,34	3,15	1,77
0,5	2,40	1,17	2,29	1,17
0,1	0,32	0,19	0,31	0,18

In table 5, respectively, Chrono speed comparison is displayed with the Euler implicit projected integrator. Contact condition is not benchmarked in Chrono engine.

Table 5. Chrono speed comparison with Euler implicit projected integrator (times faster than real-time).

Stepsize [ms]	APGD	Barzilaiborwein	Solver_DEM	SOR_MULTI	SOR
10	1,12	2,08	3,34	3,43	3,53
5	0,57	1,04	1,71	1,76	1,71
2	0,23	0,42	0,72	0,79	0,71
1	0,11	0,21	0,36	0,39	0,36
0,5	0,06	0,11	0,18	0,20	0,17
0,1	0,01	0,02	0,04	0,04	0,03

In a similar manner, table 6 contains Chrono speeds using Euler implicit linearized integrator with various solvers.

Table 6. Chrono speed comparison with Euler implicit linearized integrator (times faster than real-time).

Stepsize [ms]	APGD	Barzilaiborwein	Solver_DEM	SOR_MULTI	SOR
10	1,94	3,75	5,73	6,67	5,81
5	1,06	1,95	2,86	3,25	2,96
2	0,44	0,79	1,33	1,35	1,22
1	0,20	0,40	0,64	0,66	0,64
0,5	0,11	0,20	0,33	0,34	0,32
0,1	0,02	0,04	0,07	0,07	0,06

MeVEA results are displayed in the table 7 below. As stated earlier, results for the 0,1 millisecond step time could not be recorded. Integrator used was gear stiff and solver was a Lagrange penalty-based one. MeVEA does have an option of using Runge-Kutta 4 integrator but in this case gear stiff was utilized.

Table 7. MeVEA speed comparison (times faster than real-time).

Stepsize [ms]	Lagrange Penalty
10	85,48
5	54,67
2	22,79
1	11,52
0,5	6,02

4.4 Simulation accuracy

Even though accuracy of the simulation has to be somewhat compromised to gain a boost in simulation speed, the accuracy was measured nevertheless. To measure the accuracy, a torque was applied to the crank and the velocity of the slider was measured on each time step and the simulation was performed to the 2 second mark. Results for the slider velocity from Adams served as a comparison reference. Adams reference results were achieved using a 0.1 millisecond time step and an error tolerance value of 10^{-9} . The results are displayed in tables as root mean squared error [m/s]. In table 8, ODE's simulation accuracy is displayed, smaller value means more accurate simulation (velocity RMS deviation).

Table 8. ODE simulation RMS deviation with Worldstep and WorldQuickstep integrators.

Stepsize [ms]	Worldstep	WorldQuickstep
10	1,08	0,97
5	1,08	0,98
2	1,15	1,05
1	1,12	1,09
0,5	0,45	1,16
0,1	0,09	0,31

In table 9, RMS error of Chrono engine with Euler implicit linearized integrator is displayed. Cells marked with "INVALID" did not yield reasonable values.

Table 9. Chrono simulation RMS deviation using Euler implicit linearized integrator.

Stepsize [ms]	APGD	Barzilaiborwein	Solver_DEM	SOR_MULTI	SOR
10	INVALID	1,19	1,04	INVALID	1,18
5	INVALID	1,16	1,22	INVALID	1,33
2	INVALID	1,06	1,57	INVALID	1,60
1	INVALID	1,08	1,90	INVALID	1,21
0,5	INVALID	1,09	1,46	INVALID	1,14
0,1	INVALID	1,23	1,13	INVALID	1,16

Likewise, in table 10, the values are displayed when using the Euler implicit projected integrator.

Table 10. Chrono simulation RMS deviation using Euler implicit projected integrator.

