Simulation and Optimization of Mechanical Alloying using the Event-Driven method

Javier Barahona

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

MASTER OF APPLIED SCIENCE

in Mechanical Engineering

Ottawa-Carleton Institute for Mechanical and Aerospace Engineering
Univesity of Ottawa
Ottawa, Canada

November 2011

Abstract

Mechanical alloying is a manufacturing process that produces alloys by cold welding of powders. Usually, a vial containing both the powder and steel balls is agitated. Due to impact between the balls and balls and the vial, the powder is mechanically deformed, crushed, and mixed at nano-scales. In this thesis, a numerical model is developed to simulate the dynamics of the vial and the grinding balls of the SPEX 8000 ball milling device, a standardized equipment in both industrial and academic investigations of ball milling. The numerical model is based on the Event Driven Method, typically used to model granular flows. The method implemented is more efficient than the discrete element method used previously to study ball milling dynamics. The numerical tool obtained is useful for scale-up and optimization of mechanical alloying of various materials. An optimization study is presented for the SPEX 8000.

To my wife and our two kids.

Acknowledgements

First and foremost I offer my sincerest gratitude to my supervisor Dr. Matei Radulescu, who has supported me throughout my thesis with his guidance and patience. I am deeply in debt with him for allowing me to develop this thesis at my own pace.

I am grateful to the Defence Research and Development Canada (DRDC), Suffield, for having provided parts of the funds used for this thesis.

I would also like to thank the open source software community. The entirety of this thesis has been completed using such programs.

Lastly, I would like to thank my wife who has supported and encouraged me during these last years of study.

Contents

List of Figures

List of Tables

Acronym

ABM	Arrested Ball Milling
ARM	Arrested Reactive Milling
COR	Coefficient(s) of restitution
DEM	Discrete Element Method
EDM	Event-Driven Method
KE	Kinetic Energy
MA	Mechanical Alloying
MASHS	Activated Self-propagating High-temperature Synthesis
MD	Milling Dose
MSR	Mechanically-Induced Self-Propagating Reaction
RTM	Roto-Translational Model
SHS	Self-propagating High-temperature Synthesis
WE	Energy transferred from/to a wall

Chapter 1

Introduction

1.1 Objective

The objective of this work is to develop a numerical tool capable of modelling a ball milling device commonly used in both industrial and academic investigations using the Event-Driven Method (EDM). Ball Milling is a manufacturing process that produces alloys by cold welding (Suryanarayana 2001). In this process, material is introduced into a canister or vial along with grinding balls. The canister is agitated for some time, during which the grinding balls crush the material present between the balls and between a ball and the walls of the canister. The result of this process is typically a new alloy in the form of a powder, with mechanical, chemical and thermal properties different from the original powders. Figure 1.1 illustrates this process. The question that the present thesis addresses is the relation between the energy dissipated per collision, which is used to refine the power, and the energy input in the shaking vial of the milling device. This is achieved via simulations of the vial and grinding balls.

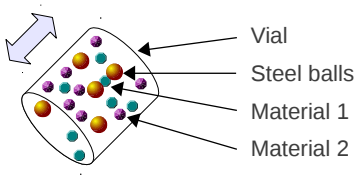


Figure 1.1: Representation of Ball Milling.

The EDM is the technique used here to simulate the dynamics of the vial and the milling balls inside the vials. The method is called "event-driven" because the model is restricted to one collision at a time, which can be predicted analytically. This makes the technique particularly suitable for treating a system of many particles. The technique

was first introduced for molecular dynamic calculations of many particles (Alder and Wainwright 1959), and has now become the standard method to model granular materials (Poschel and Schwager 2005). In the model presented, the balls are considered as single rigid bodies that travel in free-flight trajectories in between collisions. The post collision velocities of the balls are computed using the classical equation of conservation of momentum, whereas the amount of energy dissipated per collision is controlled by the coefficient of restitution (COR) controlling the collision. The value of the COR is found experimentally, and constitutes the free parameter in the model developed. The mechanical, thermal and chemical evolution of the powder itself is not directly modelled; it is accounted for in the amount of energy dissipation per collision via the COR.

EDM has been chosen because it is the most computational-efficient method that simulates systems of many-component which evolve independently (Miller and Luding 2004). The alternative method, used in previous studies of ball milling simulations (Ward et al. 2005) would have been to integrate the dynamics using a force based method. This method requires significant more resources due to finite time stepping, but does not yield more accurate results. The EDM method provides a considerable gain in computational resources without loosing accuracy (Poschel and Schwager 2005). The main motivation of the work is thus to implement and test a novel method for simulating the dynamics of the grinding particles during the ball milling process.

The numerical simulation of ball milling also provides a better understanding of the process. The EDM can recreate the ball milling process with a relatively high degree of accuracy at a macroscopic level. It permits construction and evaluation of statistical models for the impact conditions, and correlate these with experimental observations made during the milling process. It also permits to determine the time resolved rate of energy dissipation in the system, and also correlate this with the milled power evolution. This can thus provide the scaling laws for the milling process. It also permits to optimize the milling process for any desired effect, such as speed of the milling and impact amplitude. From a practical perspective for a given mill, it permits the user to choose the number of grinding balls, the powder mass, and the vial agitation amplitude in order to obtain a desired effect. This tool is also useful for purposes other than ball milling; for instance, molecular dynamics simulations, or other type of granular flows with moving containers (e.g., a salt shaker).

1.2 Spex 8000

This work is devoted to the Spex 8000[1]. The reason for choosing this equipment is that this type of mill has become the standard in laboratory investigations (Suryanarayana 2001) whereby different groups compare their results. It is thus possible to find ample information on different material processing experiments, which can serve for validation purposes of models. This ball milling device is also available in our laboratory, where it is used for reactive ball milling for Self-propagating High-temperature Synthesis (SHS) of materials, as described below.

The Spex 8000 is a small capacity high energy ball milling machine with combined kinematics (Caravati et al. 1999). Figure 1.2a shows a picture of the device. The picture has been edited to show the most relevant components. Figure 1.2b is a schematic representation of the Spex and gives a better idea of how the equipments works. The electric motor converts electric energy into mechanical energy. The belt and pulleys transmit the power from the motor to the driving shaft. There is a reduction of the rotational speed and an increment in the torque since the diameter of the receiving pulley is larger than the pulley of the electric motor. The driving shaft is supported in a bearing case. The shaft transmits the power from the belt and pulleys to the fulcrum. The fulcrum has a ball bearing but the driving shaft is not connected to it. There is a special wedge in between. This wedge makes the shaft and the bearing non-concentric, and this is what makes the vial swing. A pair of arms and an open cage carry the vial. A spring attached to one of the arms limits the range of motion of the vial. The vial follows a figure-eight trajectory, as described in section 2.2.2.

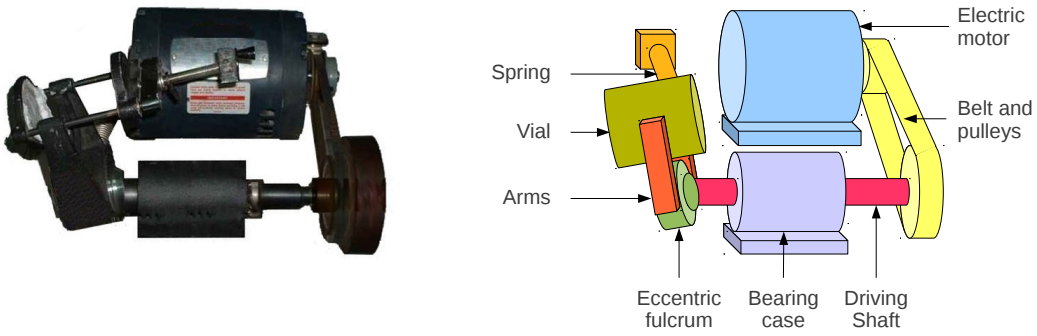


Figure 1.2: Spex 8000. a) Pictured with no vial on place. b): Schematic representation.

[1]The Spex 8000 is manufactured by SPEX CertiPrep, 203 Norcross Avenue, Metuchen, NJ 08840, U.S.A.

1.3 Background

Although it is difficult to track the origins of the ball milling machine, it was reportedly used for grinding pottery materials in 1870, using steam as its power source. Later it became electric and its use for grinding expanded to many other materials (Lynch and Rowland 2005). Uses of ball milling are diverse: it is used in grinding or mixing ores, chemicals, ceramic raw materials, and paints. The process is known as ball milling when it uses metallic balls, usually steel, whereas it is known as rod milling when it uses rods.

It was not until 1966 that the mechanical alloying (MA) arose. The technique consists of mechanically mixing, or alloying two or more components at micrometric (and even nanometric) scales with controllable chemical and phase modifications. It was the result of a long search to produce a nickel-based super-alloy for gas turbine applications (Benjamin 1976). Mechanical alloying is a technique to produce alloys, specially ceramics, composites, and intermetallics. Mechanical alloying has the ability of produce alloys that are not possible by conventional methods, such as melting.

The most important variables that affect the MA process, apart from the material itself, are the type and number of balls, use of additives, type of atmosphere, preheating and cooling temperature control, and the kinematics of the vial, as well as its size and shape. Powder size is produced from millimetres to nanometres. Equipment capacity ranges from a few grams in laboratories to a few kilograms in industry. Available equipment for MA varies in speed, vial size and shape, number of vials, type of movement (i.e. rotation, translation, or combined), and temperature control. Also, in some equipment the vial is fixed and what moves is a series of rods attached to a rotating shaft. Since equipment for MA works at higher speeds than conventional ones, these are referred to as high energy milling devices. The use of rod milling is very limited, and practically, mechanical alloying is identified with ball milling only. MA has been reviewed by Suryanarayana (2001), El-Eskandarany (2001), Chawla et al. (2007), Dunlap et al. (2000), and Koch et al. (2010).

When powder is crushed, many intricate transformations occur in its internal structure and sometimes at atomic levels. There is fracture, elastic deformation, plastic strain, layers displacement, cold welding, and other micro-structural effects. All these transformations are the normal mechanisms by which the collision energy of the grinding balls is dissipated in order to produce the desired size, shape and structure. Figure 1.3 shows a schematic illustrating the powder evolution in time of two constituents, with the possibility of appearance of new phases (shown in black).

One of the applications of the tool developed in this study is to find the time in

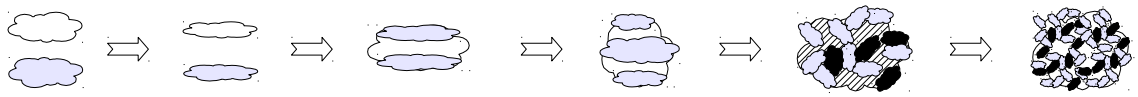


Figure 1.3: Transformation of powder in MA.

which a certain amount of mechanical work has been deposited in the powder. This time is very important in the preparation of source material for another process known as Self-propagating High-temperature Synthesis (SHS)[2]. SHS is a technique for producing advanced materials, resulting from a reaction wave that propagates from one part of the material to the rest. This wave motion is sustained by the local heat generation by exothermic alloying and by heat diffusion towards the colder material, which in turn ignites once it reaches an auto-ignition temperature. The process permits production of alloys with tunable properties. For example, TiC, the product of elemental Ti and C undergoing the SHS process, has high abrasiveness, even higher than the same products prepared by furnace technology. The composite powder TiC + Cr_3C_2 + Ni has found its application as a wear- and heat-resistant coating for the aircraft industry. SiN_4 + SiC + TiN is effective for the friction surfaces of ball-bearings (Merzhanov 2004). The SHS method was developed in 1967 and has been reviewed recently by Merzhanov (1997) and Makino (2001).

In order to improve the efficiency of the SHS process, it is necessary to have both constituents in good mechanical contact. This permits the transfer of heat more readily from different parts of the powders during the SHS process. The rate of energy release in the SHS process can also be controlled by the size of the elemental powders, with smaller grains leading to higher surface to volume ratios and higher reaction rates by mass diffusion at the interfaces. It is thus desirable to reduce the size of the grains, up to micro or nanometers, and to arrange the powder in wafer-like layers (lamella), in order to improve the SHS process. Although other techniques like physical vapour deposition of thin films can achieve the same result (Rogachev and Mukasyan 2010), MA can be scaled up for production purposes. It is thus the most suitable technique for producing powders capable of sustaining SHS reactions with tunable rates.

The SHS process can also be performed using the same equipment as the MA if the reactions start during the milling process inside the milling vial. During the powder refinement process, the structure becomes so sensitive that a normal impact produces a

[2]SHS also stands for Super Hard Steel, a registered trademark of NanoSteel Co.

localized hot-spot that can ignite the SHS process, which then ignites all the powders in the vial. When this occurs, the process is known as a Mechanically-Induced Self-Propagating Reaction or MSR (Yen et al. 1996). However, the product of this process is highly porous, which is usually undesirable. Better properties are obtained if the material is first processed by MA just before the point of ignition and then the SHS is carried out in a different type of equipment. This variation was first proposed in 1995 for the synthesis of a noncrystalline compound of FeAl (Gaffet and Bernard 2002) and was given the name of Mechanically Activated Self-propagating High-temperature Synthesis (MASHS). The variation of the MA that stops the equipment before the point of ignition is called Arrested Reactive Milling (ARM). When the ARM is done using ball milling equipment the process is called Arrested Ball Milling (ABM). The start of the SHS is marked by an abrupt rise in the temperature due to the exothermic nature of the reaction. In Arrested Ball Milling, the time required to achieve the most possible refined structure without inducing ignition is a very important processing parameter. Ideally, this time to ignition can be correlated to the so-called milling dose, which is the amount of mechanical energy deposited in the powders during the refinement process.

Indeed, the most important measurement in the MA is the milling dose. The milling dose is the amount of mechanical work performed in the powder to transform its macro- and micro-structure. The milling dose is a fraction of the kinetic energy of the milling balls that is lost in every collision. It is a fraction because part of the energy is also dissipated in heat, noise, and vibration. Moreover, not every collision produces transformations of the powder, a minimum of energy is required to cross the elastic limit of deformations. If all other losses are neglected, the milling dose is the difference in the kinetic energy of the milling balls before and after every collision, in the case of ball-ball collisions. For the case of ball-vial collisions the calculation is more complicated. The term Specific Milling Dose is defined as milling dose per unit mass of powder. This term is more relevant in studies of scaling and optimization.

The time that it takes to take the powder to a certain state can not currently be predicted due to the complex nature of the ball milling process. There is a transfer of energy from the equipment to the milling balls and from the milling balls to the powder. The rate of energy transfer is not always constant during the process, and this is due to the fact that the powder is being transformed, and with it, its mechanical properties. This variation and the large number of variables at play make it difficult to predict the process time with a mathematical model. It is difficult to model the actual physical and chemical mechanisms responsible for the formation of phases and structures, as the main processes are not well understood (Nematollahi et al. 2008). A simulation of such

phenomena would require a multi-scale approach, as solid-state inter-diffusion ultimately occurs at atomic scales, requiring molecular dynamic simulations. The current status of modelling of mechanical alloying from a small scale point of view can be found in Khina and Froes (1996) or Shirakawa et al. (2008). The combination of these phenomena with the actual simulation of the equipment might be a challenge for future researchers.

Nevertheless, a correlation between the amount of energy dissipated during a milling process, and the dissipation in each collision can be obtained by multi-particle simulations. Past investigations used such particle simulations to gain more insight between the energetics of the milling device (agitation speed and amplitude) and the energy dissipated in inelastic collisions between the balls. In all previous simulations (Ward et al. 2005, Reichardt and Wiechert 2007, Concas et al. 2006, Delogu 2008) the powder refinement was not addressed explicitly. Instead, the kinematics of the grinding balls was computed. This served to obtain a relation between the amount of energy dissipated per collision and the global energy input in the milling device. Ward et al. (2005), Concas et al. (2006), and Delogu (2008) used a discrete element particle method to integrate the dynamics of the grinding balls inside the SPEX 8000 vial. They obtained scaling laws linking the milling dose and the operation parameters of the mill. Nevertheless, the numerical method they used is a force based method, where the particles are evolved over small time steps until a collision is detected. Although such a method is suitable for systems where there are long range forces acting on the particles, such as real molecular dynamics, the very small time steps make the computations inefficient.

More recently, Reichardt and Wiechert (2007) demonstrated that the Event Driven Method (Alder and Wainwright 1959) can be applied to model the milling environment. This method is suitable for modelling systems where there are no forces acting between particles in free flight other than when they collide with each other and with the walls. This reduces the need for small time steps, since the collision time between different balls can be obtained analytically. Nevertheless, the authors (Reichardt and Wiechert 2007) did not consider either the rotation of the grinding balls in the analysis, or the energy dissipated via shearing collisions. Likewise, the analysis was only performed for a dry milling environment, without the presence of any powders. Such refinements are considered in the present study.

1.4 This study

The present study is a numerical implementation of the Event Driven Method, demonstrated as a useful tool by Reichardt and Wiechert (2007). The method is used to model

the Spex 8000 milling equipment. The intention was to combine the recent modelling techniques developed by Ward et al. (2005) into the Event Driven method, which is significantly more efficient numerically. The collision dynamics are computed using the classical equation of conservation of momentum along with the energy dissipation controlled by the coefficient of restitution (COR). This strategy follows that of Ward et al., where the normal COR is determined from experiments of balls collision with different powder layer thickness between them. The tangential COR is the one proposed by Walton and Braun (1986). Ward's normal COR is supported by experimental tests conducted by Huang et al. (1998).

Milling balls are modelled as rigid bodies that follow free-flight trajectories. The kinematics of the Spex 8000 is modelled under the roto-translational model (RTM) as proposed by Concas et al. (2006). Concas, in turn, used the good approximations of the vial motion of the Spex 8000 developed by Caravati et al. (1999). The RTM is a convenient way to find the kinematics of a point by the aggregation of single translations and rotations.

The time of ball-ball collision is obtained analytically. The time of ball-vial collision had to be determined numerically due to the complex kinematics of the Spex 8000. This is the same solution that Reichardt and Wiechert (2007) chose since an analytical approach is not possible for such collisions.

The program developed in the present thesis is an evolution of the source code developed by Poschel and Schwager (2005), which is freely available and described in chapter 3 of their monograph. Their original code was an implementation of the Event Driven Algorithm in two space dimensions and fixed domains. In the present thesis, the code was completely re-written to make it more flexible and expandable using an object-oriented framework. The code was also extended to 3D to permit realistic ball milling simulations. A moving wall capability was implemented, which permitted specification of the kinematics of the Spex 8000 vials, including both container translation and rotation. In order to analyze the 3D dynamics of balls and vial, a graphical visualization tool was implemented. The effects of gravity, rotation of the balls, and balls with non-uniform radii and masses are also features implemented in the present study.

In the developed program most of the process variables are user-changeable. It will be possible to run the program with different combinations of ball diameter, number of balls, COR, vial size, and vial kinematics. A simulation can be stopped once a prescribed milling dose has been reached, or it can be controlled by the time, number of collisions, or maximum ball speed. Many outputs such statistics or animations are available.

Finally, the program is written in C++ and delivered in its source code. It can run

in any operating system once it is compiled with a C++ compiler.

1.5 Organization

The remainder of this thesis is organized as follows: Chapter 2 provides a summary of the EDM technique. A brief comparison EDM and the force-based DEM is also included. Chapter 3 reports the implementation of the code. Chapter 4 presents a case study and also shows many of the capabilities of the program. Chapter 5 includes an optimization study and a comparison with the work of Ward et al. (2005). Appendix A includes derivations of the fundamental mathematical expressions used in the program. Appendix B describes the roto-translational model, which is a mechanism to translate between correlated coordinate systems. Appendix C is a quick reference to the X3D standard. This is the graphic format used to implement all geometry of the program. Appendix D includes instructions to use the software developed in this study.

Chapter 2

The Event-Driven Model (EDM)

2.1 Introduction to EDM

2.1.1 Modelling of the MA

Currently, the formation of thermodynamic phases and their associated molecular structures is a very difficult process to model. This is due to the fact that the physical and chemical mechanisms that contribute to this process are not very well understood (Nematollahi et al. 2008). Some models exist that attempt to describe the behaviour of a substance at the molecular level. This type of modelling approach is known as Molecular Dynamics. Molecular Dynamics modelling is able to emulate some of the phenomena that occur in alloying, such as solid-state interdiffusion, hydrogen absorption, and ion irradiation. Unfortunately, these molecular level processes are currently not taken into account in milling device simulations (Khina and Froes 1996 Shirakawa et al. 2008). To date, this remains a challenge for many researchers. To overcome this challenge, it is possible to model the milling process easily by neglecting powder transformations. This type of simulation is commonly known as "global mechanistic modelling". EDM falls into this classification. Another global mechanistic model, known as the "Discrete Element Method" (DEM) is the most common type of simulation for Ball Milling that is available. EDM and DEM models are both derived from molecular dynamics physics.

The DEM is more popular than EDM due to the limitations of EDM. A brief comparison between EDM and DEM is given later in this chapter. Comparison of EDM to any other method is beyond the scope of this thesis, however, a short introduction to the classification of the modelling, along with a description of some others models, can be found in Khina and Froes (1996). A compilation of various algorithms for granular dynamics is found in Poschel and Schwager (2005). In this last work, EDM, DEM,

and several other models are explained in detail. The program developed in this study departs from the code found in this source.

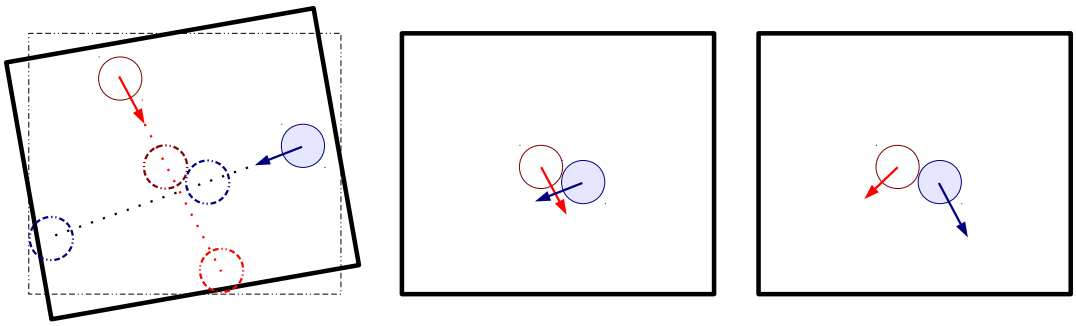
2.1.2 EDM in general

Three words can be used to summarize steps involved in the EDM algorithm: prediction, propagation, and collision. In the first step, prediction, the collision times of all of the particles are computed. Next, in the propagation step, each particle advances to its next position based on the time of when the earliest collision occurs. Finally, in the collision step, the velocities of the colliding particles are updated. The cycle is then repeated until certain conditions are satisfied. Figure 2.1 illustrates the concept. A flowchart showing the sequence of logical steps of the EDM algorithm is shown in figure 2.2. The EDM algorithm can be summarized as follows:

- Step 1: Initialization. The particles are randomly located in the computational domain. A list of collision times of all particles is created and maintained throughout the execution of the program. This avoids the need to predict the collision times after each collision occurs, making the algorithm efficient.
- Step 2. Prediction. The collision times for particles that have recently collided are re-calculated. The list of collision times is updated for those particles only, while all others collision times remain the same.
- Step 3: All particles are relocated to a new position. Particles move according to their current velocity for the next scheduled collision time. At this new location, two particles or a particle and a wall are in contact.
- Step 4: A collision occurs between the particles in contact. The new velocities of the colliding particles are computed using Newtonian mechanics.
- Step 5. Steps 2 to 5 are repeated until a stop condition is satisfied.
- Step 6. Outputs of the results are produced.