Stepsize [ms]	APGD	Barzilaiborwein	Solver_DEM	SOR_MULTI	SOR
10	INVALID	1,08	1,10	INVALID	1,12
5	INVALID	1,08	1,20	INVALID	1,20
2	INVALID	0,99	1,46	INVALID	1,27
1	INVALID	1,06	1,29	INVALID	1,20
0,5	INVALID	1,06	1,54	INVALID	1,13
0,1	INVALID	1,16	1,15	INVALID	1,38

MeVEA results are displayed in table 11. As stated earlier the 0,1 millisecond step results could not be obtained.

Table 11. MeVEA simulation RMS deviation comparison.

Stepsize [ms]	Lagrange Penalty
10	1,19
5	1,24
2	1,39
1	1,42
0,5	1,43

Following figures 21 and 22 summarize achieved results in terms of speed and accuracy. Only the best performing Chrono solver is displayed as solvers' results didn't differ much in relation to each other.

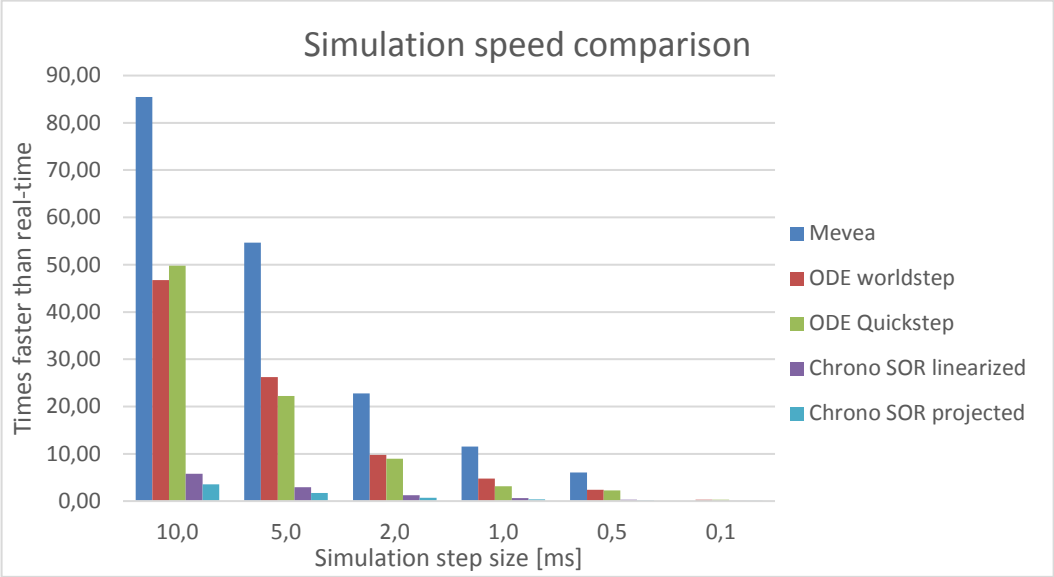

Figure 21. Simulation speed results summary.