2.1.3 Comparison between EDM and DEM

Most of the simulations found in the literature use an alternate model known as the Discrete Element Method (DEM). Both the EDM and DEM models have been derived from molecular dynamic behaviours of gases. The analogy here is that the behaviour



(a) Prediction (b) Propagation (c) Collision

Figure 2.1: Main cycle of EDM.

of milling balls resemble the random motion of molecules colliding a gas. In particular, DEM simulates the interaction of particles as a viscoelastic system, where the forces are known and related to the instantaneous state of the system. Newton's equations of motion are numerically integrated in time to obtain the new particles' positions and velocities, which are in turn used to compute the magnitude of the forces acting on the molecules. The cycle is continuously repeated in small discrete time steps. In the milling process, the mechanical work performed on the powder is computed by integrating the force that acts on a particle as it travels a certain distance within a prescribed time period. With this kind of model it is possible to simulate multiple particle interactions at one time, i.e. when more than two particles in contact. EDM, on the other hand, simplifies the computation. Instead of integrating the forces acting on each particle at each time step, the trajectories are computed from one collision to another. This has a substantial gain in terms of computational efforts required to conduct the simulation as the computer saves a lot of unnecessary processing. The most significant limitation of EDM is that only one collision between a pair of particles can be computed at a time[1]. Furthermore, the prediction of the collision times of a milling ball with a wall is not easily solved with a set of equations. The expressions that describe the cyclical trajectory of the vial in the Ball Milling machines are generally non-linear and many of them, if not all, have transcendental functions. On the other hand, the trajectory of the vial has no relevance when using DEM since there is no prediction of collisions. Instead, a collision detection is performed, which is much simpler.

[1]For many reasons, several particles group together forming what is known as cluster.

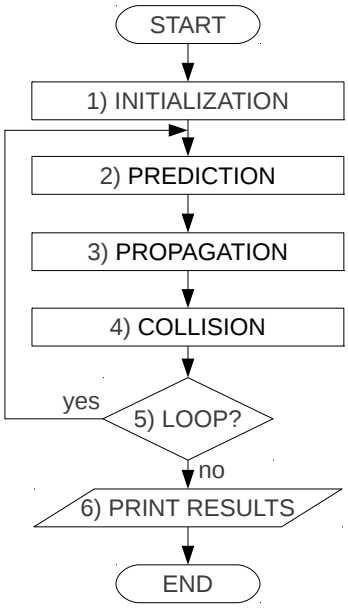


Figure 2.2: Flow diagram of EDM.

2.2 Propagation

2.2.1 Balls propagation

The milling balls are modelled as rigid bodies with mass that follows a free-flight trajectory. The only force acting on the balls is gravity. The milling balls are assumed to be perfect spheres, with their centre of mass coincident with their geometrical centre. It is assumed that the trajectory of the balls is not affected by their rotation. The trajectory of a ball is defined by:

$$\vec{r} = \vec{r_o} + \vec{v}_o \Delta t + \tfrac{1}{2}\vec{a}\Delta t^2 \tag{2.1}$$

where $\vec{r_o}$ is the initial position of the particle, $\vec{v}_o$ is the initial velocity, Δt is the change in time, and $\vec{a}$ is the acceleration due to gravity.

Derivation of physics is presented in A.2.

2.2.2 Vial propagation

Two models have been created in the past to simulate the Spex 8000: a simplified 2D model was created by Davis et al. (1988), and a more accurate 3D model was created by Caravati et al. (1999). Neither of these models allowed for an easy mathematical prediction of the ball-vial collision. An attempt to solve Caravati's model is presented

in section 2.3.3.1. Instead, a numerical method is used to find the collision time of this scenario. Caravati's model is preferred over Davis' because it is more realistic. In the current study, the large availability of computational resources allowed us to compromise calculation speed by improving accuracy. Caravati not only provides a set of equations that approximate the vial motion, but also gives a comparison with experimental measurements.

In this study, the kinematics of the vial is not implemented with the proposed set of equations but instead with the roto-translational model (RTM). The RTM allows an increase in the flexibility and readability of the code. The RTM is described in appendix B. The theory presented here is an adaptation of the work of Concas et al. (2006). The coordinate systems were rearranged in order to comply with the X3D standard. A brief description of this standard is given in appendix C. Two coordinate systems are used and they are known as the inertial and the non-inertial coordinate systems, according to the nomenclature of the appendix B. The inertial coordinate system is located at the eccentric fulcrum (a sketch of the device is found in figure 1.2) and is defined with the capital letters X, Y and Z. The non-inertial system is located at the geometrical centre of the vial as specified in appendix C.2.2 and is defined with the letters x, y and z in lower case. The axes are oriented in such a way that the front view in the X3D model corresponds to the front view of the machine. Concas applies the right-hand rule in one coordinate system and the left-hand rule in the other one. In this study, the right hand rule is used to comply with the X3D standard (section C.1). The symbol α for the second angle is replaced by -γ to emphasize this distinction. The re-arranged coordinate systems are depicted in figure 2.3. It should be noted that as a result of the re-arrangement the equations presented here do not match with those provided by Concas.

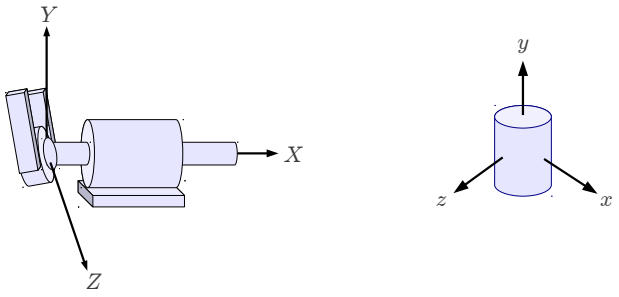
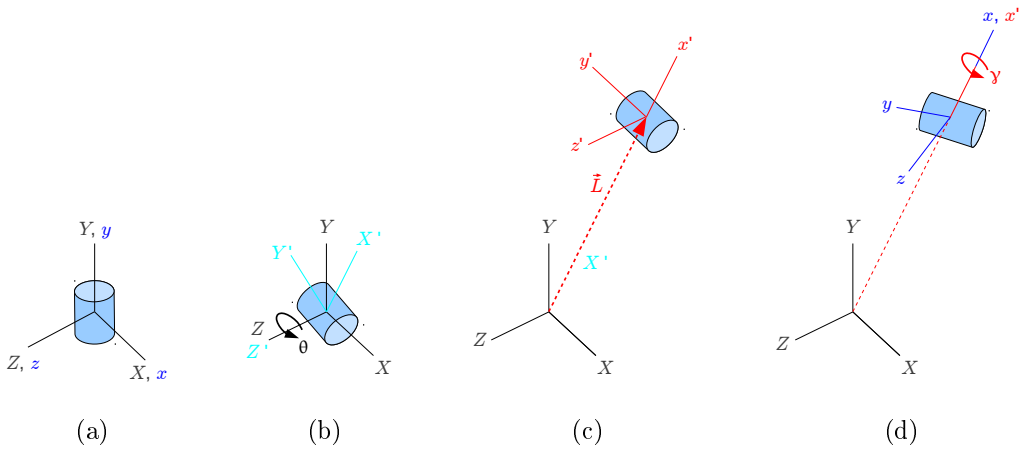


Figure 2.3: Location and orientation of both the inertial and non-inertial coordinate systems of the vial in the Spex 8000.

The vial motion is modelled with a series of nested transformations. The transformations take the vial from its initial position to its final position in three steps as illustrated

in figure 2.4. A useful analogy for understanding the vial motion is to imagine a person's forearm rotating about the elbow, with the fist acting as a vial that rotates about the wrist.



(a) (b) (c) (d)

Figure 2.4: Model of the vial motion for the Spex 8000.
a) Initial position, b) Rotation around Z, c) Translation on X', d) Rotation around x'

The vial is initially placed in such way that the two coordinate systems are coincident, as shown in figure 2.4a. The first transformation is a rotation about Z by angle θ, as shown in figure 2.4b. The corresponding rotation matrix is $\bar{\Omega}_Z(\theta)$. The angle θ as a function of time, t, is as follows

$$\theta = \theta_m \cos(w \cdot t) + \pi/2 \tag{2.2}$$

where θ_m is the maximum amplitude of this angle, and w is the rotational speed of the driving shaft. The new coordinate system is identified with X', Y' and Z'. The inertial location of a point, given by the new coordinates, is

$$\vec{P}_1(X, Y, Z) = \left[\bar{\Omega}_Z(\theta) \cdot \vec{p}(X', Y', Z')^T\right]^T$$

The second transformation is a translation of the distance l on the X' axis, as shown in figure 2.4c. The corresponding vector is

$$\vec{L}(X', Y', Z') = (l, 0, 0) \tag{2.3}$$

The non-inertial system is now located at C,

$$\vec{C}(X, Y, Z) = \left[\bar{\Omega}_Z(\theta) \cdot \vec{L}(X', Y', Z')^T\right]^T \tag{2.4}$$

The next translation generates the coordinate system identified with x', y', z'. The location of a point is then given by

$$\vec{P}_2(X,Y,Z) = \vec{C}(X,Y,Z) + \left[\bar{\Omega}_Z(\theta) \cdot \vec{p}(x',y',z')^T\right]^T$$

The previous transformation is a rotation about x' by angle γ, as shown in figure 2.4d. The corresponding rotation matrix is $\bar{\Omega}_{x'}(\gamma)$. The angle γ as a function of time, t, is given by

$$\gamma = \gamma_m \sin(w \cdot t) \tag{2.5}$$

where γ_m is the maximum amplitude of this angle, and w is the rotational speed of the driving shaft. The resulting rotation matrix is

$$\bar{\Omega} = \bar{\Omega}_{x'}(\gamma) \cdot \bar{\Omega}_Z(\theta) \tag{2.6}$$

The resulting coordinate system is defined by x, y, z and becomes the non-inertial system. The change of coordinates between systems is computed with the following equations:

$$\vec{P}(X,Y,Z) = \vec{C}(X,Y,Z) + \left[\bar{\Omega} \cdot \vec{p}(x,y,z)^T\right]^T \tag{2.7}$$

$$\vec{p}(x,y,z) = [\vec{P}(X,Y,Z) - \vec{C}(X,Y,Z)] \cdot \bar{\Omega} \tag{2.8}$$

The velocity is different for every point of space; therefore, the location of the contact point between particles must be resolved for every collision before the new velocity can be computed. The velocity of a point is computed as the sum of its relative velocities. The angular velocities around Z and x' are respectively:

$$\vec{W}_Z(\dot{\theta}) = (0,0,\dot{\theta})$$

$$\vec{W}_{x'}(\dot{\gamma}) = (\dot{\gamma},0,0)$$

where $\dot{\theta}$ and $\dot{\gamma}$ are the respective angular speeds and can be found by taking the time derivatives of θ and γ. The linear velocity of a point that rotates, by definition, is the cross product of the angular velocity vector and the vector that goes from the centre of rotation to the point of interest. The linear velocities that result from the previous rotations of the inertial system are respectively given by:

$$\begin{aligned} \vec{V}_\theta &= \vec{W}_Z \times \vec{P} \\ \vec{V}_\gamma &= \left[\bar{\Omega}_Z \cdot \vec{W}_{x'}^T\right]^T \times (\vec{P} - \vec{C}) \end{aligned}$$

The translation of the vial over the X' axis over a distance l do not contribute to the velocity of any point since this translation does not change in time. Therefore, the linear velocity of a point is only due to the two rotations, and is therefore expressed as:

$$\vec{V} = \vec{W}_Z \times \vec{P} + \left[\bar{\Omega}_Z \cdot \vec{W}_{x'}^T\right]^T \times (\vec{P} - \vec{C})$$

Finally, the dimensional values of the Spex 8000 are:

$$\begin{aligned} l &= 10\,\text{cm} \\ \gamma_m &= 15\,^\circ \\ \theta_m &= 15\,^\circ \\ w &= 1098\,\text{rpm} \end{aligned} \tag{2.9}$$

2.3 Prediction

A list of the future collision times of all of the particles is maintained throughout the execution of the program. Maintaining this list avoids the need to calculate all of the collision times repeatedly after each collision occurs. When a collision occurs, only the updated collision times for the colliding particles is calculated. All other collision times for the non-colliding particles remains the same.

2.3.1 Lists of collisions

Two lists of forecasted collision times is maintained to avoid the repetition of collision computations after every collision. Only the records of the colliding particles are updated, while the records of all other particles remain the same. The first list stores the time of all forecasted collisions in ascending order, including their corresponding particle. The second list stores the list of all forecasted collision times for each individual particle, or milling ball. Figure 2.5 shows a representation of those lists filled with sample data.

The list is updated with the following procedure:

- The times belonging to ball i are selected from the second list (on the right of figure 2.5) and deleted in the first list (on the left of figure 2.5). The same operation is carried out for particle j if it is a ball.

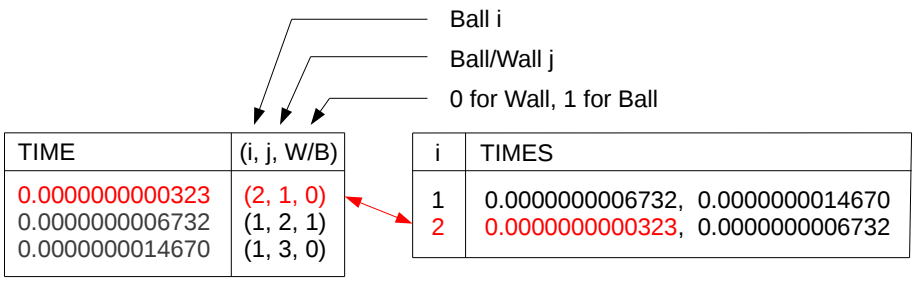


Figure 2.5: A sample of the times lists of forecasted collisions.

- The times are deleted from the second list for the balls i and j (if applicable).
- The time of collision between each of the colliding balls and all other balls and the vial is computed and stored in both lists.

2.3.2 Ball-ball collision prediction

The collision times are computed in agreement with the trajectory of the balls, given in equation 2.1. A collision occurs when both particles are in contact with one another. This happens when the distance between their centres becomes equal to the sum of their radii, i.e.:

$$\mid (\vec{r_i} + \vec{v}_i\Delta t + \tfrac{1}{2}\vec{a}\Delta t^2) - (\vec{r_j} + \vec{v}_j\Delta t + \tfrac{1}{2}\vec{a}\Delta t^2) \mid = R_i + R_j \tag{2.10}$$

where R_i and R_j are the radii of the balls.

The solution to this equation is presented in appendix A.3 and shown here

$$\Delta t = \frac{-\vec{r}_{ij} \cdot \vec{v}_{ij} \pm \sqrt{(\vec{r}_{ij} \cdot \vec{v}_{ij})^2 - (\vec{r}_{ij}^2 - R_{ij}^2)\vec{v}_{ij}^2}}{\vec{v}_{ij}^2} \tag{2.11}$$

where

$$\begin{aligned} \vec{r}_{ij} &\equiv \vec{r}_i - \vec{r}_j \\ \vec{v}_{ij} &\equiv \vec{v}_i - \vec{v}_j \\ R_{ij} &\equiv R_i + R_j \end{aligned}$$

In the solution for the collision time, the lower positive time is used as the correct one. It should be noted that the collision time is independent of gravity.

No simultaneous collisions are allowed under the EDM; however, this may happen in reality. To overcome this limitation, one of the forecasted times is shifted by a very small amount. This is a practical solution that does not really affects the accuracy of the program.

2.3.3 Ball-vial collision prediction

The collision time of a milling ball against the vial cannot be solved analytically due to the complexity of the equations, as shown in the next section. Unfortunately, a conventional numerical solution is also impractical, as the method adds an overhead cost to the simulation. This issue is discussed in the conclusion of this paper. With no alternative, the solution can only be obtained by a step-wise method. With this approach, the ball and the vial move in small time steps until they reach each other. Increased accuracy is achieved iteratively with smaller time steps.

2.3.3.1 An attempt to solve the ball-wall collision prediction analytically

The vial motion equations are taken from section 2.2.2. The geometric properties of the vial are specified according to the X3D standard and are described in appendix C.2.2.

Expanding the RTM equations The following equations are the symbolic representation of the rotational matrices from section B:

$$\bar{\Omega}_Z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

$$\bar{\Omega}_{x'}(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \tag{2.13}$$

Multiplying last two matrices as defined by equation 2.6 leads to

$$\bar{\Omega} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta\cdot\cos\gamma & \cos\theta\cdot\cos\gamma & -\sin\gamma \\ \sin\theta\cdot\sin\gamma & \cos\theta\cdot\sin\gamma & \cos\gamma \end{bmatrix} \tag{2.14}$$

Introducing matrix 2.12 and vector 2.3 into equation 2.4

$$\vec{C}(X,Y,Z) = (l\cos\theta, l\sin\theta, 0) \tag{2.15}$$

The equations of ball-wall collision are simpler in the non-inertial system of coordinates. The relative coordinates of a point using equations 2.8, 2.14, and 2.15 are:

$$\begin{aligned} x &= X\cos\theta + Y\sin\theta\cos\gamma + Z\sin\theta\sin\gamma - l(\sin^2\theta\cos\gamma + \cos^2\theta) \\ y &= -X\sin\theta + Y\cos\theta\cos\gamma + Z\cos\theta\sin\gamma + l\sin\theta\cos\theta(1-\cos\gamma) \\ z &= -Y\sin\gamma + Z\cos\gamma + l\sin\theta\sin\gamma \end{aligned} \tag{2.16}$$

Collision with the vial The collision time of a ball against a vial is found by computing the collision time of the ball with each one of the walls and then choosing the lowest collision time. This approach allows one to consider each wall as having an infinite area, which simplifies the computations significantly.

Collision with the top wall Figure 2.6 is an extension of figure 2.4 and shows the location of the top wall. The non-inertial coordinate system is located at the geometrical centre of the vial. The top wall is located at half of the height of the vial in the y-coordinate.

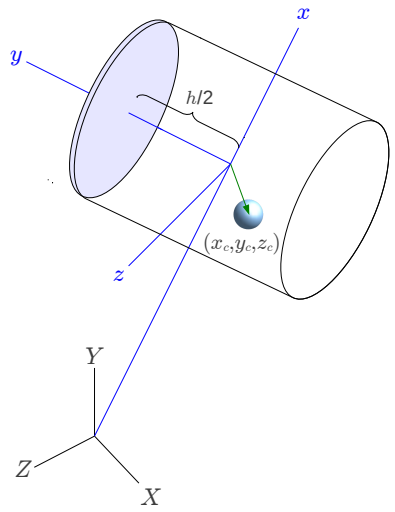


Figure 2.6: Non-inertial position of the top face of the vial in the Spex 8000.

The distance between the surface of a ball and the top wall is given by

$$d_t = h/2 - (y_c + R) \tag{2.17}$$

where y_c is the y-coordinate of the ball milling centre, R is its radius, and h is the height of the vial.

There is collision when the distance between the ball and the wall reduces to zero. This happens when the outermost y-coordinate of a milling ball is equal to half the height of the vial, i.e.

$$h/2 - y_c - R = 0$$

The last equation can be translated into the inertial coordinate system with the help of equation (2.16):

$$X_c \sin\theta - Y_c \cos\theta \cos\gamma - Z_c \cos\theta \sin\gamma - l \sin\theta \cos\theta (1 - \cos\gamma) - R + h/2 = 0$$

where X_c, Y_c, Z_c are inertial coordinates of the ball centre.

Equations A.15, 2.2, and 2.5 are introduced into the last equation to find the time of collision.

$$\begin{aligned}
&(X_o + V x_o \Delta t + \tfrac{1}{2} a_x \Delta t^2) \sin\!\Big(\theta_m \cos(w(t_o + \Delta t)) + \tfrac{\pi}{2}\Big) \\
&-(Y_o + V y_o \Delta t + \tfrac{1}{2} a_y \Delta t^2) \cos\!\Big(\theta_m \cos(w(t_o + \Delta t)) + \tfrac{\pi}{2}\Big) \cos(\gamma_m \sin(w(t_o + \Delta t))) \\
&-(Z_o + V z_o \Delta t + \tfrac{1}{2} a_z \Delta t^2) \cos\!\Big(\theta_m \cos(w(t_o + \Delta t)) + \tfrac{\pi}{2}\Big) \sin(\gamma_m \sin(w(t_o + \Delta t))) \\
&-l \sin\!\Big(\theta_m \cos(w(t_o + \Delta t)) + \tfrac{\pi}{2}\Big) \cos\!\Big(\theta_m \cos(w(t_o + \Delta t)) + \tfrac{\pi}{2}\Big) [1 - \cos(\gamma_m \sin(w(t_o + \Delta t)))] \\
&-R + \tfrac{h}{2} = 0
\end{aligned} \tag{2.18}$$

where the subscript o stands for the condition at the current time, Δt is the collision time and is the only unknown.

This previous equation is difficult to solve analytically. To simplify the equation, it is assumed that gravity, g, acts only in the negative direction of Y , and also, it is assumed that both maximum angles are equal, as stated in equation 2.9, i.e. $\theta_m = \gamma_m$. However, despite these simplifications, the equation is still difficult to solve analytically.

Figure 2.7 shows a plot of the last equation with some sample values.

Collision with the bottom wall The collision of a milling ball with the bottom wall is similar to its collision with the top wall. Figure 2.8 shows the location of the bottom wall.

The distance between the surface of a ball and the bottom wall is given by

$$d_b = y_c - R - h/2 \tag{2.19}$$

An equation, which is difficult to solve, is also obtained.

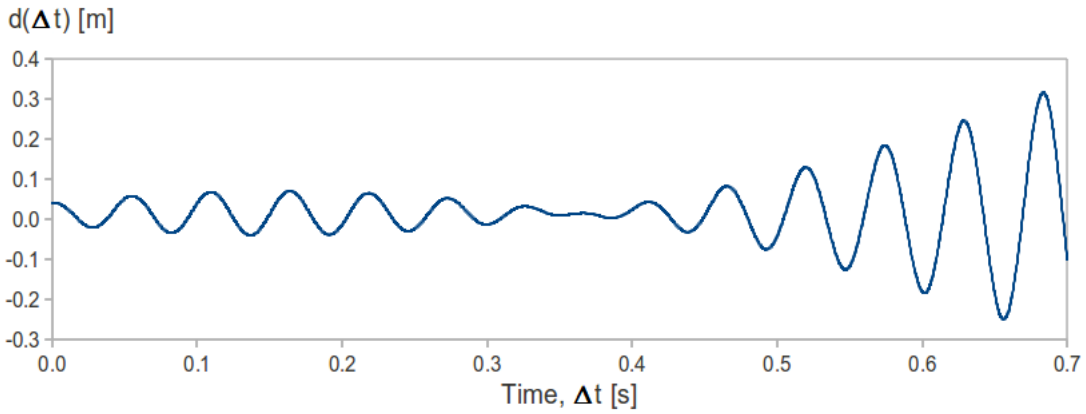


Figure 2.7: Plot of the distance between a ball and the top wall in the Spex 8000. This is a plot of the left part of the equation 2.18 with sample data. A ball of radius 0.012 m is initially placed 0.1 m on the Y-axis, with a velocity of 1.5 $^{\mathrm{m}}/_{\mathrm{s}}$, towards $+Y$. The height of the cylinder is 0.0572 m. Gravity is defined as usual (9.8 $^{\mathrm{m}}/_{\mathrm{s}^2}$ towards the negavite direccion of Y). The wall is considered borderless.

$$\begin{aligned}
&-(X_o+Vx_o\Delta t+\tfrac{1}{2}a_x\Delta t^2)\sin\!\left(\theta_m\cos(w(t_o+\Delta t))+\tfrac{\pi}{2}\right)\\
&+(Y_o+Vy_o\Delta t+\tfrac{1}{2}a_y\Delta t^2)\cos\!\left(\theta_m\cos(w(t_o+\Delta t))+\tfrac{\pi}{2}\right)\cos(\gamma_m\sin(w(t_o+\Delta t)))\\
&+(Z_o+Vz_o\Delta t+\tfrac{1}{2}a_z\Delta t^2)\cos\!\left(\theta_m\cos(w(t_o+\Delta t))+\tfrac{\pi}{2}\right)\sin(\gamma_m\sin(w(t_o+\Delta t)))\\
&+l\sin\!\left(\theta_m\cos(w(t_o+\Delta t))+\tfrac{\pi}{2}\right)\cos\!\left(\theta_m\cos(w(t_o+\Delta t))+\tfrac{\pi}{2}\right)[1-\cos(\gamma_m\sin(w(t_o+\Delta t)))]\\
&-R-\tfrac{h}{2}=0
\end{aligned} \tag{2.20}$$

Collision with the cylindrical wall The collision of a milling ball with the cylindrical wall can be obtained with the same methodology as applied for a collision with the top wall. Figure 2.9 shows a sectional cut of the vial.

The distance between the surface of a ball and the cylindrical wall is given by

$$d_c = R_v - R - \sqrt{x_c^2 + z_c^2} \tag{2.21}$$

where x_c and z_c are the x- and y-coordinates, respectively, of the ball milling centre, R is its radius, and R_v is the vial radius.

There is a collision when the last distance is equal to zero, i.e.

$$R_v - R - \sqrt{x_c^2 + z_c^2} = 0$$

The previous equation can be translated into the inertial coordinate system with the help of (2.16):

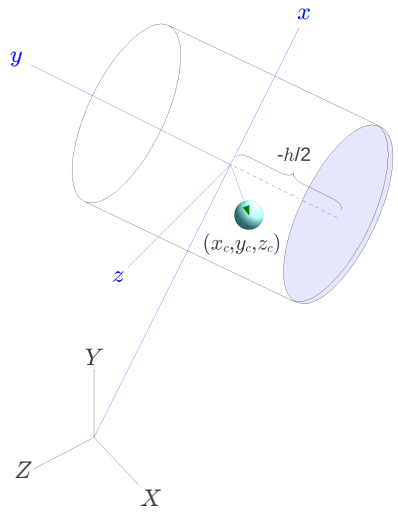


Figure 2.8: Non-inertial position of the bottom face of the vial in the Spex 8000.

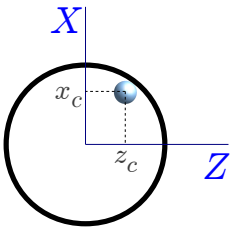


Figure 2.9: Non-inertial coordinates of the centre of a ball in contact with the cylindrical wall in the Spex 8000.

$$(X_c\cos\theta + Y_c\sin\theta\cos\gamma + Z_c\sin\theta\sin\gamma - l\cos\gamma)^2 + (-Y_c\sin\gamma + Z_c\cos\gamma + l\sin\theta\sin\gamma)^2 - (R_v - R)^2 = 0$$

where X_c, Y_c and Z_c are inertial coordinates of the ball centre.

The previous equation is written at the current time. Equations A.15, 2.2, and 2.5 are introduced into the previous equation to find the time of collision.

$$
\begin{aligned}
&\Big[(X_o+Vx_o\Delta t+\tfrac{1}{2}a_x\Delta t^2)\cos\big(\theta_m\cos(w(t_o+\Delta t))+\tfrac{\pi}{2}\big)\\
&+(Y_o+Vy_o\Delta t+\tfrac{1}{2}a\Delta t^2)\sin\big(\theta_m\cos(w(t_o+\Delta t))+\tfrac{\pi}{2}\big)\cos(\gamma_m\sin(w(t_o+\Delta t)))\\
&+(Z_o+Vz_o\Delta t+\tfrac{1}{2}a_z\Delta t^2)\sin\big(\theta_m\cos(w(t_o+\Delta t))+\tfrac{\pi}{2}\big)\sin(\gamma_m\sin(w(t_o+\Delta t)))\\
&-l\cos(\gamma_m\sin(w(t_o+\Delta t)))\Big]^2\\
&+\big[-(Y_o+Vy_o\Delta t+\tfrac{1}{2}a_y\Delta t^2)\sin(\gamma_m\sin(w(t_o+\Delta t)))\\
&+(Z_o+Vz_o\Delta t+\tfrac{1}{2}a_z\Delta t^2)\sin(\gamma_m\cos(w(t_o+\Delta t)))\\
&+l\sin\big(\theta_m\cos(w(t_o+\Delta t))+\tfrac{\pi}{2}\big)\sin(\gamma_m\sin(w(t_o+\Delta t)))\big]^2\\
&-\big[R_v-R\big]^2=0
\end{aligned}
\tag{2.22}
$$

where the subscript o stands for the condition at current time, Δt is the collision time and is also the only unknown variable. This last equation is even more difficult to solve analytically than the one for collision with the top wall (Equation 2.20).

2.3.3.2 Numerical solution

Most of the equations above are solved by using root-finding algorithms. The collision time in each case corresponds to the lowest positive root. The main drawback of conventional root-finding methods is that they require an initial guess of the root or, alternatively, a bounded region must be known. This initial guess or boundary condition is difficult to find analytically, for the same reasons that make the equations above difficult to solve. A large region can be provided; however, if the region contains more than one root the leftmost one might not be returned. On the other hand, it may happen that a ball will never reach a specific wall and thus, no root will exist. This same difficulty has been faced by Reichardt and Wiechert (2007) in their simulation of an attritor with EDM. They implemented a step-wise algorithm to find a bounded region and then used Newton's method to find the first root. A similar solution is developed here. A step-wise algorithm is also used to find the bounded region. The ball and vial are moved in small time steps until they collide, and this is equivalent to what DEM does. The distance is evaluated simultaneously for all walls to reduce computer time. The burden of implementing equations 2.18, 2.20, and 2.22 is saved by using equations 2.17, 2.19, and 2.21 instead. It was noticed that with very few changes, the step-wise algorithm used to find the bounded region could also be used to find the first root. The advantage of this approach is that not only the code is more compact, but also it is free of the other limitations of conventional numeric methods (not mentioned here). Conventional methods evaluate one wall at a time whereas the step-wise method evaluates all walls at the same time. In this study, the first root is found by approximation. Once the root is

found, the algorithm starts again from the last time with a smaller time step. This is an iterative process that stops when the desired precision is reached.

The condition that the ball and wall become in contact when the distance between them is zero is hard to meet using time steps. The collision condition was redefined for this purpose. Now, it is said that there is contact when the distance between them is lower than a certain small amount. No simultaneous collisions are allowed under the EDM. To overcome this limitation, the same procedure as the ball-ball collision is followed. One of the forecasted times is shifted by a very small amount. This is a practical solution that does not really affects the accuracy of the program.

2.4 Collision

Collision response is derived from Newtonian Mechanics. The resulting equations are the product of combining the equations of conservation of momentum with the equations of the COR. The point of contact is considered the centre of rotation for every collision. None of the forces acting on the particles causes a torque transmission since the contact area and collision time are assumed to be infinitesimally small. Consequently, there is no angular momentum transfer between the balls or in other words, the angular momentum relative to the contact point is conserved for each particle separately. Colliding particles recover their normal shape after every iteration; however, part of their energy is dissipated in the process.

In this work it is assumed that all dissipation of energy corresponds to the mechanical work performed to transform the structure of the powder. This amount is known as the "milling dose" and is computed for every collision. All other losses are neglected.

2.4.1 Coefficient of restitution (COR)

The COR is an old way to compute energy dissipation by measuring velocity. The COR is the ratio of relative velocities of two bodies before and after the impact, and it is expressed as:

$$\varepsilon = \frac{|\vec{g}'|}{|\vec{g}|}$$

where ε is the COR, $\vec{g}$ and $\vec{g}'$ are the relative velocity of the contact point before and after the collision, respectively.

A coefficient of 1 means no losses and it is said to be elastic; otherwise, there are losses and the collision is said to be inelastic. For practical reasons, it is better to work

with the normal and tangential components of the coefficient of restitution than with the formal coefficient. This leads to the following expressions:

$$\begin{aligned} (\vec{g}^n)' &= -\varepsilon^n \vec{g}^n \qquad with \quad 0 \leq \varepsilon^n \leq 1 \\ (\vec{g}^t)' &= \varepsilon^t \vec{g}^t \qquad with \quad -1 \leq \varepsilon^t \leq 1 \end{aligned} \tag{2.23}$$

where the superscripts n and t differentiate the normal and tangential components respectively.

The last set of equations can be used instead of the equation of conservation of energy. This is advantageous since the equations of the COR are of first degree, whereas the equation of conservation of energy is of second degree. Energy losses, like vibrations, heat release or noise, are all lumped in the COR.

The bad news about the COR is that it is extremely hard to find analytically in MA applications. The COR depends on many variables such as impact velocity, ball diameter and the thickness of the powder layer (Huang et al. 1998). This thickness is proportional to the ratio of total surface of the balls to the powder volume. The amount of powder in the contact area also changes from collision to collision and is hard to predict. Also, the properties of the powder change over time, as the initial material is transformed into a new alloy.

Literature about the effect of the powder in the COR is scarce and mostly limited to perpendicular ball-plate collisions. Experiments presented in Huang et al. (1998) show that there is a huge difference in the normal COR when powder is involved. This fact makes the existing of literature about the COR in powder-free collisions rather useless. Unfortunately, the values of COR provided in Huang et al. (1998) are not useful here since the tested balls are larger than the ones used in the Spex 8000. The smaller ball diameter tested by Huang is 26 mm, which is more than double for applications of the Spex, where the typical vial size is around 50 mm in both diameter and height. Until now, the only source of experimental values of the normal COR suitable for the Spex 8000 is Ward et al. (2005).

A model of the tangential COR in collisions involving powder is non-existent. Many studies of collisions simply use a tangential COR equal to 1, assuming that particles are "ideally smooth". But this is not the case for MA where a powder layer surrounds the milling balls. The tangential COR used here comes from the Incrementally Slipping Friction Model proposed by Walton and Braun (1986). This model may not be accurate in the presence of powder, but at least it provides an excellent starting point. This model was chosen to have comparable values with those used by Ward. The aforementioned

tangential COR is defined for collisions between equal sized particles by[2]:

$$\varepsilon^t = 1 - \mu(1+\varepsilon^n)\left|\vec{g}^n/\vec{g}^t\right|(1+mR^2/J)$$

where μ is the coefficient of Coulomb's friction, m, R, and J are the mass, radius, and moment of inertia of the particles.

The value of the tangential COR can be lower than -1 when $\vec{g}^t$ is near zero. Ward does not mention any limit; therefore, -1 is assumed to comply with equation 2.23. With this limit and the definition of the reduced moment of inertia, $\tilde{J} = J/mR^2$, the last equation can be reduced to:

$$\varepsilon^t = max\left(-1, 1-\mu(1+1/\tilde{J})(1+\varepsilon^n)\left|\vec{g}^n/\vec{g}^t\right|\right) \tag{2.24}$$

2.4.2 Ball-ball collision

The collision response involves the COR and the following equation of conservation of linear momentum and both equations of conservation of angular momentum evaluated at the contact point, i.e.:

$$\begin{aligned} m_i\vec{v}_i' + m_j\vec{v}_j' &= m_i\vec{v}_i + m_j\vec{v}_j \\ R_i\vec{e}_{ij}^n \times m_i\vec{v}_i' + J_i\vec{\omega}_i' &= R_i\vec{e}_{ij}^n \times m_i\vec{v}_i + J_i\vec{\omega}_i \\ R_j\vec{e}_{ij}^n \times m_j\vec{v}_j' - J_j\vec{\omega}_j' &= R_j\vec{e}_{ij}^n \times m_j\vec{v}_j - J_j\vec{\omega}_j \end{aligned} \tag{2.25}$$

where

The prime stands for the variable after collision

m_i and m_j are the masses of the balls

R_i and R_j are the radius of the balls

J_i and J_j are the moments of inertia

$\vec{v}_i$ and $\vec{v}_j$ are the translational velocities of the balls

$\vec{w}_i$ and $\vec{w}_j$ are the rotational velocities of the balls

$\vec{e}_{ij}^n$ is defined as

$$\vec{e}_{ij}^n = \frac{\vec{r}_i - \vec{r}_j}{|\vec{r}_i - \vec{r}_j|} \tag{2.26}$$

where $\vec{r}_i$ and $\vec{r}_j$ are the vectors from the origin of coordinates to the centre of each ball, respectively, $\vec{e}_{ij}^n$ is a unit vector pointing from the centre or particle j *to* the centre of

[2] ε^t is the equivalent of $-\beta$ in Walton

the particle i, and is known as the normal vector.

The relative velocity of colliding particles at the point of contact before and after every collision, $\vec{g}_{ij}$ and $\vec{g}'_{ij}$, is determined by the translational and rotational velocities of the balls:

$$\begin{aligned}\vec{g}_{ij} &= (\vec{v}_i - \vec{\omega_i} \times R_i\vec{e}^n_{ij}) - (\vec{v}_j + \vec{\omega}_j \times R_j\vec{e}^n_{ij}) \\ \vec{g}'_{ij} &= (\vec{v}'_i - \vec{\omega_i}' \times R_i\vec{e}^n_{ij}) - (\vec{v}'_j + \vec{\omega}'_j \times R_j\vec{e}^n_{ij})\end{aligned} \tag{2.27}$$

The velocities of the balls after collision are obtained by combining equations 2.23, 2.25, and 2.27. The derivation is presented in the appendix A.4 and the result presented here.

$$\begin{aligned}\vec{v}'_i &= \vec{v}_i - \frac{M_{ij}}{m_i}[(1+\varepsilon^n)\vec{g}^n_{ij} + \frac{1-\varepsilon^t}{1+\frac{M_{ij}}{m_i\tilde{J}_i}+\frac{M_{ij}}{m_j\tilde{J}_j}}\vec{g}^t_{ij}] \\ \vec{v}'_j &= \vec{v}_j + \frac{M_{ij}}{m_j}[(1+\varepsilon^n)\vec{g}^n_{ij} + \frac{1-\varepsilon^t}{1+\frac{M_{ij}}{m_i\tilde{J}_i}+\frac{M_{ij}}{m_j\tilde{J}_j}}\vec{g}^t_{ij}] \\ \vec{\omega}'_i &= \vec{\omega}_i + \frac{M_{ij}}{m_iR_i\tilde{J}_i}(\frac{1-\varepsilon^t}{1+\frac{M_{ij}}{m_i\tilde{J}_i}+\frac{M_{ij}}{m_j\tilde{J}_j}})(\vec{e}^n_{ij} \times \vec{g}^t_{ij}) \\ \vec{\omega}'_j &= \vec{\omega}_j + \frac{M_{ij}}{m_jR_j\tilde{J}_j}(\frac{1-\varepsilon^t}{1+\frac{M_{ij}}{m_i\tilde{J}_i}+\frac{M_{ij}}{m_j\tilde{J}_j}})(\vec{e}^n_{ij} \times \vec{g}^t_{ij})\end{aligned} \tag{2.28}$$

where M is the equivalent mass defined by,

$$M_{ij} \equiv \frac{m_im_j}{m_i+m_j} \tag{2.29}$$

and $\tilde{J}$ is the reduced moment of inertia,

$$\begin{aligned}\tilde{J}_i &\equiv J_i/m_iR_i^2 \\ \tilde{J}_j &\equiv J_j/m_jR_j^2\end{aligned}$$

The milling dose for the ball-ball collisions is derived in appendix A.57 and presented here:

$$D_{ij} = \tfrac{1}{2}M_{ij}\{[1-(\varepsilon^n)^2](\vec{g}^n_{ij})^2 + \frac{1-(\varepsilon^t)^2}{1+\frac{M_{ij}}{m_i\tilde{J}_i}+\frac{M_{ij}}{m_j\tilde{J}_j}}(\vec{g}^t_{ij})^2\} \tag{2.30}$$

2.4.3 Ball-wall collision

The vial and the the milling balls are simulated as colliding rigid bodies. The ball-wall collision rules can be deducted using the same logic and assumptions as for the ball-ball collision. The collision response involves the COR and the following equation of conservation of angular momentum evaluated at the contact point, i.e.:

$$R_i\vec{e}^{\,n}_{iw} \times m_i\vec{v}_i' + J_i\vec{\omega}_i' = R_i\vec{e}^{\,n}_{iw} \times m_i\vec{v}_i + J_i\vec{\omega}_i \tag{2.31}$$

where

The prime stands for the variable after collision.

m_i is the mass of the ball

R_i is the radius of the ball

J_i is the moment of inertia of the ball

$\vec{v}_i$ is the translational velocity of the ball

$\vec{w}_i$ is the rotational velocity of the ball

$\vec{e}^{\,n}_{ij}$ is defined as

$$\vec{e}^{\,n}_{iw} = \frac{\vec{r}_i - \vec{r}_w}{|\vec{r}_i - \vec{r}_w|} \tag{2.32}$$

where $\vec{r}_i$ and $\vec{r}_w$ are the distance from the origin of coordinates to the centre of the particle and to the contact point respectively, $\vec{e}^{\,n}_{iw}$ is the normal vector and is a unit vector located at the contact point and pointing to the centre of the particle i.

The relative velocity of colliding bodies at the point of contact before and after every collision, $\vec{g}_{ij}$ and $\vec{g}'_{ij}$, are:

$$\begin{aligned} \vec{g}_{iw} &= (\vec{v}_i - \vec{\omega}_i \times R_i\vec{e}^{\,n}_{iw}) - (\vec{v}_w) \\ \vec{g}'_{iw} &= (\vec{v}_i' - \vec{\omega}_i' \times R_i\vec{e}^{\,n}_{iw}) - (\vec{v}_w) \end{aligned} \tag{2.33}$$

The velocity of the ball after collision is obtained by combining equations 2.23, 2.31, and 2.33. The derivation is presented in appendix A.7 and the result is presented here:

$$\vec{v}_i' = \vec{v}_i - (1+\varepsilon^n_w)\vec{g}^{\,n}_{iw} - \frac{\varepsilon^t_w - 1}{1 + \dfrac{1}{\tilde{J}_i}}\vec{g}^{\,t}_{iw} \tag{2.34}$$

$$\vec{\omega}_i' = \vec{\omega}_i + \frac{(1-\varepsilon^t_w)}{R_i(1+\tilde{J}_i)}\vec{e}^{\,n}_{iw} \times \vec{g}^{\,t}_{iw} \tag{2.35}$$

where $\tilde{J}$ is the reduced moment of inertia

$$\tilde{J}_i \equiv J_i / m_i R_i^2$$

The wall introduces or removes energy to or from the colliding ball; however, this energy is not visible in the equations of the COR.

The milling dose for the ball-wall collisions is derived in appendix A.10 and presented here:

$$D_v = \tfrac{1}{2} m_i \{ [1 - (\varepsilon_w^n)^2] (\vec{g}_{iw}^{\,n})^2 + \frac{1 - (\varepsilon_w^t)^2}{1 + \frac{1}{\tilde{J}_i}} (\vec{g}_{iw}^{\,t})^2 \} \tag{2.36}$$

Chapter 3

Implementation

The present chapter presents the implementation of the Event Driven Method in 3D incorporating moving walls. The logical structure of the program in described in detail. This chapter may thus serve as starting point for any further modifications of the source code.

3.1 Preliminary

3.1.1 Design

A few considerations were taken into account in the design of the program:

- The program should be flexible and scalable.
- The program should comply with international standards.
- The program should be written using open-source software to avoid license issues.

3.1.2 Programming language

The program that has been developed is a re-factoring of the code found in chapter 3 of Poschel and Schwager (2005) which is in C++. Therefore, C++ was the obvious choice of programming language to implement this code. C++ is also the programming language used in most of the existing MA simulations. The program was written using the editor NetBeans[1]. Figure 3.1 shows a screen shot of NetBeans showing the default configuration file for the program developed here. The code was tested only in Ubuntu

[1] www.NetBeans.com.

Linux[2] and compiled with g++[3] inside NetBeans. The code was designed using the guidelines found in Meyers (2005).

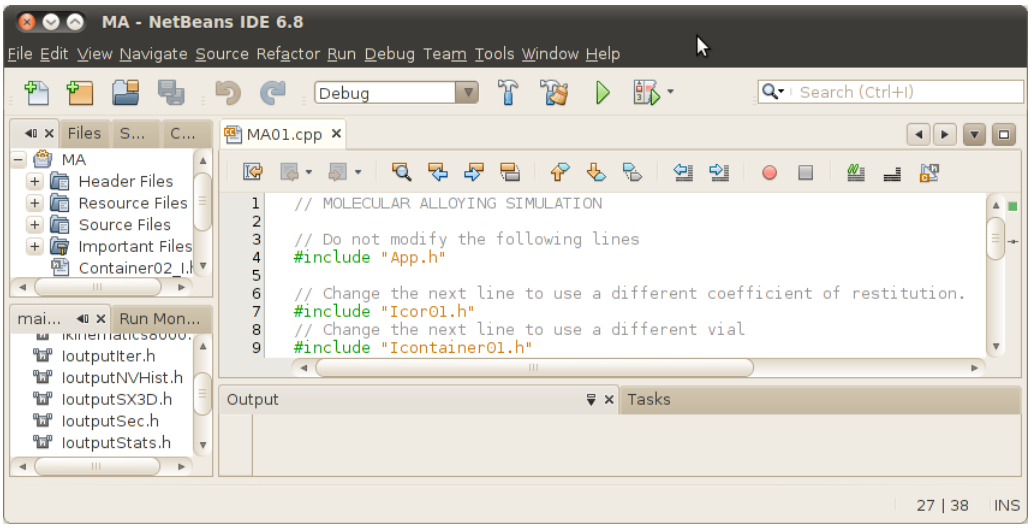


Figure 3.1: A NetBeans IDE screenshot.