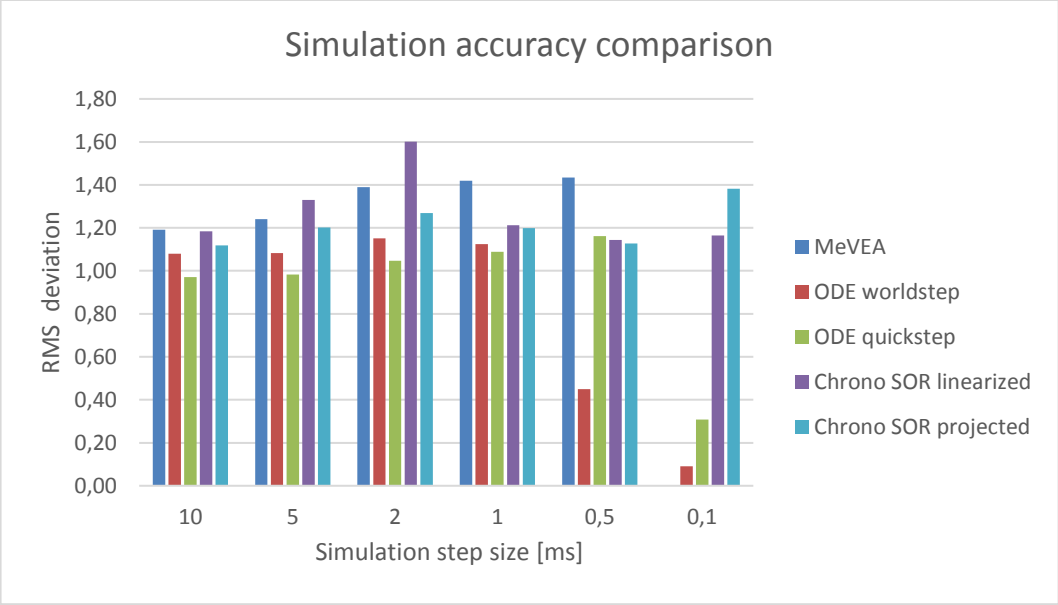

Figure 22. Simulation accuracy results summary.

4.5 Analysis

Based on the documented results it became apparent, that good accuracy is not achieved on any of the physics engines, however ODE seems to converge the best when decreasing the time step, but that would require a very small time step, which is not applicable. Since the torque is enough to rotate the crank, the slider's maximum velocity should be increasing as time goes on. Physics engines showed that the maximum velocity increased until some point in time, after which the velocity maximum no longer increased. Results support the idea that the point after which maximum velocity no longer increased was affected by time step on each physics engine. This behavior is demonstrated in figure 23 below. Damping coefficients for both linear and angular motion were set to zero in ODE. The reason for ODE and Chrono damping the system much earlier is believed to lie in the integrator that both of these physics engines utilized. Implicit integrators have been known to decrease energy in the systems, which can be translated to damping (Smith 2006, p. 82).

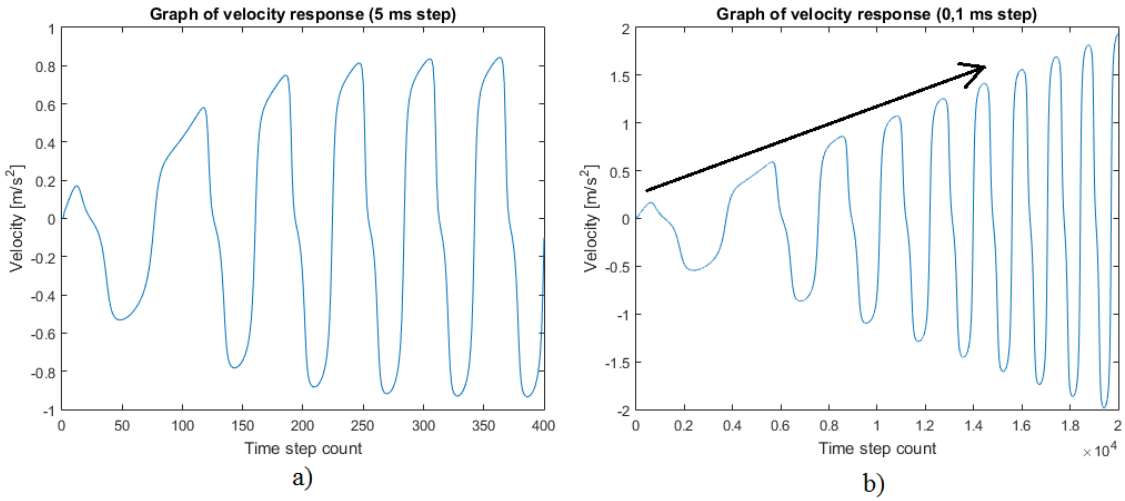

Figure 23. Velocity response graphs of ODE's worldstep solver with (a) 5 ms time step, (b) 0,1 ms time step.