The program developed here uses many of the Object-Oriented features of C++. The structure of the code was divided into so-called classes, with every class having its own file. This separation into classes allows the splitting of the program into logical sections. There is one class for the vial, another for the kinematics of the device, another for the balls, and so on.

Inheritance is one of the C++ features used here. The program is designed to be extendible, which is allowed by inheritance. Individual classes containing the rules for derived classes were written for the extendible parts of the code. For example, in the code the file Container.h is a foundation class for vials of different shapes, and the file Container01.h is an inherited class for cylindrical vials. Multiple inheritance is allowed in C++; but, it is avoided here to prevent public functions of different inherited classes from receiving the same name.

3.1.3 User interface

There is not an interface as such. There is no GUI (Graphical User Interface) or configuration file. The user must write a program in C++ and let the compiler recognize it as the main function. A user can choose which libraries to use or even create new ones.

[2] www.ubuntu.com
[3] http://gcc.gnu.org

Every component has user-oriented functions.

The default main file is MA01.cpp. The file is divided into three logical sections: libraries, commands, and launch. The first section names the libraries to use. This is done with the C++ instruction #include. All the principal components of the program, such as the vial or its kinematics, are referred here. The commands section is the actual configuration of parameters. In this section the components of the simulation are created and personalized.

The program was built to be flexible and extensible. For instance, a GUI can be created in the future.

3.1.4 Measurement system

All computations are performed using the metric system, specifically the MKS one. Some inputs let the user to use a different unit such as inches; but, all units are converted to the metric system later on. The file Conversions.h contains the equivalences between the unit systems.

3.1.5 Vectors

All the physics theory used in this program is expressed in terms of mathematical vectors. The file Xyzvector.h contains the class XYZVector to represent Cartesian vectors. This class allows vector-wise computations. This avoids making computation for every coordinate axis. As a consequence, the program uses fewer lines of code, and provides a more clear visualization of the physics inside the code.

3.1.6 Geometry

The program developed here involves geometry. The X3D standard was chosen as the foundation of all geometry for the reasons given in 3.1.1. Relevant details of this standard are summarized in the appendix C.

Three 3D entities of the X3D specification were created. A box, a cylinder, and a sphere are found respectively in the files G3DBox.h, G3DCylinder.h, and G3DSphere.h. A user interface for the same entities can be found respectively in the files IG3DBox.h, IG3DCylinder.h, and IG3DSphere.h. 3D entities can be solid or not. 3D entities have volume but no mass.

Two Cartesian coordinate systems are used to describe the vial motion in the Spex 8000. Their location and orientation is explained in section 2.2.2.

3.1.7 Vial

Vials are constructed with one or more X3D entities (See section 3.1.6). The walls of a vial are allowed to move freely. As an example, the vial can be a piston. Vials are independent from the kinematics of the machine.

The vial in the program is changeable. The prototype for all vials is the class Vial found in the file Vial.h. The prototype for all vial interfaces is the class IVial, which can be found in the file IVial.h.

The file Vial01.h is a cylindrical vial with fixed walls. This vial is the one used in the case of the Spex 8000. The file IVial01.h asks the user for parameters. The user parameters are described in appendix D.2.5.

3.1.8 Milling balls

Milling balls are spheres with kinematic properties. The balls are extensions of the class IG3DSphere. The class IG3DSphere is the geometrical representation of spheres, as explained in section 3.1.6. Balls have density, mass, moment of inertia, linear and angular speed, and linear and angular kinetic energy.

Only one type of ball was created for the program. Balls are implemented under the class "Sphere" and can be found in the file Sphere.h. The file Isphere.h asks the user for parameters. The user parameters are described in the appendix D.2.4.

3.1.9 Gravity

Gravity is optional in the program. Gravity only affects the trajectory of the milling balls. The trajectory is parabolic with gravity, whereas it is straight with no gravity. Gravity points in the negative Y direction according to the coordinate system defined here and shown in figure 2.3, i.e (0,-9.8,0) [m/s^2].

3.2 Implementation of the EDM in this study

The vial and the milling balls are simulated using EDM. The role of the powder is modelled by the coefficient of restitution, which is only used in the collision responses. The subsections below follow the numeration shown in figure 2.2.

3.2.1 Initialization

The simulation starts reading the commands of the user interface mentioned in section 3.1.3. The vial and the milling balls are created, as explained in sections 3.1.7 and 3.1.8 respectively. The vial is placed in its initial position using the procedure described later in section 3.2.3.1. All balls are randomly placed in the vial one after another. If a ball overlaps with another ball it is relocated until it has its own space. The governing parameters of the initial placement of balls are stated in appendix D.2.4.6. The lists of collisions described below in section 3.2.2.1 are created. The simulation now enters the cycle of prediction, propagation, and collision shown in figure 2.2. The first prediction is special. It computes the time for all balls in order to completely fill the collision lists.

3.2.2 Prediction

3.2.2.1 Lists of collisions

The collision lists are implemented practically as in Poschel and Schwager (2005). The collision lists can be found in the file Timetable.h. The first list is implemented with *map*. *Map* is a C++ standard container which automatically keeps its items sorted.

3.2.2.2 Ball-ball collision prediction

The implementation of ball-ball prediction is found in the file Motion.h under the function *ppcoll*. The equation 2.11 has numerical problems for small distances. A small manipulation helps to solve this inconvenience,

$$\Delta t = \left(\frac{-\vec{r}_{ij}\cdot\vec{v}_{ij} - \sqrt{(\vec{r}_{ij}\cdot\vec{v}_{ij})^2 - (\vec{r}_{ij}^2 - R_{ij}^2)\vec{v}_{ij}^2}}{\vec{v}_{ij}^2}\right)\left(\frac{-\vec{r}_{ij}\cdot\vec{v}_{ij} + \sqrt{(\vec{r}_{ij}\cdot\vec{v}_{ij})^2 - (\vec{r}_{ij}^2 - R_{ij}^2)\vec{v}_{ij}^2}}{-\vec{r}_{ij}\cdot\vec{v}_{ij} + \sqrt{(\vec{r}_{ij}\cdot\vec{v}_{ij})^2 - (\vec{r}_{ij}^2 - R_{ij}^2)\vec{v}_{ij}^2}}\right)$$

Finally,

$$\Delta t = \frac{\vec{r}_{ij}^2 - R_{ij}^2}{-\vec{r}_{ij}\cdot\vec{v}_{ij} + \sqrt{(\vec{r}_{ij}\cdot\vec{v}_{ij})^2 - (\vec{r}_{ij}^2 - R_{ij}^2)\cdot\vec{v}_{ij}^2}} \tag{3.1}$$

3.2.2.3 Ball-vial collision prediction

The file WallPredictor01.h contains the actual algorithm for prediction while the file Iwpredictor01.h asks the user for parameters. The algorithm is an implementation of a

interactive step-wise method based. Figure 3.2 illustrates this algorithm.

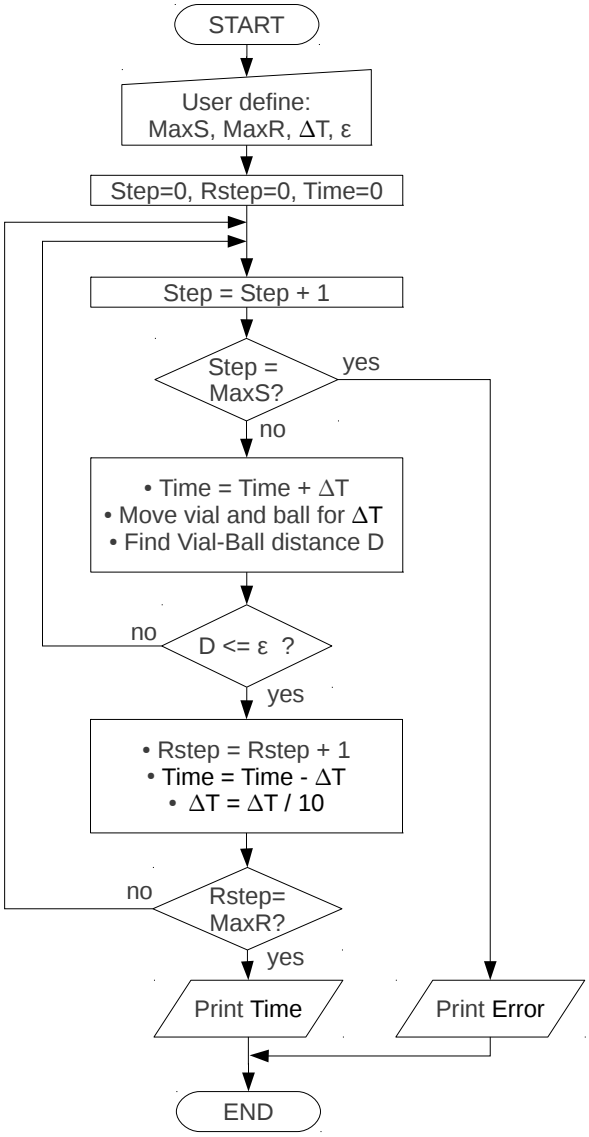


Figure 3.2: Flow diagram of the time predicting algorithm.

The parameters shown in the flowchart have the following default values in the actual implementation: $MaxS = 1\text{x}10^{7}$, $MaxR = 12$, $\Delta T = 1\text{x}10^{-2}$ s, and $MaxS = 1\text{x}10^{-5}$ m. The adjustment of those parameters required significant testing; therefore, their use is discouraged and not mentioned in the user guide.

3.2.3 Propagation

3.2.3.1 Milling balls propagation

The computation for the trajectory of the balls is stored separately in its own file. The current implementation of the equations is found in the file Motion.h under the function *project*. A user can implement a different model in another file and use it.

3.2.3.2 Vial propagation

The kinematics of the vial is implemented using the RTM. The RTM is implemented as an independent infrastructure in the file Kinematics.h. Only the relative transformations described in appendix B were developed. Thanks to RTM, it is easy to separate the geometry of the container from the kinematics of the machine. Some of the algorithms, such as the location of the balls and the ball-vial collision prediction, are simplified when the vial is used as the frame of reference.

The rotation matrix is constructed using quaternions. This is a common technique for computer graphics animations that doesn't need to be explained here.

The file Kinematics8000.h contains the actual kinematics of the Spex 8000 while the file Ikinematics8000.h asks the user for parameters. The user parameters are described in appendix D.2.5.

3.2.4 Collision

The file SphereResponse01.h contains the actual ball-ball collision response while the file Isresponse01.h asks the user for parameters. The file WallPredictor01.h contains the actual ball-wall collision response while the file Iwresponse01.h asks the user for parameters. The only parameter in both responses is the COR and it should be introduced as described in appendix D.2.7.

It is outside of the scope of the present study to investigate the COR dependence on different materials being milled. Therefore, the specification of the COR is left to the user. It is possible to specify a COR as simple as a constant or as complex as dependent on the particularities of every single collision.

3.2.5 Loop

Some user-defined criteria are evaluated at this point to decide if the simulation must continue. A new cycle is initiated again if the simulation continues; otherwise, the routine jumps to the output section. The program ends satisfactorily if no error occurred during

the simulation. The options for stopping the simulation and the way they should be introduced is presented in appendix D.2.2.

Every stop criterion is a class itself and is inherited from the class *Stop*. The class *Stop* is found in the file Stop.h. Subclasses of the class *Stop* have access to the main simulation variables in order to decide when to stop. The interface of the stops is embedded in the *App* object, which is found in App.h. More stops can be created via this App object.

3.2.6 Outputs

The outputs constitute the regular functioning of the program to export data. The available outputs and their uses are explained in appendix D.4. The program sends the main simulation variables to every output after every collision. Outputs have enough information at the end of the simulation to produce their respective files. The infrastructure built to implement the geometry (section 3.1.6) was extended in order to be able to export graphical content to files.

The outputs were implemented in such way that they are fully expansible and flexible. A user can implement new outputs easily. There is one class per output and every class has it own file. All outputs are inherited from the class *Output*, which is found in the file Output.h.

Chapter 4

Simulation of an example

4.1 Introduction

This chapter shows some capabilities of the software developed in this study. This chapter is divided into three sections. The first section shows an example of how to introduce the values to the program. The second section shows how to produce some outputs. The last section presents the configuration file at its full extension. One of the simulations of the chapter 5 is presented here as a case study. The code shown here is for reference only. An explanation of every instruction is provided in appendix D.

One of the capabilities of the program is to let the user choose the unit of measurement. For example, the diameter of the balls can be selected in inches while the vial can be introduced in centimetres. All amounts are converted to the metric system before the simulation starts. The code also permits a series of user-defined outputs. Outputs can be text files as statistics of the collision processes, or graphics like snapshots or animations. Almost a hundred variables can be reported. The list of variables can be found in appendix D.5.

The program has other capabilities not shown here. The software was designed to have exchangeable parts such as the vial or the kinematics of the machine, or even, the most fundamental parts like the collision response can be replaced.

4.2 Introducing the simulation parameters

4.2.1 Simulation duration

A simulation can stop after certain time or accumulated milling dose. The case study was configured to last 150 seconds, and the following line tells the program to do so:

```
Config->setMaximumTime(150);
```

4.2.2 Introducing the milling balls

The case study runs with one steel ball of diameter 9.52 mm. This is how those numbers are introduced into the program:

```
Config->setDiameter(9.52, "mm");
Config->setSphereDensity(8, "g/cm3");
Config->addSpheres(1);
```

4.2.3 Vial

The default vial is a cylinder. The diameter, height, and colour can be changed. The following lines show how the corresponding code looks like:

```
Collision::IVial01 * Container = new Collision::IVial01(&Interface);
Container->cylinder.setDiameter(38, "mm");
Container->cylinder.setHeight(57, "mm");
Container->cylinder.setColour("yellow");
```

4.2.4 Kinematics

The kinematics is pre-configured for the Spex 8000. This case study runs with angular speed of 1054 rpm rather than the default one. The following code shows how to create and change the kinematics:

```
Collision::IKinematics8000 * Kinematics =
    new Collision::IKinematics8000(&Interface);
Kinematics->setAngularSpeed(1054, "rpm");
```

4.2.5 Coefficient of restitution

The Incrementally Slipping Friction Model proposed by Walton and Braun Walton and Braun (1986) is used in the case study. A friction coefficient of 0.4 and a normal coefficient of restitution of 0.5 are the selected values. The following code shows how to set those values:

```
Collision::ICOR02 * COR = new Collision::ICOR02(&Interface);
COR->setNormalCOR(0.5);
COR->setFrictionCoeff(0.4);
```

4.3 Outputs

There are many outputs available from statistics to animations. It is up to the user to choose which ones to use. The outputs have to be configured before the simulation starts. The program keeps a database of variables that is updated after every collision and then send to all outputs.

4.3.1 Snapshots

The program can produce 3D Snapshots using the X3D format. More information about this format is found in appendix C. The following code shows how to produce a snapshot of the second collision. The contact point is shown as a small yellow sphere.

```
// SINGLE GRAPHICS FRAMES
Collision::IOutputSX3D * X3DSnapshots =
    new Collision::IOutputSX3D(&Interface);
X3DSnapshots->timer.setInitialIteration(2);
X3DSnapshots->timer.setIterations(1);
X3DSnapshots->showContactPoint(true);
X3DSnapshots->contactPoint.setRadius(1, "mm");
X3DSnapshots->contactPoint.setColour("yellow");
Config->addOutput(X3DSnapshots);
```

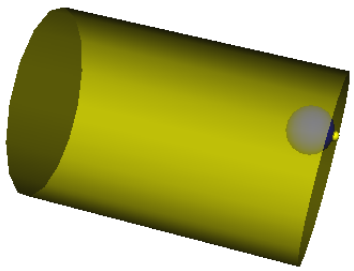


Figure 4.1: 3D model of the vial and a ball. The smaller sphere (in yellow) is the representation of the contact point.

4.3.2 Animation

Animations create a graphical animation of the system. The animation is a video. The video looks like a sequence of snapshots. Animations are created using the X3D format. More information about this format is found in appendix C. The following code is an example of how to create a video. The video starts at time zero and last 4 seconds.

```
// ANIMATION
Collision::IOutputX3D * X3DAnimation =
    new Collision::IOutputX3D(&Interface);
X3DAnimation->timer.setInitialTime(0);
X3DAnimation->timer.setDuration(4);
X3DAnimation->setAnimationSpeed(0.001);
X3DAnimation->setAnimationFPM(60);
X3DAnimation->setAnimationMinFrames(24);
Config->addOutput(X3DAnimation);
```

4.3.3 Accumulated Milling dose

This output produces a report of various values related with the accumulated milling dose at the end of the simulation. The result of this output is written in a CSV file. The following code prints the milling dose, the milling dose rate, the inverse of the milling dose rate, the specific milling dose, and specific milling dose rate. The inverse of the milling dose rate is proportional to the processing time and may be useful in comparing experimental data. There are other possible values to print out.

```
// ACCUMULATED MILLING DOSE
Collision::IOutputMD * MD = new Collision::IOutputMD(&Interface);
MD->timer.setInitialTime(30, "s");
MD->showField(Collision::OutputMD::dMD);
MD->showField(Collision::OutputMD::dMDRate);
MD->showField(Collision::OutputMD::dMDRateInverse);
MD->showField(Collision::OutputMD::dSMD);
MD->showField(Collision::OutputMD::dSMDRate);
Config->addOutput(MD);
```

MD	MD RATE	MD RATE INV	SPEC MD	SPEC MD RATE
124.1	1.034	0.967	62052.3	517.081

4.3.4 Frequency table

This output produces a table of frequencies of any double value from the table D.5. This output is useful to produce histograms. The resulting data is written in a file with extension CSV. This output has few options such as the display of values out of bounds, or the display of values per wall. The following example creates a table of frequencies of the normal speed before collision with six intervals.

```
Collision::IOutputDFrecuency * NVF =
    new Collision::IOutputDFrecuency(&Interface);
NVF->filename.setFilePrefix("NVFrecuencies");
NVF->timer.setInitialTime(30, "s");
NVF->showHeader(Collision::OutputDFrecuency::headerIntervalNumber);
NVF->showHeader(Collision::OutputDFrecuency::headerLowerBounds);
NVF->showHeader(Collision::OutputDFrecuency::headerUpperBounds);
NVF->showHeader(Collision::OutputDFrecuency::headerMeans);
NVF->showSection(Collision::OutputDFrecuency::sectionWallDetails);
NVF->setDouble(Collision::dVelNormalRelBeforeMag);
NVF->setInitialValue(0);
NVF->setFinalValue(6);
NVF->setNumberOfIntervals(6);
Config->addOutput(NVF);
```

INTERVAL #	1	2	3	4	5	6
LOWER BOUND (LB)	0	1	2	3	4	5
UPPER BOUND (UB)	1	2	3	4	5	6
MEAN OF INTERVAL	0.5	1.5	2.5	3.5	4.5	5.5
WALL 0	1765	523	275	434	844	276
WALL 1	1809	493	243	394	857	306
WALL 2	41209	2831	1072	376	100	9
B2W	44783	3847	1590	1204	1801	591
B2B	0	0	0	0	0	0
TOTAL	44783	3847	1590	1204	1801	591

4.3.5 Statistics

This output produces statistical amounts such summation, average, minimum, maximum, and rate of any of the values of the tables D.3 and D.5. Statistics are produced at the end of the simulation. The content in the output file is self-explanatory. CSV is

the format used. The following code generates statistics of the number of collisions and the milling dose.

```
Collision::IOutputStatistics * Statistics =
  new Collision::IOutputStatistics(&Interface);
Statistics->timer.setInitialTime(30, "s");
Statistics->addBool(Collision::bCol, "COL.", Collision::sSum,
    Collision::sPercentage, Collision::sRate);
Statistics->addDouble(Collision::dMillingDose, "MIL. DOSE",
    Collision::sAve, Collision::sMax);
Config->addOutput(Statistics1);
```

The following table shows the results obtained for the previous code

STATISTICS					
Initial Time:	30	s	#	of balls:	1
Final Time:	150.005	s	#	of walls:	3
Rec. Time:	120.005	s			

COL.	WALL 0	WALL 1	WALL 2	T. WALL	B2W	B2B	TOTAL
Sum []	4579	4286	46694	55559	55559	0	55559
Percen [%]	8.242	7.714	84.044	100	100	0	100
Rate [/s]	38.157	35.715	389.101	462.972	462.972	0	462.972

MIL. DOSE	WALL 0	WALL 1	WALL 2	T. WALL	B2W	B2B	TOTAL
Average [/col]	0.010	0.011	0.001	0.002	0.002	0	0.002
Max []	0.073	0.082038	0.0374788	0.082038	0.082	0	0.082

4.3.6 Trends

Trends shows statistical information of some variables in time-intervals basis. This output helps to visualize how the system behaves in time, especially if a graph is created from the data. This output accepts summation, average, minimum, maximum, and rate of the values of the tables D.4 and D.5, and summation and rate of values from table D.3. This output writes a text file in CSV format. The following code and table are an example of this output.

```
Collision::IOutputTrends * Trends =
    new Collision::IOutputTrends(&Interface);
```

```
Trends->timer.setTimeInterval(0.5, "s");
Trends->addBool(Collision::bCol, Collision::sSum, "COL SUM");
Trends->addBool(Collision::bCol, Collision::sRate, "COL RATE");
Trends->addDouble(Collision::dEnergyTransferWall, Collision::sRate,
    "WE RATE");
Trends->addDouble(Collision::dMillingDose, Collision::sRate, "MD RATE");
Config->addOutput(Trends);
```

TIME	COL SUM	COL RATE	WE RATE	MD RATE
0.507	243	479.473	1.279	1.197
1.032	136	259.080	1.062	1.036
1.535	173	343.487	1.026	1.112
2.036	215	429.123	1.165	1.118

The following option shows accumulated values from the beginning of the recording time instead of accumulated values per time spans.

```
Trends->setModeAccum();
```

The following table is an example of the accumulated number of collisions, the transfer of energy from the wall, the milling dose, and the difference between the last two amounts.