The maximum velocity should increase as demonstrated by the arrow in figure 23b. This was not witnessed especially when larger time step values were used. Adams did not in turn damp the system even with larger tolerances or step sizes.

A rather surprising result was that Chrono could not perform up to bar in terms of accuracy. Especially the APGD solver, which was presumed to be the most accurate, could not yield

any reasonable results. Analyzing the behavior of APGD solver in graphical representation showed that the constraints were not forced at all. This resulted in the slider "falling" through the prismatic constraint axis and hinged crank rotating about another axis. Similarly some of the integrators that were used with Chrono solvers demonstrated incapability to solve the problem. Euler explicit integrator not being applicable was expected but Runge-Kutta 4 making the simulation go unstable was a surprise, since it was expected to be the most accurate integrator.

As to the speeds, it was expected that ODE would be the fastest and Chrono the slowest with MeVEA somewhere in between. However, MeVEA proved to be the fastest of the three, while Chrono was clearly the slowest. MeVEA was not only marginally faster but almost two times faster than ODE, even though the tolerance was set to a low value (10^{-9}) in MeVEA, which should increase the solution times. Chrono wasn't even close to as fast as the other two physics engines, which to some extent was an expected result. The vast difference in speed of Chrono and others was still quite unexpected since even the iterative solver algorithm of Chrono (SOR) coupled with a similar integrator scheme as ODE (Euler implicit) was still more than ten times slower than ODE and MeVEA. It was interesting to discover that SOR multithreaded didn't provide any boost in speed and yielded inaccurate results.

The speed of the contact condition in ODE slowed the system down approximately by a factor of two. This was to be expected since the LCP solver uses a number of inner loops to approximate the contact. When contact was taken into account it was noteworthy that the step size should be quite small due to the impulse-based method of Baraff. With a bigger step size, the bodies seem to vibrate on the ground and appear instable due to impulse force applied by the contact condition and gravity pulling bodies down. Therefore if accurate contact modeling is a necessity, the contact algorithm need to be evaluated more elaborately.

Even though the hand model was not a part of simulation speed measurement or accuracy comparison, it still provided key takeaways with regards to the applicability of ODE in simplified biomechanical models. First of all, ODE seemed to go unstable quite easily, when using universal joints. Universal joints require a number of restricting parameters to be set that stabilize the joints. The way they were set in this thesis' model was by first limiting the

maximum turns of each hinge axis to infinity and negative infinity. This is due to there being the low stop and high stop and this step was just the preliminary stage. After this the actual joint angular limits for each axis (two axes for universal joint) could be set with low stop being less than 0 and high stop above 0.

According to ODE documentation, the joint stops suffer from a "jumping motion" when the joints is moved away from a stop. To counteract this a special fudge factor is also applied to the universal joints, since it became apparent that they were affected by this twitching motion. The last set parameter was a bounce parameter that affected the joints' restitution when hitting end stops. (Smith 2006, p. 40.)

As with the hinge joints, only applied parameters were high and low stops, but empirical study showed that neither the low stop, nor high stop should be set to value of zero. Setting a stop to a value of zero increases the possibility of a LCP error, which results in joint error. A mock-up PID algorithm was also programmed to enable controlling of the hand. In all its simplicity, it takes the angle of the joint in question and applies a torque according to the preferred angle of the joint. PID controller consists of proportional, integral and derivative part, which all contribute to the controller in a different way. Proportional part simply multiplies the difference between the observed angle and the goal angle, while the integral part sums up all the differences in time and should decrease the steady-state error. The derivative part is used to decrease the overshoot but does require intricate testing to get right. When using the hand control scheme, ODE's global Error Reducing Parameter (ERP) was seen to cause some instability when a large value was used (0.6 - 0.9). Controller itself required that the time step is set to a reasonably low value due to the predetermined gain values. Also, if the values are tweaked, it is evident that the gains of the proportional and especially the derivative part have quite a significant role in the stability of the system.

Since positions and rotations of the hands are defined in global parameters, matrix multiplication need to be performed to set the correct rotational values for bodies. ODE has built-in rotation matrix multiplication functionality that was utilized. Multiplying two rotation matrices is done in ODE by first transforming the matrices to quaternions and then multiplying these two and after that transforming the quaternion representation back to transformation matrix form. Therefore, ODE prefers quaternion representation.

As of right now, hand model uses capsule geometries to describe fingers. Dimensions of these capsules comes from a script that determines the dimensions according to the mass and length of the human. However, usage of capsules has one problem, when setting the joints between finger bodies, last body of the finger may intersect the first resulting in an unsolvable joint setup. This is demonstrated in figure 24 below.

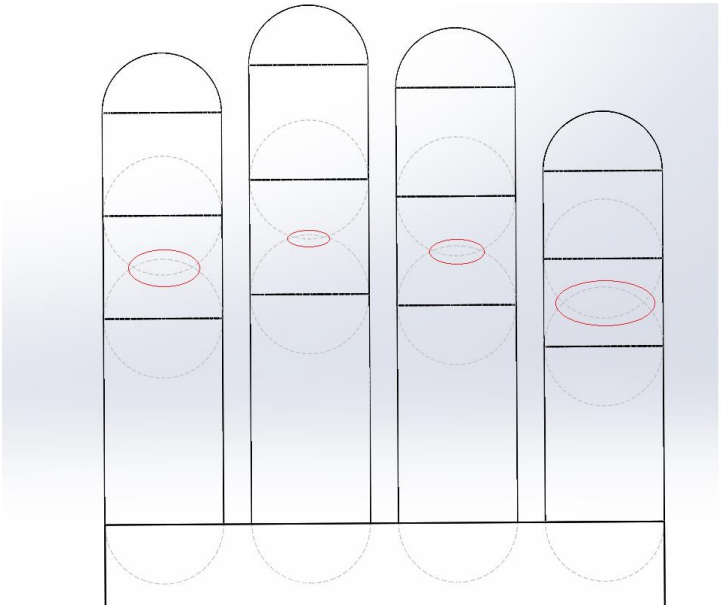

Figure 24. Overlapping joint setup.

As can be seen in the above figure, areas with a red circle, cause issues when creating joints between finger bodies in ODE. The issue is caused by the definition of the capsule length, which is the cylinder distance that does not consider the width of the capsule. To counteract this, bodies of each finger needed to be moved further away from each other. Although this does result in inaccurate finger dimensions, it is the only way to form fingers using capsules. The alternative is to use cylinders.

4.6 Reliability, validity and error analysis of the study

Reliability of the studies related to spatial slider-crank model are good. The model is fairly simple and easy to program to the physics engines. Reliability of the models is further increased by the used workflow, which starts of by making the visual representation to validate the model. Sources of error in simulations stem from numerical approximation, which is impossible to avoid. Certain algorithms do approximate more, but since the goal of

this research was not to achieve accurate results to a specific problem but rather compare the results, errors are to be expected. Although errors and bugs can occur, they can be traced to be a part of the source codes provided by the physics engines. These errors are out of the scope of this thesis as it would require significant time to research both ODE's and Chrono's source code elaborately. Obviously, the major features have to be considered, for example the ERP and CFM values of the ODE. These were set to lower limits, so that they have no effect on results.

Validity of the hand model is a bit harder to evaluate. Since the visual representation is working with the control scheme, the model can be said to be valid. Although some questionable behavior was witnessed with the thumb joint, decreasing the time step to a more reasonable value and tweaking the joint parameters alleviated the problem. Moreover, there is always room for improvement, especially when it comes to programming. For example, the empiric values used to form geometries of the fingers is somewhat arbitrary. Likewise, the joint parameters and PID gains are experimental. Since the hand model was not used in benchmarking and only served to evaluate the feasibility of ODE in simplified biomechanical simulation, it has achieved its' task.