Time	COL	WE	MD	E. BAL
0.507	381	0.543738	0.474177	0.0695611
1.032	726	1.04232	0.98563	0.0566945
1.535	897	1.56167	1.50777	0.0538986
2.036	1073	2.06791	2.03579	0.0321234

4.3.7 Trends of instantaneous values

This output is similar to trends except that it does not produce statistical values. It only prints out the actual value of the variables. This output is useful for quality assurance purposes. This output writes a text file in CSV format. The following code shows the actual value of several variables.

```
Collision::IOutputTrendsInst * TrendsInst =
    new Collision::IOutputTrendsInst(&Interface);
TrendsInst->timer.setInitialTime(30);
```

```
TrendsInst->addDouble(Collision::dTimeCurrent, "Time");
TrendsInst->addInteger(Collision::iIteration, "It");
TrendsInst->addInteger(Collision::iFirst, "Id1");
TrendsInst->addInteger(Collision::iSecond, "Id2");
TrendsInst->addBool(Collision::bBallBall, "B/V", "B", "V");
TrendsInst->addDouble(Collision::dEnergyTransferWall, "E. WALL");
TrendsInst->addDouble(Collision::dMillingDose, "MD");
TrendsInst->addDouble(Collision::dEnergyBalance, "ENER. BAL");
Config->addOutput(TrendsInst);
```

Time	It	Id1	Id2	B/V	E. WALL	MD	ENER. BAL
30.0003	12747	0	2	V	0.0030	1.427e-05	0.0030
30.0009	12748	0	2	V	-0.0024	2.283e-05	-0.0024
30.0020	12749	0	2	V	0.0058	1.592e-05	0.0058
30.0027	12750	0	2	V	-0.0048	2.564e-05	-0.0049
30.0039	12751	0	2	V	0.0094	1.946e-05	0.0094

4.3.8 Trends of ball data

This output is similar to instantaneous trends, but it was created to present data of the milling balls. The variables available to display are density, mass, reduced moment of inertia, moment of inertia, translational speed, rotational speed, translational energy, rotational energy, and total energy. It is possible to produce statistics such summation, average, minimum, maximum from all the balls together. The following code and table are an example to compute the average speed and maximum speed.

```
Collision::IOutputTrendsBalls * TB =
    new Collision::IOutputTrendsBalls(&Interface);
TB->timer.setInitialTime(30);
TB->timer.setTimeInterval(0.5, "s");
TB->addField(Collision::Sphere::dTransSpeed, Collision::sAve, false,
    "Ave Speed" );
TB->addField(Collision::Sphere::dTransSpeed, Collision::sMax, true,
    "Max Speed");
Config->addOutput(TB);
```

Time	It	Ave Speed	Max Speed	Id
30.509	12996	1.808	1.808	0
31.016	13101	5.102	5.102	0
31.519	13272	3.910	3.910	0
32.028	13367	5.466	5.466	0

4.4 Sample code

The code of this chapter is presented in the full extent in this section.

```
// BALL MILLING SIMULATION
#include "IConfig.h"  // Main file.
#include "Icor02.h"  // COR
#include "IVial01.h"  // Vial
#include "Ikinematics8000.h"  // Motion of the vial
#include "Iwpredictor01.h"  // Sphere-wall collision predictor
#include "Isresponse01.h"   // Sphere-Wall collision response
#include "Iwresponse01.h"  // Wall-Wall collision response
#include "IoutputDFrecuency.h"   // Table of frequencies
#include "IoutputIter.h"  // Iteration details
#include "IOutputMD.h"  // Accumulated milling dose
#include "IoutputStatistics.h"       // Statistics
#include "IoutputSX3D.h"   // Graphical output per iteration in X3D format
#include "IOutputTrace.h"
#include "IoutputTrends.h"   // Trends
#include "IoutputTrendsInst.h"   // Trends instantaneous
#include "IoutputTrendsBalls.h"  // Trends of balls variables
#include "IoutputX3D.h"   // Graphical simulation in X3D format
int main() {
  // Simulation Parameters
  Collision::Database database;
  Collision::Interface Interface(&database);
  Collision::IConfig * Config = new Collision::IConfig(&Interface, &database);
  Config->setOutputFolder("OPT4"); // Outputs
  Config->setMaximumTime(150);
  Config->setGravity(); // Gravity ON/OFF
  Config->setDiameter(9.52, "mm");
  Config->setSphereDensity(8, "g/cm3");
  Config->setSphereVelocity(0, 0, 0, "m/s");
  Config->setInitialS2SphereGap(1, "mm");
  Config->setInitialS2WallGap(1, "mm");
  Config->setSphereMaxRelocations(500);
```

```
Config->setColour("blue");
Config->addSpheres(1); // Number of particles
// CONTAINER
Collision::IVial01 * Container = new Collision::IVial01(&Interface);
Container->cylinder.setDiameter(38, "mm");
Container->cylinder.setHeight(57, "mm");
Container->cylinder.setColour("yellow");
// KINEMATICS
Collision::IKinematics8000 * Kinematics = new
    Collision::IKinematics8000(&Interface);
Kinematics->setAngularSpeed(1054, "rpm");
// ALL TIMES COR
Collision::ICOR02 * COR = new Collision::ICOR02(&Interface);
COR->setNormalCOR(0.5);
COR->setFrictionCoeff(0.4);
// BALL-WALL RESPONSE
Collision::IWResponse01 * Wresponse = new Collision::IWResponse01(&Interface);
Wresponse->setCOR(COR);
 // SPHERE-SPHERE RESPONSE
Collision::ISResponse01 * Sresponse = new Collision::ISResponse01(&Interface);
Sresponse->setCOR(COR);
//PREDICTION
Collision::IWPredictor01 * Wpredictor = new
    Collision::IWPredictor01(&Interface);
// OPTIMIZATION
Collision::IOutputMD * MD = new Collision::IOutputMD(&Interface);
MD->timer.setInitialTime(30, "s");
MD->showPowderMass(2, "g");
MD->showChargeRatio(2.5);
MD->showField(Collision::OutputMD::dMD);
MD->showField(Collision::OutputMD::dMDRate);
MD->showField(Collision::OutputMD::dMDRateInverse);
MD->showField(Collision::OutputMD::dSMD);
MD->showField(Collision::OutputMD::dSMDRate);
Config->addOutput(MD);
// ANIMATION
Collision::IOutputX3D * X3DAnimation = new Collision::IOutputX3D(&Interface);
X3DAnimation->timer.setInitialTime(0);
X3DAnimation->timer.setIterations(4);
X3DAnimation->setAnimationSpeed(0.001); // Playback speed
X3DAnimation->setAnimationFPM(60); // Frames per minute
X3DAnimation->setAnimationMinFrames(24);
X3DAnimation->showContactPoint(true);
```

```
X3DAnimation->contactPoint.setRadius(1, "mm");
X3DAnimation->contactPoint.setColour("yellow");
Config->addOutput(X3DAnimation);
// SINGLE GRAPHICS FRAMES
Collision::IOutputSX3D * X3DSnapshots = new Collision::IOutputSX3D(&Interface);
X3DSnapshots->timer.setInitialIteration(1);
X3DSnapshots->timer.setIterations(2);
X3DSnapshots->showContactPoint(true);
X3DSnapshots->contactPoint.setRadius(1, "mm");
X3DSnapshots->contactPoint.setColour("yellow");
Config->addOutput(X3DSnapshots);
// NORMAL VELOCITY HISTOGRAM
Collision::IOutputDFrecuency * NVFrecuencies = new
    Collision::IOutputDFrecuency(&Interface);
NVFrecuencies->filename.setFilePrefix("NVFrecuencies");
NVFrecuencies->timer.setInitialTime(30, "s");
NVFrecuencies->restrictions.addRestrictionBool(Collision::bBallBall, false);
NVFrecuencies->setDouble(Collision::dVelNormalRelBeforeMag,
    "NORMAL VELOCITY BEFORE COLLISION");
NVFrecuencies->setInitialValue(0.025);
NVFrecuencies->setFinalValue(7.025);
NVFrecuencies->setNumberOfIntervals(10);
Config->addOutput(NVFrecuencies);
// STATISTICS
Collision::IOutputStatistics * Statistics =
    new Collision::IOutputStatistics(&Interface);
Statistics->timer.setInitialTime(30, "s");
Statistics->addBool(Collision::bCol, "COL.", Collision::sSum,
    Collision::sPercentage, Collision::sRate);
Statistics->addDouble(Collision::dMillingDose, "MIL. DOSE", Collision::sAve,
    Collision::sMax);
Config->addOutput(Statistics1);
// TRENDS
Collision::IOutputTrends * Trends = new
    Collision::IOutputTrends(&Interface);
Trends->timer.setTimeInterval(0.5, "s");
Trends->addBool(Collision::bCol, Collision::sSum, "COL SUM");
Trends->addBool(Collision::bCol, Collision::sRate, "COL RATE");
Trends->addDouble(Collision::dEnergyTransferWall, Collision::sRate, "WE RATE");
Trends->addDouble(Collision::dMillingDose, Collision::sRate, "MD RATE");
Config->addOutput(Trends);
// BALLS TRENDS
```

```
  Collision::IOutputTrendsBalls * TB = new
      Collision::IOutputTrendsBalls(&Interface);
  TB->timer.setInitialTime(30);
  TB->timer.setTimeInterval(0.5, "s");
  TB->addField(Collision::Sphere::dTransSpeed, Collision::sAve, false, "Ave Speed" );
  TB->addField(Collision::Sphere::dTransSpeed, Collision::sMax, true, "Max Speed");
  Config->addOutput(TB);
  // TRENDS INST
  Collision::IOutputTrendsInst * TrendsInst = new
      Collision::IOutputTrendsInst(&Interface);
  TrendsInst->timer.setInitialTime(30);
  TrendsInst->timer.setTimeInterval(0.5, "s");
  TrendsInst->restrictions.addRestrictionUInt(Collision::iFirst, 19);
  TrendsInst->addDouble(Collision::dTimeCurrent, "Time");
  TrendsInst->addInteger(Collision::iIteration, "It");
  TrendsInst->addInteger(Collision::iFirst, "Id1");
  TrendsInst->addInteger(Collision::iSecond, "Id2");
  TrendsInst->addBool(Collision::bBallBall, "B/V", "B", "V");
  TrendsInst->addDouble(Collision::dEnergyTransferWall, "ENER. WALL");
  TrendsInst->addDouble(Collision::dMillingDose, "MD");
  TrendsInst->addDouble(Collision::dEnergyBalance, "ENER. BAL");
  TrendsInst->addDouble(Collision::dMillingDoseAcum, "ACC MD");
  TrendsInst->addInteger(Collision::iAdjustmentPos,"A.Pos");
  TrendsInst->addInteger(Collision::iAdjustmentTime,"A.Time");
  Config->addOutput(TrendsInst);
  // TRACE
  Collision::IOutputTrace * Trace = new Collision::IOutputTrace(&Interface);
  Trace->timer.setTimeInterval(0.5, "s");
  Trace->setBallId(19);
  Config->addOutput(Trace);
  // ITERATIONS
  Collision::IOutputIter * Iteration = new Collision::IOutputIter(&Interface);
  Iteration->timer.setInitialIteration(993763);
  Iteration->timer.setIterations(3);
  Iteration->restrictions.addRestrictionUInt(Collision::iFirst, 9);
  Config->addOutput(Iteration);
  // Application start up - Do not change this block
  Collision::Core App = Config->getCore(Container, Kinematics, Wpredictor,
      Sresponse, Wresponse);
  delete Config;
  App.run();
  return (EXIT_SUCCESS);
}
```

Chapter 5

Optimization

One application of the program developed in this study is to find the optimal operative conditions of the Spex 8000. The optimization objective is to maximize the use of the equipment. The higher the efficiency of the process the lower the processing time, which in turn increases the savings in all other aspects. The processing time depends on many variables as mentioned in the introduction. The optimal operative condition is the combination of values of process variables that lead to the highest processing rates. Optimization can also be conducted by minimizing energy consumption; however, experimental data are but unavailable to perform a comparison.

This optimization study recreates the experiments mentioned in Ward et al. (2005). They simulated the Spex 8000 using DEM, a method explained in section 2.1.3. This is the only source found that provides enough experimental information to start a systematic study the Spex 8000. Unfortunately, the experimental information provided is not enough to test the accuracy of the program developed here, and only a partial comparison is performed here, similar to the modelling attempted by Ward et al.

The only experimental finding provided by Ward et al is the milling time. Those results are shown in figure 4 of their publication and brought here in the second row of figure 5.2. From this figure it is possible to find out the optimal point. A plot of the milling time for the combination of three different parameters is shown. This time is the delay until the powders in the vial reach a critical refinement when they start reacting in the vial by the SHS process, as explained in the introduction of this thesis.

Ward et al. made a partial validation of the numerical results with a visual comparison of the inverse of the MD rate with experimental times. The higher the MD rate $\dot{D}$ the shorter the processing time t, i.e.:

$$t \propto \frac{1}{\dot{D}}$$

The program developed in this thesis can not compute the milling time without knowing in advance another parameter like the milling dose or number of collisions. Therefore, the same approach made by Ward to validate the program seems to be the logical procedure to follow. Moreover, this kind of validation agrees with the accuracy of the equations used to computed the milling dose. The milling dose is computed making few assumptions. One assumption is that the change of kinetic energy of the colliding milling balls goes entirely to transform the powder. In reality, part of this energy is lost into heat due to friction. The powder spreads out as the balls collide due to the granular nature of the powder. Part of the kinetic energy of the ball is first dissipated by friction before it starts producing plastic deformation to the powder. Another assumption is that the tangential COR was assumed to follow the model of Walton and Braun; however, its validity in the presence of powder is yet to be proved.

The parameters of the study are the same as those chosen by Ward to make the studies comparable. However, it does not mean that those parameters are the most relevant. Further studies are required to identify the key parameters. The parameters used in the current simulations are:

- Two powder masses: 2 and 5 grams.
- Four ball diameters: 2.36, 3.16, 4.76, and 9.52 millimetres.
- Three charge ratios: 2.5, 5, and 10.

The charge ratio is defined as:

$$C_R = \frac{n_b m_b}{m_p} \tag{5.1}$$

where n_b is the number of balls, m_p is the powder mass, and m_b is the mass of a single ball.

Ward et al. employ two types of powders for the analysis: Al-Fe_2O_3 and Al-MoO_3. However, they provide the normal COR for the first material only. For this reason, the simulations here are limited to Al-Fe_2O_3. Thirty six simulations were performed by Ward to combine the effect of the three parameters and the two materials. Here, with only one material, the number of simulations is reduced to 24 (2 powder masses times 4 ball diameters times 3 charge ratios).

Table 5.1 shows the diameter and mass of the milling balls used. The mass is computed using a density of 8 g/cm^3 (8000 kg/m^3), which corresponds to the density of steel.

Table 5.1: Diameters and mass of milling balls.

Diameter, ø [mm]	2.36	3.16	4.76	9.52
Mass [g]	0.055	0.132	0.452	3.614

Table 5.2 shows the total milling balls mass and number of balls. The sum of the balls mass is found by multiplying the charge ratio and the powder mass. The number of balls is found from equation 5.1 and table 5.1.

Table 5.2: Total mass and number of milling balls.

Powder mass [g]	C_R	Total ball mass [g]	Number of balls per ø			
			2.36	**3.16**	**4.76**	**9.52**
2	2.5	5	91	38	11	1
	5	10	182	76	22	3
	10	20	363	151	44	6
5	2.5	12.5	227	95	28	3
	5	25	454	189	55	7
	10	50	908	378	111	14

Table 5.3 shows the normal COR, ε^n, used by Ward. et al. Those values come from the figure 2 of their paper. The normal COR was found to be dependent on the thickness of the powder layer expected to coat the ball surfaces. The normal COR varies between 0.5 and 0.8. The thickness of the powder layer is function of the powder volume and the surface area of the balls, or indirectly, this thickness is a function of the charge ratio and the diameter of milling balls. It is important to mention that the shape and dimensions of the vial have an effect on the thickness of the powder layer since part of the powder is accumulated in the corners and edges of the vial walls. This last irregularity is not taken into account for simplicity.

Table 5.3: Normal COR.

	Diameter [mm]			
C_R	**2.36**	**3.16**	**4.76**	**9.92**
2.5	0.56	0.54	0.53	0.5
5	0.64	0.60	0.56	0.52
10	0.80	0.74	0.64	0.55

The tangential COR is computed by Ward using the model of Walton and Braun. This model is presented in section 2.4.1. The reduced moment of inertia $\tilde{J}$ for solid spheres is 2/5. The friction coefficient μ used by Ward is 0.4. Then, the tangential COR becomes:

$$\varepsilon^t = max\left(-1, 1 - 1.4(1+\varepsilon^n)\left|\vec{g}^n/\vec{g}^t\right|\right) \tag{5.2}$$

The other parameters in the simulation by Ward that are relevant to the simulation with EDM are condensed in the table 5.4.

Table 5.4: Parameters common to all simulations.

Vial diameter	38 mm
Vial height	57 mm
Friction coefficient μ	0.4
Gravity	9.8 m/s^2
Values of equation 2.9	
Angular speed w	1054 rpm
Arm length l	10 cm
Vial angle γ_m	15°
Vial angle θ_m	15°

The MD rate, $\dot{D}$, is found from the ratio of the computed MD, D, and the simulation time, Δt, i.e.

$$\dot{D} = \frac{D}{\Delta t}$$

The MD is computed in the program as the accumulated MD per collision, i.e.

$$D = \sum_{k=1}^{n_c} D_k$$

where n_c is the number of collisions that occur during the simulation.

The MD per collision is computed with the equation 2.30 for ball-ball collisions, and with the equation 2.36 for ball-vial collisions. All balls in a simulation are equal meaning they have the same diameter, mass, and moment of inertia. The reduced moment of inertia, $\tilde{J}$, for solid spheres is 2/5. Therefore, the MD per collision is equal to

$$D_k = \frac{1}{\beta} m_b \left\{ [1 - (\varepsilon^n)^2](\vec{g}^n)^2 + \frac{2}{7}[1 - (\varepsilon^t)^2](\vec{g}^t)^2 \right\}$$

where β is 4 for ball-ball and 2 for ball-vial collisions respectively, m_b is the mass of a single ball.

All variables have been defined. It is now possible start the simulations. All balls start from rest at the beginning of the simulation. It takes some time to the system to reach its equilibrium state. Preliminary tests have shown that the equilibrium time is reached in less than half a second for all simulations. The milling dose rate for two

simulations is shown in figure 5.1. Information from all simulations is collected after 30 seconds to be completely sure that a steady state has been reached.

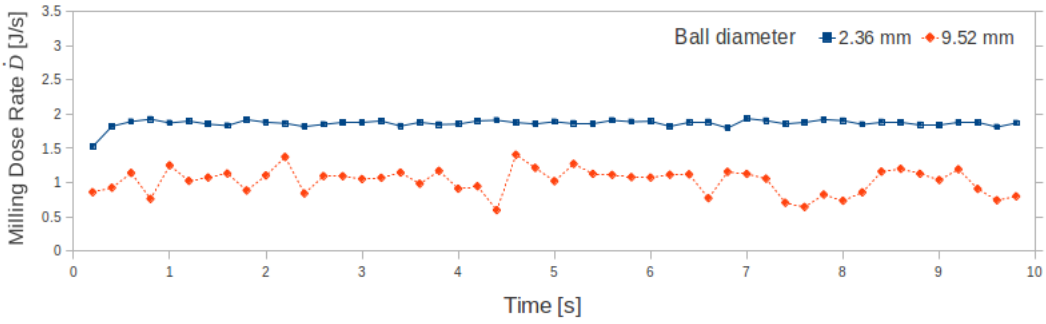


Figure 5.1: Plot of temporal changes in computed milling dose rate, $\dot{D}$. The graphs correspond to the 2 g sample with a charge ratio of 2.5. Linear interpolation.

The numerical results of the inverse of $\dot{D}$ for different diameters and charge ratio are presented in table 5.5.

Table 5.5: Numerical results of the inverse of the milling dose rate (in s/J).

	2g sample			*5g sample*		
DIAM	CHARGE RATIO			CHARGE RATIO		
	2.5	5	10	2.5	5	10
2.36	0.5344	0.2752	0.1647	0.2064	0.1101	0.0684
3.16	0.5425	0.2722	0.1538	0.2061	0.1072	0.0625
4.76	0.5770	0.2812	0.1452	0.2123	0.1078	0.0572
9.52	0.9670	0.2931	0.1428	0.2869	0.1184	0.0606

Data from table 5.5 and numerical and experimental results in Ward are shown in figure 5.2. This graph is the only way to compare all results since no tabulated data is found in Ward. It can be observed that the results do not differ significantly. The shape and positions of the curves do not match exactly but they look coherent. The series of data that correspond to a charge ratio of 2.5 for the 5 g sample is the only series that looks out of order.

It seems that the EDM proposed in this study is more accurate that the DEM proposed by Ward by comparing the shapes of the curves. Last affirmation comes from comparing the graphs of the figure 5.2. Unfortunately, Ward only shows the numerical results for the 2 g sample mass. Another numerical result is obtained for the 5 g sample and a C_R of 5 in the figure 8 of Ward. The actual values of the inverse of the milling dose do not differ significantly. However, it is not correct to validate the results of a numerical simulation with another numerical simulation. The actual experimental values

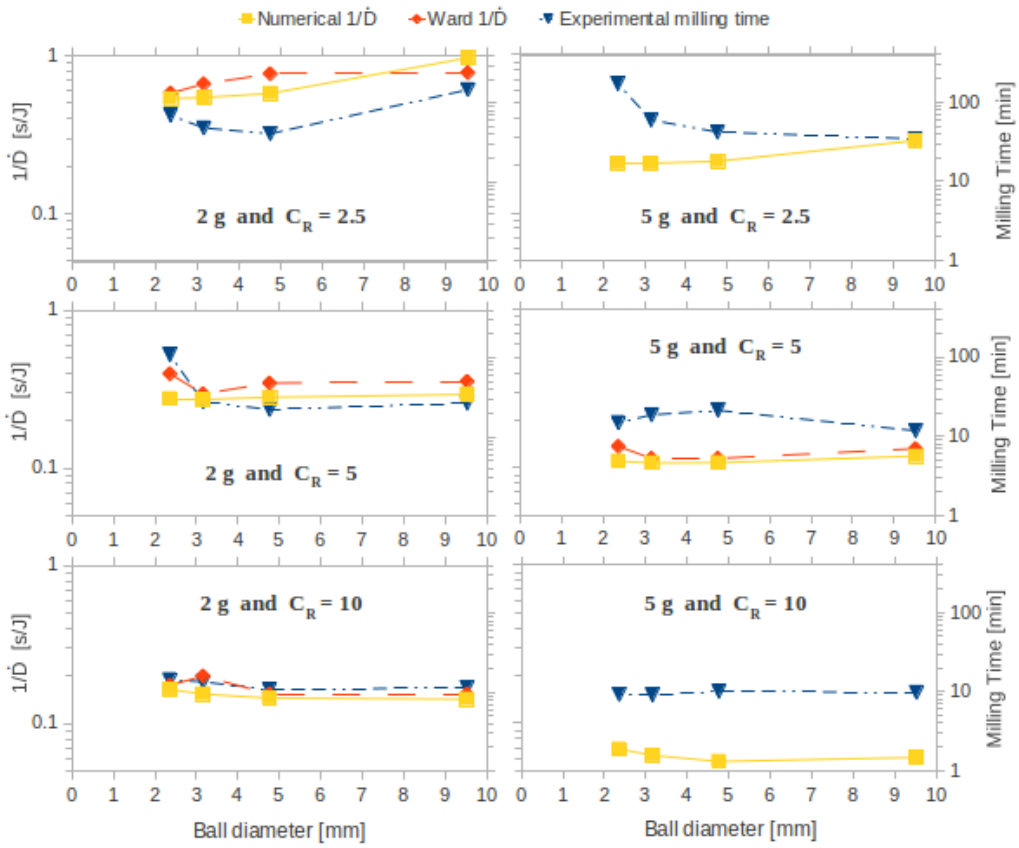


Figure 5.2: Comparison between numerical results of the inverse of the milling dose and experimental results of the milling time. One set of data was produced using the program developed in this study. The other two sets of data are numerical and experimental results shown in Ward et al. (2005) (Figures 4, 7 and 8 of their paper). Linear interpolation.

and not the shape of the curves is the right way to make a comparison. Once again, more experimental data is needed to evaluate the accuracy of both models.