4.7 Further research

After implementing the hand model, the flagship demonstrator obviously requires for the rest of the human body to be modeled with sufficient accuracy. This will obviously require more computational effort to simulate, which is why topics of interest going forward include many upcoming additions to the performance of physics engines. Parallel computing and processing is a much-anticipated addition which covers the utilization of graphical processing unit as well as taking advantage of the multiple cores of a processor efficiently. Emerging new formulations, for example the recursive algorithm briefly introduced in this thesis, could provide an essential boost to computational power. In addition to the software the amount of computational power machines can output based on hardware will increase over time, thus providing faster simulations.

The hand model should be studied in detail and optimized as needed. At the moment, there has not been a simulation where the hand grasps on to something, which is a necessary feature in climbing exercise.

5 CONCLUSIONS

In this thesis, the feasibility of using Open Dynamics Engine, Chrono and MeVEA in biomechanical modeling was examined. The research was done by studying the performance of each these physics engines' in a simple multibody benchmark. The goal was to determine, which would be the quickest to solve the problem and what would be the accuracy when compared to reference results by Adams. ODE has been regarded as the prime candidate by the main research workers in this joint academia project and thus a simple model of human hand was created in the ODE environment. Literature review introduced basic concepts of multibody dynamics and studied the properties of Chrono and ODE and third-party benchmarks were also introduced to clarify some of the drawbacks that have been encountered with these physics engines.

Many sources refer to ODE as a tool for gaming rather than a qualitative simulation tool as it applies several approximations thus sacrificing accuracy for speed. It has fared quite well in the earlier benchmarks, but in the more recent ones it did not perform up to bar in grasping tasks, which is an essential part of the flagship demonstrator. Some improvements have been proposed throughout the years and the community is still actively involved in its development since the original developer has left the project.

Meanwhile project Chrono is actively developed in the University of Wisconsin-Madison and it is aimed to be used in qualitative tasks. While it showed great promise, it is still for many parts a work-in-progress. Nevertheless, future improvements will include some interesting features, such as parallel computing.

The results for the benchmarks showed that ODE was tens of times faster than Chrono, but MeVEA was the fastest of the three being about two times faster than ODE. Surprisingly the accuracy of Chrono was nowhere close to what was expected and ODE was the clear winner in this category. Despite being the most accurate, ODE required a very small time step to yield accurate results. Adding contact condition doubled the time it took to solve the simulation in ODE.

Creating the hand model in ODE environment was not a quick task. Since there is no graphical method of creating geometries, the geometry has to be coded using C-language. The workflow requires the usage of ODE's own drawstuff library to validate the model, although any other user implemented library would work. Compared with Adams, the model takes significantly longer to create, but nevertheless the documentation is satisfactory and functions are organized in a coherent manner. As for the model itself, there were some problems with joints that were solved using several distinct joint parameters and decreasing the time step. The original hand model had to be tweaked slightly in order to not violate ODE's internal rules. In the end, the hand model was controllable and was performing tasks accordingly.

All in all, the goals of the research were met as the benchmarks' results shed light into the applicability of chosen physics engines. The model of hands also formed a basis for expanding the model to include the rest of the human form. This will be a high-level goal in future research as the combined research project, that this thesis is a part of, requires the whole human model. The advances of formulations and increased efficiency in hardware usage will be points of interest as the research project progresses.

REFERENCES

Adams. 2017. [At MSC Software's Adams product www-pages]. [Referenced 23.9.2017]. Available: http://www.mscsoftware.com/product/adams

Anitescu, M. & Potra, F. A. 1997. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. Nonlinear Dynamics, 14. Pp. 231-247.

Baraff, D. 1989. Analytical methods for dynamic simulation of non-penetrating rigid bodies. Proceedings of the 1989 conference on Computer graphics. Boston, Massachusetts, USA 31.7-4.8.1996. SIGGRAPH '89. Pp. 223-232.

Baraff, D. 1994. Fast contact force computation for nonpenetrating rigid bodies. Proceedings of the 1994 conference on Computer graphics. Orlando, USA 24-29.7.1994. SIGGRAPH '94. Pp. 23-34.