The same numerical results can be used to seek the optimal point. The MD per unit mass is used instead of the plain MD to be able to compare the results from samples of different mass. The MD per unit mass is known as the Specific Milling Dose (SMD) and is expressed as,

$$D_m = \frac{D}{m_p}$$

where m_p is the powder mass.

The optimal point will be the combination of parameters that maximize the SMD rate. The SMD rate is found computing the accumulated SMD and divided it by the simulation time, Δt, i.e.

$$\dot{D}_m = \frac{D_m}{\Delta t} = \frac{\dot{D}}{m_p}$$

The results for all simulations are presented in figure 5.3. It can be observed from the graphs that higher SMD rates are obtained by using higher charge ratios, which agrees with the experimental data. It seems that the diameter of the balls or the number of balls have little influence on the output for a constant value of charge ratio; however, this last affirmation is not true for experimental data where balls with higher diameters are more effective, or a very small number of balls is less effective. It was also found that the results for both powder masses are similar, which implies that the powder mass does not affect the SMD rate if the charge ratio is kept the same. This similarity is also found in the experimental data.

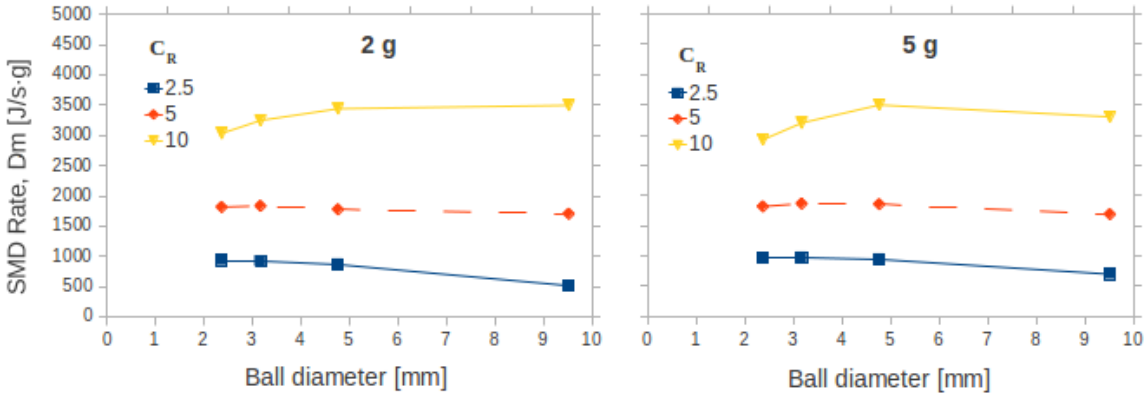


Figure 5.3: Specific milling dose rate from numerical simulations.

In conclusion, the optimal point is reached with larger charge ratios, and that means larger diameters or number of balls. More simulations are required to conclude what might be the optimal diameter or the optimal number of balls. Due to the small number of observation and the lack of independence and normality between the variables, it is not feasible to make a statistical model able to accurately describe the machine operation Tabachnick and Fidell (2007).

The EDM is more efficient that the DEM in terms of computer resources as mentioned in the introduction. Unfortunately, Ward et al. do not provide information about computer processing time to perform a comparison. Some rates are presented in table 5.6 just to give an idea of what it takes to conduct a simulation. The computer time is mainly affected by the number of collisions, as one may expect.

Table 5.6: Collision rates using both simulation and computer times, and the ratio of computer time per simulation time. The data shown corresponds to the 5g sample. The computer used was a Dell Inspiron One with Ubuntu Linux 10.10.

Diameter [mm]	# Balls	Collision rate [col/s]	Collisions / comp. time [col/s]	computer time / simulation time
2.36	227	64366	1254	52
2.36	454	126155	973	130
2.36	908	246846	722	342
3.16	95	28122	1562	18
3.16	378	110817	1049	106
4.76	28	8672	1897	5
4.76	55	18273	1713	11
4.76	111	38476	1570	25
9.52	3	1158	1737	1
9.52	7	2566	2309	2
9.52	14	6488	1747	4

Chapter 6

Conclusions

In this thesis, a computer program was developed to simulate the Spex 8000, a ball milling machine, using the Event Driven Method. This program is a new tool aimed to help in the research of mechanical alloying by ball milling. The program is also useful in finding the optimal operation of the equipment. The program is written in such a way that it is easily adaptable to other uses or devices. The correctness of the developed program was verified. Several quality tests have been performed to guarantee that the numerical computations are performed as intended. The program includes few outputs designed for quality purposes.

The main drawback of the developed program, like any other program for simulation, is that its accuracy is limited to the accuracy of the underlying theoretical models implemented. The numerical results shown in chapter 5 agree only partially with the experimental results. Although the Newtonian approach to ball collisions is correct, the use of a single coefficient of restitution may be restrictive to accurately model the collision process. In real devices, the amount of powder that is present between two balls colliding is variable, which has been shown to affect the collision response. Variables such as elastic limit, impact velocity, incidence angle, and rotational speed also need to be included for a better fidelity to the real dynamics. In the future, a more realistic collision model will need to be developed and tested.

Nevertheless, the EDM proved to be a robust technique for simulation of colliding balls. The EDM is a feasible technique in both computational and technical aspects. The literature recognizes the EDM as a faster method than DEM. Unfortunately, it was not possible to gain all the advantages offered by EDM, which was one of the goals of this research program. The EDM could not be applied in a straightforward way to the specific case of the Spex 8000. The ball-vial collision detection could not be solved analytically because of the complex kinematics of the vial. Instead, a numerical method had to be

deployed. The collision time was found by steps like in the DEM. As a result, the use of a numerical method added an overhead to the simulation diminishing the simplicity and speed natural to the EDM. Also, collision times fell in multiples of the time step which resulted in simultaneous collisions. On the other hand, EDM, by definition, processes only one event at a time. A modification had to be added to the program to overcome this last limitation. Once again, the program lost simplicity and speed. On the other hand, ball-ball collision detection could be easily implemented. Briefly, the developed program can successfully simulate the Spex 8000 by EDM; but, the performance of the program is much lower than was intended.

From the point of view of the MA, the output of the software agrees with the experimental data presented by Ward. The optimal point of operation of the Spex 8000 is reached with larger charge ratios, and that means larger diameters or number of balls. More simulations are required to conclude what might be the optimal diameter or the optimal number of balls.

Future research

It is recommended that several additional studies may be carried out in the future:

1. The COR for ball collisions involving powder needs further investigation. The presence of powder surrounding the balls may affect significantly the collision response. For example, the impact velocity has low relevance in collisions with powder (Huang et al. 1998) compared to collisions without powder (Ramirez et al. 1999). The incidence angle and rotational speed play a role in impacts without powder (Dong and Moys 2006) and their effect on impacts involving powder could be also investigated. Current models of the COR do not take account of the elastic limit of the material. It is assumed that every collision involves powder transformation; however, in low impact collisions, the stress is not sufficiently high to permit plastic deformation.

2. A new modelling method is proposed. Small adjustments were made to apply the EDM to the Spex 8000. Those modifications reduced both the performance and readability of the program. A hybrid method that combines characteristics of the EDM and DEM is proposed. The idea is to keep the collision response and the ball-ball collision prediction of the EDM, and introduce the ball-vial collision prediction using the DEM approach. The system progresses in time in a combination of time steps and ball-ball collision times. The model could also be improved if

simultaneous collisions are accepted; but, the limitation of the collision between only two bodies is maintained. To move the vial-ball system in time steps not only allows the detection of simultaneous ball-vial collisions but also avoids the inefficiency of the numerical method used to predict the ball-vial collision for a single ball.

3. Some aspects of the developed program can be improved. For example, a graphical interface (GUI) could be implemented. The program can also be modified to perform computations in parallel, and hence, make use of the multiprocessing capabilities of today's computers.

4. The study of optimization can be extended to other variables such as the geometry and size of the vial. It was noticed in some experimental studies that some portions of the powder are left unprocessed in the internal edges of the vial (Ward et al. 2005). Rounded edges are preferred to overcome this limitation; but, the effect of the size and shape of the vial on the efficiency of the process is yet to be determined. A partial effect of the vial length in the Spex 8000 has been found numerically by DEM (Prasad and Theuerkauf 2009); but, experimental verification is needed.

5. Another topic of study could be the realization of a comparative study of the performance and accuracy of different models and/or simulation software involving MA equipment. This will be a great contribution to the state of art of the simulation of MA. It will be interesting to see how the program developed in this study is ranked among all other software.

6. The EDM can be combined with local models to take account of micro-structural changes. This is something new in the simulation of the MA. The amount of computer power available today makes it possible. Moreover, the developed software is expansible and can easily accept these kind of improvements. However, this new generation of modelling requires extensive experimental research, and therefore, it may take a long time before an accurate theoretical model is available.

Bibliography

B. J. Alder and T. E. Wainwright. Studies in molecular dynamics. 1. general method. *Journal of Chemical Physics*, 31(2):459–466, 1959. URL http://dx.doi.org/10.1063/1.1730376.

J. S. Benjamin. Mechanical alloying. *Scientific American*, 234(5):40–49, 1976.

N. V. Brilliantov, F. Spahn, J. Hertzsch, and T. Pöschel. Model for collisions in granular gases. *Physical Review E.*, 53(5):5382–5392, May 1996. URL http://dx.doi.org/10.1103/PhysRevE.53.5382.

C. Caravati, F. Delogu, G. Cocco, and M. Rustici. Hyperchaotic qualities of the ball motion in a ball milling device. *Chaos*, 9(1):219–226, 1999. URL http://dx.doi.org/10.1063/1.166393.

V. Chawla, S. Prakash, and B. S. Sidhu. State of the art: Applications of mechanically alloyed nanomaterials - a review. *Matererials and Manufactoring Processes*, 22(4):469–473, 2007. ISSN 1042-6914. URL http://dx.doi.org/10.1080/10426910701235900.

A. Concas, N. Lai, M. Pisu, and G. Cao. Modelling of comminution processes in spex mixer/mill. *Chemical Engineering Science*, 61(11):3746–3760, 2006. URL http://dx.doi.org/10.1016/j.ces.2006.01.007.

R. M. Davis, B. McDermott, and C. C. Koch. Mechanical alloying of brittle materials. *Metallurgical and Materials Transactions*, 19(12):2867–2874, 1988. URL http://dx.doi.org/10.1007/BF02647712.

F. Delogu. A combined experimental and numerical approach to the kinetics of mechanically induced phase transformations. *Acta Materialia*, 56(4):905–912, February 2008. URL http://dx.doi.org/10.1016/j.actamat.2007.10.041.

H. Dong and M.H. Moys. Experimental study of oblique impacts with initial spin.

Powder Technology, 161(1):22 – 31, 2006. ISSN 0032-5910. URL http://dx.doi.org/10.1016/j.powtec.2005.05.046.

RA Dunlap, DA Small, GR MacKay, JW O'Brien, JR Dahn, and ZH Cheng. Materials preparation by ball milling. *Canadian Journal of Physics*, 78(3):211–229, 2000. ISSN 0008-4204. URL http://dx.doi.org/10.1139/cjp-78-3-211.

M.S. El-Eskandarany. *Mechanical Alloying for fabrication of advanced engineering materials*. William Andrew Publishing/Noyes, Norwich, NY (USA), 2001.

E. Gaffet and F. Bernard. Mechanically activated powder metallurgy processing: A versatile way towards nanomaterials synthesis. *Annales De Chimie-science Des Materiaux*, 27(6):47–59, 2002. URL http://dx.doi.org/10.1016/S0151-9107(02)90014-0.

H. Huang, M. P. Dallimore, J. Pan, and P. G. McCormick. An investigation of the effect of powder on the impact characteristics between a ball and a plate using free falling experiments. *Materials Science and Engineering*, 241(1-2):38–47, January 1998. URL http://dx.doi.org/10.1016/S0921-5093(97)00470-X.

B. B. Khina and F. H. Froes. Modeling mechanical alloying: Advances and challenges. *Jom (A journal of the Minerals Metals & Materials Society)*, 48(7):36–38, July 1996.

C. Koch, R. Scattergood, K. Youssef, E. Chan, and Y. Zhu. Nanostructured materials by mechanical alloying: new results on property enhancement. *Journal of Materials Science*, 45(17):4725–4732, 2010. ISSN 0022-2461. URL http://dx.doi.org/10.1007/s10853-010-4252-7.

A. J. Lynch and C. A. Rowland. *The History of Grinding*. Society for Mining Metallurgy & Exploration, Littleton, Colorado (USA), 2005. URL http://www.smenet.org/store/mining-books.cfm/History-of-Grinding/238-6E.

A. Makino. Fundamental aspects of the heterogeneous flame in the self-propagating high-temperature synthesis (shs) process. *Progress in Energy and Combustion Science*, 27(1):1–74, 2001. URL http://dx.doi.org/10.1016/S0360-1285(00)00004-6.

A Merzhanov. Fundamentals, achievements, and perspectives for development of solid-flame combustion. *Russian Chemical Bulletin*, 46(1):8–32, 1997. URL http://dx.doi.org/10.1007/BF02495340.

A Merzhanov. The chemistry of self-propagating high-temperature synthesis. *Journal of Materials Chemistry*, 14:1779–1786, 2004. URL http://dx.doi.org/10.1039/B401358C.

S. Meyers. *Effective C++: 55 Specific ways to improve your programs and designs*. Addison-Wesley Professional, Upper Saddle River, NJ (USA), 3rd edition, 2005.

S. Miller and S. Luding. Event-driven molecular dynamics in parallel. *Journal of Computational Physics*, 193(1):306–316, 2004. ISSN 0021-9991. URL http://dx.doi.org/10.1016/j.jcp.2003.08.009.

G. A. Nematollahi, E. Marzbanrad, and A. R. Aghaei. Molecular dynamics investigation of mechanical mixing in mechanical alloying. *Materials Science and Engineering A*, 492(1-2):455–459, 2008. URL http://dx.doi.org/10.1016/j.msea.2008.03.049.

T. Poschel and T. Schwager. *Computational granular dynamics - Models and algorithms*. Springer, Heidelberg (Germany), 2005.

D. V. N. Prasad and J. Theuerkauf. Effect of grinding media size and chamber length on grinding in a spex mixer mill. *Chemical Engineering & Technology*, 32(7):1102–1106, 2009. ISSN 1521-4125. URL http://dx.doi.org/10.1002/ceat.200900035.

R. Ramirez, T. Poschel, N. V. Brilliantov, and T. Schwager. Coefficient of restitution of colliding viscoelastic spheres. *Physical Review E*, 60(4):4465–4472, 1999. URL http://dx.doi.org/10.1103/PhysRevE.60.4465.

R. Reichardt and W. Wiechert. Event driven algorithms applied to a high energy ball mill simulation. *Granular Matter*, 9(3-4):251–266, 2007. URL http://dx.doi.org/10.1007/s10035-006-0034-y.

A. S. Rogachev and A. S. Mukasyan. Combustion of heterogeneous nanostructural systems (review). *Combustion, Explosion and Shock Waves*, 46(3):243–266, 2010. URL http://dx.doi.org/10.1007/s10573-010-0036-2.

Y. Shirakawa, Y. Hayashi, K. Kadota, H. Mio, H. Ohtsuki, A. Shimosaka, and J. Hidaka. Molecular dynamics simulations of structural disordering and forming defects in a milling process for selenium. *Journal of Nanoparticle Research*, 10(4):577–584, April 2008. URL http://dx.doi.org/10.1007/s11051-007-9287-6.

C. Suryanarayana. Mechanical alloying and milling. *Progress in Materials Science*, 46 (1-2):1–184, 2001. URL http://dx.doi.org/10.1016/S0079-6425(99)00010-9.

B. G. Tabachnick and L. S. Fidell. *Using multivariate statistics*. Pearson/Allyn & Bacon, Boston, 5th international edition, February 2007. ISBN 0205465250. URL www.abacon.com/tabachnick.

O. R. Walton and R. L. Braun. Viscosity, granular-temperature, and stress calculations for shearing assemblies of inelastic, frictional disks. *Journal of Rheology*, 30(5):949–980, October 1986. URL http://dx.doi.org/10.1122/1.549893.

T. S. Ward, W. L. Chen, M. Schoenitz, R. N. Dave, and E. L. Dreizin. A study of mechanical alloying processes using reactive milling and discrete element modeling. *Acta Materialia*, 53(10):2909–2918, 2005. URL http://dx.doi.org/10.1016/j.actamat.2005.03.006.

B. K. Yen, T. Aizawa, and J. Kihara. Synthesis and formation mechanisms of molybdenum silicides by mechanical alloying. *Materials Science and Engineering A-structural Materials Properties Microstructure and Processing*, 220(1-2):8–14, December 1996. URL http://dx.doi.org/10.1016/S0921-5093(96)10430-5.

Appendix A

Equations and mathematical derivations

A.1 Relevant vector properties

Commutative cross product

$$\vec{a} \times \vec{b} = -\vec{b} \times \vec{a} \tag{A.1}$$

The cross product of parallels vector yields to zero

$$\vec{a} \times \vec{b} = 0 \qquad if \quad \vec{a} \parallel \vec{b} \tag{A.2}$$

Commutative dot product and cross product

$$\vec{a} \cdot \vec{b} \times \vec{c} = \vec{a} \times \vec{b} \cdot \vec{c} \tag{A.3}$$

The square of the sum of two vectors is analog to the square of two scalars

$$(\vec{a} + \vec{b})^2 = \vec{a}^2 + 2\vec{a} \cdot \vec{b} + \vec{b}^2 \tag{A.4}$$

A vector can be expressed in terms of its normal and tangential components in reference to a normal direction.

$$\vec{a} = \vec{a}^n + \vec{a}^t \tag{A.5}$$

The normal component of a vector given the normal direction is

$$\vec{a}^{\,n} = (\vec{a} \cdot \vec{e}^{\,n}) \cdot \vec{e}^{\,n} \tag{A.6}$$

The tangential component of a vector given the normal direction

$$\vec{a}^{\,t} = \vec{e}^{\,n} \times (\vec{a} \times \vec{e}^{\,n}) \tag{A.7}$$

Or

$$\vec{a}^{\,t} = \vec{a} - \vec{a}^{\,n} \tag{A.8}$$

Operations like sum or subtraction can be performed component-wise. If $\vec{c} = \vec{a} \pm \vec{b}$ then

$$\begin{aligned} \vec{c}^{\,n} &= \vec{a}^{\,n} \pm \vec{b}^{\,n} \\ \vec{c}^{\,t} &= \vec{a}^{\,t} \pm \vec{b}^{\,t} \end{aligned} \tag{A.9}$$

The dot product of perpendicular components is zero,

$$\vec{a}^{\,n} \cdot \vec{b}^{\,t} = 0 \tag{A.10}$$

The magnitude of a vector can be computed by

$$|\vec{a}|^2 = \vec{a} \cdot \vec{a} \tag{A.11}$$

The magnitude of a vector also can be computed from its perpendicular components as

$$|\vec{a}|^2 = |\vec{a}^{\,n}|^2 + \left|\vec{a}^{\,t}\right|^2 \tag{A.12}$$

Last expression can be rewritten as the sum of the dot product of its components

$$\vec{a}^{\,2} = (\vec{a}^{\,n})^2 + (\vec{a}^{\,t})^2 \tag{A.13}$$

A.2 Free-flight trajectory of milling balls

This section describes the trajectory of the milling balls between collisions when the only force acting on the balls is the gravity. Milling balls are assumed to be perfect spheres with their centre of mass coincident with their geometrical centre. It is assumed that the trajectory of the balls are not affected by their rotation. The velocity of a particle is derived from the definition of acceleration as follows:

$$\vec{a} \equiv d\vec{v}/dt$$

where $\vec{a}$ is the acceleration vector, $\vec{v}$ is the velocity vector, t is an scalar for time. In this case, acceleration is the gravity and is constant. Integrating last equation with constant acceleration yields:

$$\vec{v} = \vec{v}_o + \vec{a}\Delta t \tag{A.14}$$

where $\vec{v}$ is the velocity after a the time span Δt, and $\vec{v}_o$ is the velocity at initial time. The position of a particle is obtained from

$$\vec{v} = d\vec{r}/dt$$

where $\vec{r}$ is the vector from the origin of coordinates to the centre of mass of the particle. Integrating last equation and making use of equation A.14

$$\vec{r} = \vec{r}_o + \vec{v}_o\Delta t + \tfrac{1}{2}\vec{a}\Delta t^2 \tag{A.15}$$

where $\vec{r_o}$ is the position of the particle at initial time.

A.3 Ball-ball collision prediction

The collision times are computed in agreement with the trajectory of the balls, given in equation A.15. A collision occurs when both particles are in contact with one another. This happens when the distance between their centres becomes equal to the sum of their radii, i.e.:

$$\mid (\vec{r_i} + \vec{v}_i\Delta t + \tfrac{1}{2}\vec{a}\Delta t^2) - (\vec{r_j} + \vec{v}_j\Delta t + \tfrac{1}{2}\vec{a}\Delta t^2) \mid = R_i + R_j \tag{A.16}$$

where R_i and R_j are the radii of the balls.

By grouping some terms, last equation becomes:

$$\mid \vec{r_{ij}} + \vec{v}_{ij}\Delta t \mid = R_{ij}$$

where:

$$\begin{aligned} \vec{r}_{ij} &\equiv \vec{r}_i - \vec{r}_j \\ \vec{v}_{ij} &\equiv \vec{v}_i - \vec{v}_j \\ R_{ij} &\equiv R_i + R_j \end{aligned}$$

The following equation is obtained when the terms are taken outside the absolute value,

$$(\vec{r_{ij}} + \vec{v}_{ij}\Delta t)^2 = R_{ij}^2$$

Or

$$\vec{v}_{ij}^2\Delta t^2 + 2\vec{r}_{ij} \cdot \vec{v}_{ij}\Delta t + \vec{r}_{ij}^2 - R_{ij}^2 = 0$$

The last equation is of quadratic form and can be solved for the collision time, yielding

$$\Delta t = \frac{-2\vec{r}_{ij} \cdot \vec{v}_{ij} \pm \sqrt{(2\vec{r}_{ij} \cdot \vec{v}_{ij})^2 - 4\vec{v}_{ij}^2(\vec{r}_{ij}^2 - R_{ij}^2)}}{2\vec{v}_{ij}^2}$$

Simplifiying,

$$\Delta t = \frac{-\vec{r}_{ij} \cdot \vec{v}_{ij} \pm \sqrt{(\vec{r}_{ij} \cdot \vec{v}_{ij})^2 - (\vec{r}_{ij}^2 - R_{ij}^2)\vec{v}_{ij}^2}}{\vec{v}_{ij}^2} \tag{A.17}$$

In the solution for the collision time, the lower positive time is used as the correct one. If no real solution is posible it means that the balls will not collide.

A.4 Ball-ball collision response

The model here came from section 3.2 of Poschel and Schwager (2005); but, the resulting equations are redefined here to include the mass and moment of inertia of each ball. The assumptions of the collision are mentioned in section 2.4. Equations 2.23, A.18, and 2.25 are known as the collision rules and are brought here:

$$\begin{aligned} \vec{g}_{ij} &= (\vec{v}_i - \vec{\omega_i} \times R_i\vec{e}_{ij}^n) - (\vec{v}_j + \vec{\omega}_j \times R_j\vec{e}_{ij}^n) \\ \vec{g}'_{ij} &= (\vec{v}'_i - \vec{\omega_i}' \times R_i\vec{e}_{ij}^n) - (\vec{v}'_j + \vec{\omega}'_j \times R_j\vec{e}_{ij}^n) \end{aligned} \tag{A.18}$$

$$(\vec{g}_{ij}^n)' = -\varepsilon^n\vec{g}_{ij}^n \qquad with \quad 0 \leq \varepsilon^n \leq 1 \tag{A.19}$$

$$(\vec{g}_{ij}^{\,t})' = \varepsilon^t \vec{g}_{ij}^{\,t} \qquad with \quad -1 \leq \varepsilon^t \leq 1 \tag{A.20}$$

$$\begin{aligned} m_i\vec{v}_i' + m_j\vec{v}_j' &= m_i\vec{v}_i + m_j\vec{v}_j \\ R_i\vec{e}_{ij}^{\,n} \times m_i\vec{v}_i' + J_i\vec{\omega}_i' &= R_i\vec{e}_{ij}^{\,n} \times m_i\vec{v}_i + J_i\vec{\omega}_i \\ R_j\vec{e}_{ij}^{\,n} \times m_j\vec{v}_j' - J_j\vec{\omega}_j' &= R_j\vec{e}_{ij}^{\,n} \times m_j\vec{v}_j - J_j\vec{\omega}_j \end{aligned} \tag{A.21}$$

where

m_i and m_j are the respectively masses of the spheres

J_i and J_j are the respectively moments of inertia

R_i and R_j are the respectively radius of the spheres

$\vec{v}_i$ and $\vec{v}_j$ are the respectively translational velocities of the spheres

$\vec{w}_i$ and $\vec{w}_j$ are the respectively rotational velocities of the spheres

$\vec{g}_{ij}$ is the relative velocity at the contact point

The prime in the previous symbols stands for the variable after collision.

The normal and tangential components of the contact velocity in equation A.18 can be determined with the help of equations A.6 and A.8

$$\vec{g}_{ij}^{\,n} = \vec{v}_{ij}^{\,n} \tag{A.22}$$

$$\vec{g}_{ij}^{\,t} = \vec{v}_{ij}^{\,t} - (R_i\vec{\omega}_i + R_j\vec{\omega}_j) \times \vec{e}_{ij}^{\,n} \tag{A.23}$$

$$(\vec{g}_{ij}^{\,n})' = (\vec{v}_{ij}^{\,n})' \tag{A.24}$$

$$(\vec{g}_{ij}^{\,t})' = (\vec{v}_{ij}^{\,t})' - (R_i\vec{\omega}_i' + R_j\vec{\omega}_j') \times \vec{e}_{ij}^{\,n} \tag{A.25}$$

where

$$\vec{v}_{ij} = \vec{v}_i - \vec{v}_j \tag{A.26}$$

$$\vec{v}_{ij}' = \vec{v}_i' - \vec{v}_j' \tag{A.27}$$

Arranging equations A.21

$$\vec{v}_j' = \vec{v}_j - \frac{m_i}{m_j}(\vec{v}_i' - \vec{v}_i) \tag{A.28}$$

$$\vec{\omega}_i' = \vec{\omega}_i - \frac{m_i R_i}{J_i} \vec{e}_{ij}^n \times (\vec{v}_i' - \vec{v}_i) \tag{A.29}$$

$$\vec{\omega}_j' = \vec{\omega}_j + \frac{m_j R_j}{J_j} \vec{e}_{ij}^n \times (\vec{v}_j' - \vec{v}_j) \tag{A.30}$$

It is easy to work with component directions since coefficients of restitution are defined in this way. Beginning in the normal direction and equation A.27

$$(\vec{v}_i^n)' = (\vec{v}_{ij}^n)' + (\vec{v}_j^n)'$$

Combining with equations A.24 and A.28

$$(\vec{v}_i^n)' = (\vec{g}_{ij}^n)' + \vec{v}_j^n - \frac{m_i}{m_j}[(\vec{v}_i^n)' - \vec{v}_i^n]$$

Combining with equation A.19 and introducing $\vec{v}_i^n - \vec{v}_i^n$, which is a null vector

$$(1 + \frac{m_i}{m_j})(\vec{v}_i^n)' = -\varepsilon^n \vec{g}_{ij}^n + \vec{v}_j^n - \vec{v}_i^n + (1 + \frac{m_i}{m_j})\vec{v}_i^n$$

Introducing equation A.26

$$(\vec{v}_i^n)' = \frac{-\varepsilon^n \vec{g}_{ij}^n - \vec{v}_{ij}^n}{1 + \frac{m_i}{m_j}} + \vec{v}_i^n$$

Introducing equation A.22:

$$(\vec{v}_i^n)' = \vec{v}_i^n - \frac{m_j}{m_i + m_j}(1 + \varepsilon^n)\vec{g}_{ij}^n$$

Finally, one of unknowns is derived by introducing the equivalent mass.

$$(\vec{v}_i^n)' = \vec{v}_i^n - \frac{M_{ij}}{m_i}(1 + \varepsilon^n)\vec{g}_{ij}^n \tag{A.31}$$

where M is the equivalent mass defined by

$$M_{ij} \equiv \frac{m_i m_j}{m_i + m_j} \tag{A.32}$$

The tangential component of the velocity of the particle i can be obtained from equation A.25

$$(\vec{v}_i^{\,t})' = (\vec{v}_j^{\,t})' + (\vec{g}_{ij}^{\,t})' + (R_i\vec{\omega}_i' + R_j\vec{\omega}_j') \times \vec{e}_{ij}^{\,n}$$

Introducing the tangential component from the coefficient of restitution, equation A.20, and the equation from conservation of angular momentum, equations A.29 and A.30, into previous equation:

$$(\vec{v}_i^{\,t})' = (\vec{v}_j^{\,t})' + \varepsilon^t\vec{g}_{ij}^{\,t} + [R_i\vec{\omega}_i - \frac{m_iR_i^2}{J_i}\vec{e}_{ij}^{\,n} \times (\vec{v}_i' - \vec{v}_i) + R_j\vec{\omega}_j + \frac{m_jR_j^2}{J_j}\vec{e}_{ij}^{\,n} \times (\vec{v}_j' - \vec{v}_j)] \times \vec{e}_{ij}^{\,n}$$

With the reduced moment of inertia $\tilde{J} \equiv J/mR^2$ and the relation A.7

$$(\vec{v}_i^{\,t})' = (\vec{v}_j^{\,t})' + \varepsilon^t\vec{g}_{ij}^{\,t} + (R_i\vec{\omega}_i + R_j\vec{\omega}_j) \times \vec{e}_{ij}^{\,n} - \frac{1}{\tilde{J}_i}(\vec{v}_i' - \vec{v}_i)^t + \frac{1}{\tilde{j}_j}(\vec{v}_j' - \vec{v}_j)^t$$

Combining with equation A.28

$$(\vec{v}_i^{\,t})' = \varepsilon^t\vec{g}_{ij}^{\,t} + (R_i\vec{\omega}_i + R_j\vec{\omega}_j) \times \vec{e}_{ij}^{\,n} - \frac{1}{\tilde{J}_i}(\vec{v}_i^{\,t})' + \frac{1}{\tilde{J}_i}\vec{v}_i^{\,t} + (1 + \frac{1}{\tilde{j}_j})[\vec{v}_j^{\,t} - \frac{m_i}{m_j}((\vec{v}_i^{\,t})' - \vec{v}_i^{\,t})] - \frac{1}{\tilde{j}_j}\vec{v}_j^{\,t}$$

And introducing $\vec{v}_i^{\,t} - \vec{v}_i^{\,t}$, which is a null vector

$$[1 + \frac{1}{\tilde{J}_i} + \frac{m_i}{m_j}(1 + \frac{1}{\tilde{j}_j})](\vec{v}_i^{\,t})' = \varepsilon^t\vec{g}_{ij}^{\,t} + (R_i\vec{\omega}_i + R_j\vec{\omega}_j) \times \vec{e}_{ij}^{\,n} + [1 + \frac{1}{\tilde{J}_i} + \frac{m_i}{m_j}(1 + \frac{1}{\tilde{j}_j})]\vec{v}_i^{\,t} - \vec{v}_{ij}^{\,t}$$

Introducing equation A.23

$$(\vec{v}_i^{\,t})' = \vec{v}_i^{\,t} - \frac{1 - \varepsilon^t}{m_i(\frac{1}{m_i} + \frac{1}{m_i\tilde{J}_i} + \frac{1}{m_j} + \frac{1}{m_j\tilde{J}_j})}\vec{g}_{ij}^{\,t}$$

Introducing equation A.32

$$(\vec{v}_i^{\,t})' = \vec{v}_i^{\,t} - \frac{M_{ij}}{m_i}\frac{1 - \varepsilon^t}{(1 + \frac{M_{ij}}{m_i\tilde{J}_i} + \frac{M_{ij}}{m_j\tilde{J}_j})}\vec{g}_{ij}^{\,t} \tag{A.33}$$

The vector $\vec{v}_i'$ is obtaining by combining its components which are defined by equations A.31 and A.33

$$\vec{v}_i' = \vec{v}_i - \frac{M_{ij}}{m_i}[(1+\varepsilon^n)\vec{g}_{ij}^n + \frac{1-\varepsilon^t}{1+\frac{M_{ij}}{m_i\tilde{J}_i}+\frac{M_{ij}}{m_j\tilde{J}_j}}\vec{g}_{ij}^t] \tag{A.34}$$

The vector $\vec{v}_j'$ is obtaining by combining equation A.28 and previous equation

$$\vec{v}_j' = \vec{v}_j + \frac{M_{ij}}{m_j}[(1+\varepsilon^n)\vec{g}_{ij}^n + \frac{1-\varepsilon^t}{1+\frac{M_{ij}}{m_i\tilde{J}_i}+\frac{M_{ij}}{m_j\tilde{J}_j}}\vec{g}_{ij}^t] \tag{A.35}$$

Angular velocity $\vec{\omega}_i'$ is obtaining by combining equations A.29 and A.34 and using the relation A.2 and the reduced moment of inertia $\tilde{J} \equiv J/mR^2$ a

$$\vec{\omega}_i' = \vec{\omega}_i + \frac{M_{ij}}{m_i R_i \tilde{J}_i}(\frac{1-\varepsilon^t}{1+\frac{M_{ij}}{m_i\tilde{J}_i}+\frac{M_{ij}}{m_j\tilde{J}_j}})(\vec{e}_{ij}^n \times \vec{g}_{ij}^t) \tag{A.36}$$

Angular velocity $\vec{\omega}_j'$ is obtaining by combining equations A.30 and A.35

$$\vec{\omega}_j' = \vec{\omega}_j + \frac{M_{ij}}{m_j R_j \tilde{J}_j}(\frac{1-\varepsilon^t}{1+\frac{M_{ij}}{m_i\tilde{J}_i}+\frac{M_{ij}}{m_j\tilde{J}_j}})(\vec{e}_{ij}^n \times \vec{g}_{ij}^t) \tag{A.37}$$

Simplifying

$$\vec{v}_i' = \vec{v}_i + \Delta\vec{v}_i \tag{A.38}$$

$$\vec{v}_j' = \vec{v}_j + \Delta\vec{v}_j \tag{A.39}$$

$$\vec{\omega}_i' = \vec{\omega}_i + \Delta\vec{\omega}_i \tag{A.40}$$

$$\vec{\omega}_j' = \vec{\omega}_j + \Delta\vec{\omega}_j \tag{A.41}$$

where

$$\Delta\vec{v}_i = -\frac{M_{ij}}{m_i}[A\vec{g}_{ij}^n + B\vec{g}_{ij}^t] \tag{A.42}$$

$$\Delta\vec{v}_j = \frac{M_{ij}}{m_j}(A\vec{g}_{ij}^n + B\vec{g}_{ij}^t) \tag{A.43}$$

$$\Delta\vec{\omega}_i = \frac{M_{ij}}{m_i R_i \tilde{J}_i}(\vec{e}_{ij}^{\,n} \times B\vec{g}_{ij}^{\,t}) \tag{A.44}$$

$$\Delta\vec{\omega}_j = \frac{M_{ij}}{m_j R_j \tilde{J}_j}(\vec{e}_{ij}^{\,n} \times B\vec{g}_{ij}^{\,t}) \tag{A.45}$$

$$A = (1 + \varepsilon^n) \tag{A.46}$$

$$B = \frac{1 - \varepsilon^t}{1 + \dfrac{M_{ij}}{m_i \tilde{J}_i} + \dfrac{M_{ij}}{m_j \tilde{J}_j}} \tag{A.47}$$

A.5 Change of energy of particles after a ball-ball collision

The change of kinetic energy (KE) is the difference between the KE after collision and the KE before. Beginning with particle i

$$\Delta E_{K,i} = KE'_{K,i} - KE_{K,i}$$

The kinetic energy is the sum of the translational and rotational, therefore

$$\Delta E_{K,i} = [\tfrac{1}{2} m_i (\vec{v}_i')^2 + \tfrac{1}{2} J_i (\vec{\omega}_i')^2] - [\tfrac{1}{2} m_i \vec{v}_i^{\,2} + \tfrac{1}{2} J_i \vec{\omega}_i^{\,2}]$$

Using the reduced moment of inertia $\tilde{J} \equiv J/mR^2$

$$\Delta E_{K,i} = \tfrac{1}{2} m_i \{(\vec{v}_i')^2 - \vec{v}_i^{\,2} + \tilde{J}_i R_i^2 [(\vec{\omega}_i')^2 - \vec{\omega}_i^{\,2}]\}$$

The change of velocities, equations A.38 and A.40, allow to define this change of kinetic energy with terms before collision.

$$\Delta E_{K,i} = \tfrac{1}{2} m_i \{(\vec{v}_i + \Delta\vec{v}_i)^2 - (\vec{v}_i)^2 + \tilde{J}_i R_i^2 [(\vec{\omega}_i + \Delta\vec{w}_i)^2 - \vec{\omega}_i^{\,2}]\}$$

$$\Delta E_{K,i} = \tfrac{1}{2} m_i [2\vec{v}_i \Delta\vec{v}_i + \Delta\vec{v}_i^{\,2} + \tilde{J}_i R_i^2 (2\vec{\omega}_i \Delta\vec{w}_i + \Delta\vec{w}_i^{\,2})] \tag{A.48}$$

where Δv_i is defined in equation A.42 and

$$\Delta\vec{v}_i^{\,2} = \frac{M_{ij}^2}{m_i^2}[A^2(\vec{g}_{ij}^{\,n})^2 + B^2(\vec{g}_{ij}^{\,t})^2] \tag{A.49}$$

$$\Delta\vec{w}_i^2 = \frac{M_{ij}^2 B^2}{m_i^2 \tilde{J}_i^{\,2} R_i^2}(\vec{g}_{ij}^{\,t})^2 \tag{A.50}$$

Introducing equations A.42, A.44, A.49, and A.50 into equation A.48.

$$\begin{aligned}\Delta E_{K,i} \quad = \tfrac{1}{2} m_i \{ \quad & -\frac{2M_{ij}}{m_i}\vec{v}_i(A\vec{g}_{ij}^{\,n} + B\vec{g}_{ij}^{\,t}) + \frac{M_{ij}^2}{m_i^2}[A^2(\vec{g}_{ij}^{\,n})^2 + B^2(\vec{g}_{ij}^{\,t})^2] \\ & + \tilde{J}_i R_i^2 [\frac{2M_{ij}}{m_i \tilde{J}_i R_i}\vec{\omega}_i(\vec{e}_{ij}^{\,n} \times B\vec{g}_{ij}^{\,t}) + \frac{M_{ij}^2 B^2}{m_i^2 \tilde{J}_i^2 R_i^2}(\vec{g}_{ij}^{\,t})^2] \quad \}\end{aligned}$$

Finally,

$$\Delta E_{K,i} = \tfrac{1}{2} M_{ij}\{-2\vec{v}_i(A\vec{g}_{ij}^{\,n} + B\vec{g}_{ij}^{\,t}) + \frac{M_{ij}}{m_i}[A^2(\vec{g}_{ij}^{\,n})^2 + B^2(1 + \frac{1}{\tilde{J}_i})(\vec{g}_{ij}^{\,t})^2] + 2R_i B\vec{\omega}_i(\vec{e}_{ij}^{\,n} \times \vec{g}_{ij}^{\,t})\} \tag{A.51}$$

Similarly, it is possible to compute the change of kinetic energy for the other particle. For particle j is the same equation but using the j subscript instead of i

$$\Delta E_{K,j} = \tfrac{1}{2} m_j [2\vec{v}_j \Delta\vec{v}_j + \Delta\vec{v}_j^{\,2} + \tilde{J}_j R_j^2 (2\vec{\omega}_j \Delta\vec{w}_j + \Delta\vec{w}_j^{\,2})] \tag{A.52}$$

where Δv_j is defined in equation A.43 and

$$\Delta\vec{v}_j^{\,2} = \frac{M_{ij}^2}{m_j^2}[A^2(\vec{g}_{ij}^{\,n})^2 + B^2(\vec{g}_{ij}^{\,t})^2] \tag{A.53}$$

$$\Delta\vec{w}_j^{\,2} = \frac{M_{ij}^2 B^2}{m_j^2 \tilde{J}_j^2 R_j^2}(\vec{g}_{ij}^{\,t})^2 \tag{A.54}$$

Introducing equations A.43, A.45, A.53, and A.54 into equation A.52.

$$\begin{aligned}\Delta E_{K,j} \quad = \tfrac{1}{2} m_j \{ \quad & \frac{2M_{ij}\vec{v}_j}{m_j}(A\vec{g}_{ij}^{\,n} + B\vec{g}_{ij}^{\,t}) + \frac{M_{ij}^2}{m_j^2}[A^2(\vec{g}_{ij}^{\,n})^2 + B^2(\vec{g}_{ij}^{\,t})^2] \\ & + \tilde{J}_j R_j^2 [2\vec{\omega}_j (\frac{M_{ij}}{m_j R_j \tilde{J}_j}(\vec{e}_{ij}^{\,n} \times B\vec{g}_{ij}^{\,t})) + \frac{M_{ij}^2 B^2}{m_j^2 \tilde{J}_j^2 R_j^2}(\vec{g}_{ij}^{\,t})^2] \quad \}\end{aligned}$$

Finally,

$$\Delta E_{K,j} = \tfrac{1}{2} M_{ij}\{2\vec{v}_j(A\vec{g}_{ij}^{\,n} + B\vec{g}_{ij}^{\,t}) + \frac{M_{ij}}{m_j}[A^2(\vec{g}_{ij}^{\,n})^2 + B^2(1+\frac{1}{\tilde{J}_j})(\vec{g}_{ij}^{\,t})^2] + 2R_j B\vec{\omega}_j(\vec{e}_{ij}^{\,n} \times \vec{g}_{ij}^{\,t})\} \tag{A.55}$$

A.6 Computation of the milling dose for the ball-ball collision

Starting with the conservation of energy equation and assuming no other losses than the milling dose.

$$E_{K,T} = E'_{K,T} + D_{ij}$$

where $E_{K,T}$ and $E'_{K,T}$ are the total kinetic energy before and after collision respectively, and D_{ij} is the milling dose. For two particles i and j the milling dose is

$$D_{ij} = (E_{K,i} + E_{K,j}) - (E'_{K,i} + E'_{K,j}) \tag{A.56}$$

In terms of the change of energy per particle

$$D_{ij} = -\Delta E_{K,i} - \Delta E_{K,j}$$

Introducing equations A.51 and A.55

$$\begin{aligned} D_{ij} = &-\tfrac{1}{2} M_{ij}\{-2\vec{v}_i(A\vec{g}_{ij}^{\,n} + B\vec{g}_{ij}^{\,t}) + \frac{M_{ij}}{m_i}[A^2(\vec{g}_{ij}^{\,n})^2 + B^2(1+\frac{1}{\tilde{J}_i})(\vec{g}_{ij}^{\,t})^2] + 2R_i B\vec{\omega}_i(\vec{e}_{ij}^{\,n} \times \vec{g}_{ij}^{\,t})\} \\ &-\tfrac{1}{2} M_{ij}\{2\vec{v}_j(A\vec{g}_{ij}^{\,n} + B\vec{g}_{ij}^{\,t}) + \frac{M_{ij}}{m_j}[A^2(\vec{g}_{ij}^{\,n})^2 + B^2(1+\frac{1}{\tilde{J}_j})(\vec{g}_{ij}^{\,t})^2] + 2R_j B\vec{\omega}_j(\vec{e}_{ij}^{\,n} \times \vec{g}_{ij}^{\,t})\} \end{aligned}$$

Arranging

$$\begin{aligned} D_{ij} \;= \tfrac{1}{2} M_{ij}\{ \;& 2(\vec{v}_i - \vec{v}_j)(A\vec{g}_{ij}^{\,n} + B\vec{g}_{ij}^{\,t}) - (\frac{1}{m_i} + \frac{1}{m_j}) M_{ij} A^2(\vec{g}_{ij}^{\,n})^2 - (\frac{1}{m_i} + \frac{1}{m_j}) M_{ij} B^2(\vec{g}_{ij}^{\,t})^2 \\ & -(\frac{1}{m_i\tilde{J}_i} + \frac{1}{m_j\tilde{J}_j}) M_{ij} B^2(\vec{g}_{ij}^{\,t})^2] - 2B(R_i\vec{\omega}_i + R_j\vec{\omega}_j)(\vec{e}_{ij}^{\,n} \times \vec{g}_{ij}^{\,t}) \quad \} \end{aligned}$$