Baraff, D. 1996. Linear-Time Dynamics using Lagrange Multipliers. Proceedings of the 1996 conference on Computer graphics. New Orleans, Louisiana, USA 4-9.8.1996. SIGGRAPH '96. Pp. 137-146.

Boeing, A. & Bräunl, T. 2007. Evaluation of real-time physics simulation systems. Proceedings of the 2007 conference on Computer graphics and interactive techniques in Australia and Southeast Asia. Perth, Australia 1-4.12.2007. GRAPHITE '07. Pp. 281-288.

Brief Introduction. 2016. [At Project Chrono www-pages]. [Referenced 29.5.2017]. Available: http://api.projectchrono.org/introduction_chrono.html

Burgermeister, B., Arnold, M. & Esterl B. 2006. DAE time integration for real-time applications in multi-body dynamics. Journal of Applied Mathematics and Mechanics, 86: 10. Pp. 759-771.

Callejo, A., Narayanan, S. H. K., García de Jalón, J., Norris, B. 2014. Performance of automatic differentiation tools in the dynamic simulation of multibody systems. Journal of Advances in Engineering Software. Vol. 73. Pp. 35-44.

Chaudhary, H. & Saha, S. K. 2009. Dynamics and Balancing of Multibody Systems. Berlin: Springer-Verlag. 178 p. Lecture Notes in Applied and Computational Mechanics. Volume 37.

Chrono::Engine. 2016. [At Project Chrono www-pages]. [Referenced 29.5.2017]. Available: http://api.chrono.projectchrono.org/group__chrono.html

Chrono Frequently Asked Questions. 2016. [At Project Chrono www-pages]. [Referenced 29.5.2017]. Available: http://projectchrono.org/faq/

Cottle, R. W. & Dantzig, G. B. 1968. Complementary pivot theory of mathematical programming. In: Hoffman, A. J. (editor-in-chief). Linear Algebra and its Applications. Volume 1:1. Pp. 103-125.

Cottle, R. W., Pang, J-S, Stone, R. E. 1992. The linear complementarity problem. Boston: Academic Press. 761 p. Classics in applied mathematics. Volume 60.

Coutinho, M. G. 2013. Guide to Dynamic Simulations of Rigid Bodies and Particle Systems. London: Springer-Verlag. 399 p.

Drumwright, E., Hsu, J., Koenig N. & Shell, D. 2010. Extending Open Dynamics Engine for Robotics Simulation. Proceedings of the 2010 conference on Simulation, modeling, and programming for autonomous robots. Darmstadt, Germany 15-18.11.2010. SIMPAR ’10. Pp. 38-50.

Erez, T., Tassa, Y. & Todorov, E. 2015. Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX. Proceedings of the IEEE International Conference on Robotics and Automation. Seattle, WA, USA: 26-30.5.2015. Pp. 4397-4404.

Erleben, K. 2005. Stable, Robust, and Versatile Multibody Dynamics Animation. Copenhagen, Denmark: University of Copenhagen. 222 p. Ph.D. dissertation.

Haataja, J., Heikonen, J., Leino, Y., Rahola, J., Ruokolainen, J. & Savolainen, V. 1999. Numeeriset menetelmät käytännössä. Helsinki: Picaset Oy. 381 p. Written in Finnish.

Hairer, E., Norsett, S. P. & Wanner, G. 1993. Solving Ordinary Differential Equations I. 2nd edition. New York: Springer-Verlag. 528 p. Springer Series in Computational Mathematics. Volume 8.

Hou, A. 2017. Master of Science (Technology), Lappeenranta University of Technology. Email interview 3.5.2017. Interviewer Niko Niemi. Notes are occupied by the interviewer.

Hsu, J. M. & Peters, S. C. 2014. Extending Open Dynamics Engine for the DARPA Virtual Robotics Challenge. In: Brugali, D., Broenink, J. F., Kroeger T. & MacDonald, B. A. Simulation, Modeling, and Programming for Autonomous Robots. SIMPAR 2014. Lecture Notes in Computer Science, vol. 8810. Cham: Springer. Pp. 37-48.

Hummel J., Wolff R., Stein T., Gerndt A. & Kuhlen T. 2012. An Evaluation of Open Source Physics Engines for Use in Virtual Reality Assembly Simulations. In: Bebis G. et al. Advances in Visual Computing. Proceedings of International Symposium on Visual Computing. Lecture Notes in Computer Science, vol. 7432. Springer, Berlin, Heidelberg. Pp. 346-357.

Hämäläinen, P. 2016. Virtual Coach Based on Multibody Dynamics – VIMU. Research Plan. 15 p.

Introduction to modules. 2016. [At Project Chrono www-pages]. [Referenced 29.5.2017]. Available: http://api.projectchrono.org/modularity.html

Kajastila R., Holsti, L., Hämäläinen, P. 2016. The augmented climbing wall: high-exertion proximity interaction on a wall-sized interactive surface. Proceedings of the 34th annual

conference on Human Factors in Computing Systems. San Jose, USA 7-12.5.2016. Pp. 758-769.

Masoudi, R. 3D Rigid Slider-Crank Mechanism [online document]. Library of Computational Benchmark Problems. [Referenced 14.4.2017]. 3 p. Available as PDF-file: https://www.iftomm-multibody.org/media/problems/spatial%20rigid%20slider-crank%20mechanism/spatial%20rigid%20slider-crank%20mechanism.pdf

Project Chrono – An Open Source Physics Engine. 2016. [At Project Chrono www-pages]. [Referenced 29.5.2017]. Available: https://projectchrono.org/

Shabana, A. A. 1998. Dynamics of multibody systems. 2nd edition. Cambridge: The Press Syndicate of the University of Cambridge. 372 p.

Shabana, A. A. 2001. Computational dynamics. 2nd edition. John Wiley & Sons, Inc. 502 p.

Simeon, B. 2013. Computational flexible multibody dynamics. Berlin: Springer-Verlag. 249 p.

Simulation system. 2016. [At Project Chrono www-pages]. [Referenced 29.5.2017]. Available: http://api.chrono.projectchrono.org/simulation_system.html#solvers

Smith, R. 2006. Open Dynamics Engine User Guide [online document]. [Referenced 3.2.2017]. 92p. Available as PDF-file: http://ode.org/ode-latest-userguide.pdf

Software for Real-Time Simulation | MeVEA. 2016. [At MeVEA www-pages]. [Referenced 29.5.2017]. Available: http://mevea.com/products/software/

Stewart, D. E. & Trinkle, J. E. 1996. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. International Journal for Numerical Methods in Engineering, 39: 15. Pp. 2673-2691.

Tasora, A. 2016. Time integration in Chrono::Engine [online document]. ProjectCHRONO technical documentation. 44 p. Available as PDF-file: http://www.projectchrono.org/assets/white_papers/ChronoCore/integrator.pdf

Todorov, E., Erez, T. & Tassa, Y. 2012. MuJoCo: A physics engine for model-based control. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems. Vilamoura, Portugal 7-12.12.2012. Pp. 5026–5033.

Volino, P. 2004. Collision Detection for Deformable Objects. In: Hadap, S. & Eberle, D. Collision Detection and Proximity Queries. SIGGRAPH 2004 Course. Pp. 22-23.

Wittenburg, J. 2008 Dynamics of Multibody Systems. 2nd edition. Berlin: Springer-Verlag. 223 p.

Yu, Q. & Chen, I. M. 2000. A direct violation correction method in numerical simulation of constrained multibody systems. Computational Mechanics, 26: 1. Springer-Verlag. Pp. 52-57.

Appendix I

The source codes for benchmarks and hand model.

https://github.com/NikoNiemi/Master-s-Thesis---Niko-Niemi