Using equation A.26, A.32, and A.3

$$D_{ij} = \tfrac{1}{2}M_{ij}\{ \quad 2\vec{v}_{ij}(A\vec{g}^n_{ij} + B\vec{g}^t_{ij}) - A^2(\vec{g}^n_{ij})^2 - B^2(\vec{g}^t_{ij})^2 - (\frac{1}{m_i\tilde{J}_i} + \frac{1}{m_j\tilde{J}_j})M_{ij}B^2(\vec{g}^t_{ij})^2$$
$$-2B(R_i\vec{\omega}_i + R_j\vec{\omega}_j) \times \vec{e}^n_{ij}\vec{g}^t_{ij} \quad \}$$

$\vec{v}_{ij}$ is expanded to its components $\vec{v}_{ij} = \vec{v}^n_{ij} + \vec{v}^t_{ij}$, and using the relation A.10

$$D_{ij} = \tfrac{1}{2}M_{ij}\{ \quad 2A\vec{v}^n_{ij}\vec{g}^n_{ij} + 2B\vec{v}^t_{ij}\vec{g}^t_{ij} - A^2(\vec{g}^n_{ij})^2 - B^2(\vec{g}^t_{ij})^2 - (\frac{1}{m_i\tilde{J}_i} + \frac{1}{m_j\tilde{J}_j})M_{ij}B^2(\vec{g}^t_{ij})^2$$
$$-2B(R_i\vec{\omega}_i + R_j\vec{\omega}_j) \times \vec{e}^n_{ij}\vec{g}^t_{ij} \quad \}$$

Introducing equation A.22

$$D_{ij} = \tfrac{1}{2}M_{ij}\{2A\vec{g}^n_{ij}\vec{g}^n_{ij} - A^2(\vec{g}^n_{ij})^2 - (1 + \frac{M_{ij}}{m_i\tilde{J}_i} + \frac{M_{ij}}{m_j\tilde{J}_j})B^2(\vec{g}^t_{ij})^2 + 2B[\vec{v}^t_{ij} - (R_i\vec{\omega}_i + R_j\vec{\omega}_j) \times \vec{e}^n_{ij}]\vec{g}^t_{ij}\}$$

Introducing equation A.23

$$D_{ij} = \tfrac{1}{2}M_{ij}\{A(2-A)(\vec{g}^n_{ij})^2 + [2 - (1 + \frac{M_{ij}}{m_i\tilde{J}_i} + \frac{M_{ij}}{m_j\tilde{J}_j})B]B(\vec{g}^t_{ij})^2\}$$

Replacing A and B by their actual values from equations A.46 and A.47 respectively

$$D_{ij} = \tfrac{1}{2}M_{ij}\{ \quad (1+\varepsilon^n)[2 - (1+\varepsilon^n)](\vec{g}^n_{ij})^2$$
$$+[2 - (1 + \frac{M_{ij}}{m_i\tilde{J}_i} + \frac{M_{ij}}{m_j\tilde{J}_j})(\frac{1-\varepsilon^t}{1 + \frac{M_{ij}}{m_i\tilde{J}_i} + \frac{M_{ij}}{m_j\tilde{J}_j}})]\frac{1-\varepsilon^t}{1 + \frac{M_{ij}}{m_i\tilde{J}_i} + \frac{M_{ij}}{m_j\tilde{J}_j}}(\vec{g}^t_{ij})^2 \quad \}$$

Finally[1]

$$D_{ij} = \tfrac{1}{2}M_{ij}\{[1 - (\varepsilon^n)^2](\vec{g}^n_{ij})^2 + \frac{1 - (\varepsilon^t)^2}{1 + \frac{M_{ij}}{m_i\tilde{J}_i} + \frac{M_{ij}}{m_j\tilde{J}_j}}(\vec{g}^t_{ij})^2\} \qquad \text{(A.57)}$$

[1]This result is confirmed with the equation 7 of Brilliantov et al. (1996).

A.7 Ball-wall collision response

The assumptions of the collision are mentioned in section 2.4. Equations 2.23, A.58, and 2.31 are known as the collision rules and are brought here:

$$\begin{aligned} \vec{g}_{iw} &= (\vec{v}_i - \vec{\omega}_i \times R_i \vec{e}^{\,n}_{iw}) - (\vec{v}_w) \\ \vec{g}'_{iw} &= (\vec{v}'_i - \vec{\omega}'_i \times R_i \vec{e}^{\,n}_{iw}) - (\vec{v}_w) \end{aligned} \tag{A.58}$$

$$(\vec{g}^{\,n}_{iw})' = -\varepsilon^n_w \vec{g}^{\,n}_{iw} \qquad with \quad 0 \leq \varepsilon^n_w \leq 1 \tag{A.59}$$

$$(\vec{g}^{\,t}_{iw})' = \varepsilon^t_w \vec{g}^{\,t}_{iw} \qquad with \quad -1 \leq \varepsilon^t_w \leq 1 \tag{A.60}$$

$$R_i \vec{e}^{\,n}_{iw} \times m_i \vec{v}'_i + J_i \vec{\omega}'_i = R_i \vec{e}^{\,n}_{iw} \times m_i \vec{v}_i + J_i \vec{\omega}_i \tag{A.61}$$

where

The prime stands for the variable after collision.

m_i is the mass of the ball

R_i is the radius of the ball

J_i is the moment of inertia of the ball

$\vec{v}_i$ is the translational velocity of the ball

$\vec{w}_i$ is the rotational velocity of the ball

$\vec{e}^{\,n}_{ij}$ is the normal vector

The normal and tangential components of the contact velocity in equation A.58 can be determined with the help of equations A.6 and A.8:

$$\vec{g}^{\,n}_{iw} = \vec{v}^{\,n}_{iw} \tag{A.62}$$

$$\vec{g}^{\,t}_{iw} = \vec{v}^{\,t}_{iw} - \vec{\omega}_i \times R_i \vec{e}^{\,n}_{iw} \tag{A.63}$$

$$(\vec{g}^{\,n}_{iw})' = (\vec{v}^{\,n}_{iw})' \tag{A.64}$$

$$(\vec{g}^{\,t}_{iw})' = (\vec{v}^{\,t}_{iw})' - \vec{\omega}'_i \times R_i \vec{e}^{\,n}_{iw} \tag{A.65}$$

where

$$\vec{v}_{iw} = \vec{v}_i - \vec{v}_w \tag{A.66}$$

The velocity of the wall does not change with the collision, thus

$$\vec{v}_w' = \vec{v}_w \tag{A.67}$$

To compute the velocities after collision it is better to find their normal and tangential velocities first. Let's start with the normal component of the translational velocity of the sphere. Equation A.64 is a good departure point.

$$(\vec{v}_i^{\,n})' = (\vec{v}_w^{\,n})' + (\vec{g}_{iw}^{\,n})'$$

Combining last equation with equations A.67 and A.59

$$(\vec{v}_i^{\,n})' = \vec{v}_w^{\,n} - \varepsilon_w^n \vec{g}_{iw}^{\,n}$$

Introducing $\vec{v}_i^{\,n} - \vec{v}_i^{\,n}$, which is a null vector, and using equation A.66

$$(\vec{v}_i^{\,n})' = \vec{v}_i^{\,n} - \vec{v}_{iw}^{\,n} - \varepsilon_w^n \vec{g}_{iw}^{\,n}$$

Combining with equation A.62 we got the normal velocity of the particle i after collision

$$(\vec{v}_i^{\,n})' = \vec{v}_i^{\,n} - (1 + \varepsilon_w^n)\vec{v}_{iw}^{\,n}$$

Or

$$(\vec{v}_i^{\,n})' = \vec{v}_i^{\,n} - (1 + \varepsilon_w^n)\vec{g}_{iw}^{\,n} \tag{A.68}$$

Equation A.65 comes handily for the tangential component of the translational velocity

$$(\vec{v}_i^{\,t})' = (\vec{v}_w^{\,t})' + (\vec{g}_{iw}^{\,t})' + \vec{\omega}_i' \times R_i \vec{e}_{iw}^{\,n} \tag{A.69}$$

The angular velocity after collision can be obtained from equation A.61

$$\vec{\omega}_i' = \vec{\omega}_i - \frac{m_i R_i}{J_i} \vec{e}_{iw}^{\,n} \times (\vec{v}_i' - \vec{v}_i) \tag{A.70}$$

Introducing this last equation and equations A.67 and A.60 into equation A.69

$$(\vec{v}_i^{\,t})' = \vec{v}_w^{\,t} + \varepsilon_w^t \vec{g}_{iw}^{\,t} + [\vec{\omega}_i - \frac{m_i R_i}{J_i} \vec{e}_{iw}^{\,n} \times (\vec{v}_i' - \vec{v}_i)] \times R_i \vec{e}_{iw}^{\,n}$$

With the reduced moment of inertia $\tilde{J} \equiv J/mR^2$ and relation A.7

$$(\vec{v}_i^{\,t})' = \vec{v}_w^{\,t} + \varepsilon_w^t \vec{g}_{iw}^{\,t} + \vec{\omega}_i \times R_i \vec{e}_{iw}^{\,n} - \frac{1}{\tilde{J}_i}(\vec{v}_i' - \vec{v}_i)^t$$

Introducing $\vec{v}_i^{\,t} - \vec{v}_i^{\,t}$, which is a null vector

$$(1 + \frac{1}{\tilde{J}_i})(\vec{v}_i^{\,t})' = (1 + \frac{1}{\tilde{J}_i})\vec{v}_i^{\,t} + \varepsilon_w^t \vec{g}_{iw}^{\,t} - (\vec{v}_{iw}^{\,t} - \vec{\omega}_i \times R_i \vec{e}_{iw}^{\,n})$$

Combining with equation A.63

$$(\vec{v}_i^{\,t})' = \vec{v}_i^{\,t} - \frac{1 - \varepsilon_w^t}{1 + \frac{1}{\tilde{J}_i}} \vec{g}_{iw}^{\,t} \tag{A.71}$$

Finally, the vector $\vec{v}_i'$ is obtaining by combining its components which are defined by equations A.68 and A.71

$$\vec{v}_i' = \vec{v}_i - (1 + \varepsilon_w^n)\vec{g}_{iw}^{\,n} - \frac{\varepsilon_w^t - 1}{1 + \frac{1}{\tilde{J}_i}} \vec{g}_{iw}^{\,t} \tag{A.72}$$

The angular velocity $\vec{\omega}_i'$ is obtaining using equation A.70

$$\vec{\omega}_i' = \vec{\omega}_i - \frac{m_i R_i}{J_i} \vec{e}_{iw}^{\,n} \times (\vec{v}_i' - \vec{v}_i)$$

Introducing equation A.72

$$\vec{\omega}_i' = \vec{\omega}_i - \frac{m_i R_i}{J_i} \vec{e}_{iw}^{\,n} \times [-(1 + \varepsilon_w^n)\vec{g}_{iw}^{\,n} - \frac{1 - \varepsilon_w^t}{1 + \frac{1}{\tilde{J}_i}} \vec{g}_{iw}^{\,t}]$$

The product $\vec{e}_{iw}^{\,n} \times \vec{g}_{iw}^{\,n}$ is zero since both vectors are parallel, then with the reduced moment of inertia $\tilde{J} \equiv J/mR^2$

$$\vec{\omega}_i' = \vec{\omega}_i + \frac{(1 - \varepsilon_w^t)}{R_i(1 + \tilde{J}_i)} \vec{e}_{iw}^{\,n} \times \vec{g}_{iw}^{\,t} \tag{A.73}$$

Simplifying,

$$\vec{v}_i' = \vec{v}_i - C g_{iw}^n - D \vec{g}_{iw}^{\,t} \tag{A.74}$$

$$\vec{\omega}_i' = \vec{\omega}_i + \frac{1}{\tilde{J}_i R_i} \vec{e}_{iw}^{\,n} \times D \vec{g}_{iw}^{\,t} \tag{A.75}$$

where

$$C = (1 + \varepsilon_w^n) \tag{A.76}$$

$$D = \frac{1 - \varepsilon_w^t}{1 + \dfrac{1}{\tilde{J}_i}} \tag{A.77}$$

The post-collision velocities also can be expressed in terms of their change of velocities:

$$\vec{v}_i' = \vec{v}_i + \Delta\vec{v}_i$$

$$\vec{\omega}_i' = \vec{\omega}_i + \Delta\vec{\omega}_i$$

where

$$\Delta\vec{v}_i = -C\vec{g}_{iw}^n - D\vec{g}_{iw}^t \tag{A.78}$$

$$\Delta\vec{\omega}_i = \frac{1}{\tilde{J}_i R_i}\vec{e}_{iw}^n \times D\vec{g}_{iw}^t \tag{A.79}$$

A.8 Change of energy of a ball after it collides with a wall

The change of kinetic energy (KE) is the difference between the KE after and before collision

$$\Delta E_{K,i} = E'_{K,i} - E_{K,i}$$

The kinetic energy is the sum of the translational and rotational, therefore

$$\Delta E_{K,i} = [\tfrac{1}{2} m_i (\vec{v}_i')^2 + \tfrac{1}{2} J_i (\vec{\omega}_i')^2] - [\tfrac{1}{2} m_i \vec{v}_i^{\,2} + \tfrac{1}{2} J_i \vec{\omega}_i^{\,2}]$$

Using the reduced moment of inertia $\tilde{J} \equiv J/mR^2$ and the relation A.5

$$\Delta E_{K,i} = \tfrac{1}{2} m_i \{(\vec{v}_i^{\,n\prime})^2 + (\vec{v}_i^{\,t\prime})^2 - (\vec{v}_i^{\,n})^2 - (\vec{v}_i^{\,t})^2 + \tilde{J}_i R_i^2 [(\vec{\omega}_i')^2 - \vec{\omega}_i^{\,2}]\}$$

Introducing equations A.74 and A.75 and using the relation A.4

$$\Delta E_{K,i} = \tfrac{1}{2}m_i\{ \quad (\vec{v}_i^{\,n})^2 - 2C\vec{v}_i^{\,n}\vec{g}_{iw}^{\,n} + C^2(\vec{g}_{iw}^{\,n})^2 + (\vec{v}_i^{\,t})^2 - 2D\vec{v}_i^{\,t}\vec{g}_{iw}^{\,t} + D^2(\vec{g}_{iw}^{\,t})^2 - (\vec{v}_i^{\,n})^2$$
$$-(\vec{v}_i^{\,t})^2 + \tilde{J}_i R_i^2[\vec{\omega}_i^{\,2} + \frac{2}{\tilde{J}_i R_i}\vec{\omega}_i\vec{e}_{iw}^{\,n} \times D\vec{g}_{iw}^{\,t} + \frac{1}{\tilde{J}_i^{\,2} R_i^2}(\vec{e}_{iw}^{\,n} \times D\vec{g}_{iw}^{\,t})^2 - \vec{\omega}_i^{\,2}] \quad \}$$

Replacing $\vec{v}_i$ by $\vec{v}_{iw} + \vec{v}_w$ from equation A.66, and using the property A.3

$$\Delta E_{K,i} = \tfrac{1}{2}m_i\{ \quad -2C(\vec{v}_{iw}^{\,n} + \vec{v}_w^{\,n})\vec{g}_{iw}^{\,n} + C^2(\vec{g}_{iw}^{\,n})^2 - 2D(\vec{v}_{iw}^{\,t} + \vec{v}_w^{\,t})\vec{g}_{iw}^{\,t} + D^2(\vec{g}_{iw}^{\,t})^2$$
$$+2DR_i\vec{\omega}_i \times \vec{e}_{iw}^{\,n}\vec{g}_{iw}^{\,t} + \frac{D^2}{\tilde{J}_i}(\vec{g}_{iw}^{\,t})^2 \quad \}$$

v_w^n and $\vec{v}_w^{\,t}$ can be replaced by $\vec{v}_w$ in this last equation without affecting the result.

$$\Delta E_{K,i} = \tfrac{1}{2}m_i\{ \quad -2Cv_{iw}^n\vec{g}_{iw}^{\,n} - 2C\vec{v}_w\vec{g}_{iw}^{\,n} + C^2(\vec{g}_{iw}^{\,n})^2 - 2D(\vec{v}_{iw}^{\,t} - \vec{\omega} \times R_i\vec{e}_{iw}^{\,n})\vec{g}_{iw}^{\,t}$$
$$-2D\vec{v}_w\vec{g}_{iw}^{\,t} + D^2(\vec{g}_{iw}^{\,t})^2 + \frac{D^2}{\tilde{J}_i}(\vec{g}_{iw}^{\,t})^2 \quad \}$$

Introducing equations A.62 and A.63

$$\Delta E_{K,i} = -\tfrac{1}{2}m_i\{C(2-C)(\vec{g}_{iw}^{\,n})^2 + 2(C\vec{g}_{iw}^{\,n} + D\vec{g}_{iw}^{\,t})\vec{v}_w + D[2 - (1 + \frac{1}{\tilde{J}_i})D](\vec{g}_{iw}^{\,t})^2\}$$

Replacing C and D by their actual values from equations A.76 and A.77 respectively

$$\Delta E_{K,i} = -\tfrac{1}{2}m_i\{ \quad (1+\varepsilon_w^n)[2 - (1+\varepsilon_w^n)](\vec{g}_{iw}^{\,n})^2 + 2[(1+\varepsilon_w^n)\vec{g}_{iw}^{\,n} + \frac{1-\varepsilon_w^t}{1+\frac{1}{\tilde{J}_i}}\vec{g}_{iw}^{\,t}]\vec{v}_w$$
$$+\frac{1-\varepsilon_w^t}{1+\frac{1}{\tilde{J}_i}}[2 - (1+\frac{1}{\tilde{J}_i})(\frac{1-\varepsilon_w^t}{1+\frac{1}{\tilde{J}_i}})](\vec{g}_{iw}^{\,t})^2 \quad \}$$

Finally, we got the expected value

$$\Delta E_{K,i} = -\tfrac{1}{2}m_i\{[1-(\varepsilon_w^n)^2](\vec{g}_{iw}^n)^2 + 2[(1+\varepsilon_w^n)\vec{g}_{iw}^n + \frac{1-\varepsilon_w^t}{1+\frac{1}{\tilde{J}_i}}\vec{g}_{iw}^t]\vec{v}_w + \frac{1}{1+\frac{1}{\tilde{J}_i}}[1-(\varepsilon_w^t)^2](\vec{g}_{iw}^t)^2\} \tag{A.80}$$

Last equation can be simplifying by including after-collision terms

$$\Delta E_{K,i} = \tfrac{1}{2}m_i(2\vec{v}_w + \vec{g}_{iw} + \vec{g}'_{iw})\Delta\vec{v}_i$$

A.9 Energy transfer to or from the wall

A moving wall introduces or removes energy to or from the sphere. This energy along with the milling dose are the remaining unknowns; The number of unknowns is higher than the number of equations. To overcome this limitation the wall is set to rest and the sphere now moves with the relative velocity instead of the absolute one. With this change, the unknown energy from the wall is neglected and the equation becomes solvable.

Starting with the conservation of energy equation and assuming no other losses than the milling dose the following equation is obtained:

$$\tilde{E}_K = \tilde{E}'_K + D_v$$

where $\tilde{E}_K$ and $\tilde{E}'_K$ are the kinetic energy before and after collision respectively, and D_v is the milling dose.

The milling dose for one particle is

$$D_v = \tilde{E_{K,i}} - \tilde{E}'_{K,i}$$

The kinetic energy is the sum of both translational and rotational energies, therefore

$$D_v = [\tfrac{1}{2}m_i\vec{\tilde{v}}_i^2 + \tfrac{1}{2}J_i\vec{\tilde{\omega}}_i^2] - [\tfrac{1}{2}m_i(\vec{\tilde{v}}'_i)^2 + \tfrac{1}{2}J_i(\vec{\tilde{\omega}}'_i)^2]$$

In the relative frame of reference

$$\begin{aligned}\vec{\tilde{v}}_i &= \vec{v}_{iw} \\ \vec{\tilde{v}}_i' &= \vec{v}_{iw}' \\ \vec{\tilde{\omega}}_i &= \vec{\omega}_i \\ \vec{\tilde{\omega}}_i' &= \vec{\omega}_i'\end{aligned}$$

Then

$$D_v = [\tfrac{1}{2}m_i\vec{v}_{iw}^2 + \tfrac{1}{2}J_i\vec{\omega}_i^2] - [\tfrac{1}{2}m_i(\vec{v}_{iw}')^2 + \tfrac{1}{2}J_i(\vec{\omega}_i')^2]$$

Replacing $\vec{v}_{iw}$ by $\vec{v}_i - \vec{v}_w$ and $\vec{v}_{iw}'$ by $\vec{v}_i' - \vec{v}_w$

$$D_v = \tfrac{1}{2}m_i(\vec{v}_i^2 - 2\vec{v}_i\vec{v}_w + \vec{v}_w^2) + \tfrac{1}{2}J_i\vec{\omega}_i^2 - \tfrac{1}{2}m_i[(\vec{v}_i')^2 - 2\vec{v}_i'\vec{v}_w + \vec{v}_w^2] + \tfrac{1}{2}J_i(\vec{\omega}_i')^2$$

$$D_v = [\tfrac{1}{2}m_i\vec{v}_i^2 + \tfrac{1}{2}J_i\vec{\omega}_i^2] - [\tfrac{1}{2}m_i(\vec{v}_i')^2 + \tfrac{1}{2}J_i(\vec{\omega}_i')^2] + \tfrac{1}{2}m_i(-2\vec{v}_i\vec{v}_w + \vec{v}_w^2) - \tfrac{1}{2}m_i(-2\vec{v}_i'\vec{v}_w + \vec{v}_w^2)$$

$$D_v = E_K - E_K' + \tfrac{1}{2}m_i[2(\vec{v}_i' - \vec{v}_i)\vec{v}_w]$$

$$D_v = E_K - E_K' + m_i\vec{v}_w\Delta\vec{v}_i$$

Now, in absolute terms, the energy transferred by a wall, E_W, appears in the equation of energy balance.

$$E_K + E_W = E_K' + D_v \tag{A.81}$$

It can be seen clearly that:

$$E_W = m_i\vec{v}_w\Delta\vec{v}_i$$

Introducing equation A.78

$$E_W = -m_i\vec{v}_w(C\vec{g}_{iw}^n + D\vec{g}_{iw}^t)$$

Or

$$E_W = -m_i[(1+\varepsilon_w^n)\vec{g}_{iw}^n + \frac{1-\varepsilon_w^t}{1+\frac{1}{\tilde{J}_i}}\vec{g}_{iw}^t]\vec{v}_w \tag{A.82}$$

A.10 Computation of the milling dose for the ball-wall collision

The milling dose can be computed form equation A.81

$$D_v = E_W - \Delta E_K \tag{A.83}$$

The values of E_W andΔE_K can be obtained from equations A.82 and A.80 respectively

$$\begin{aligned} D_v &= -m_i[(1+\varepsilon_w^n)\vec{g}_{iw}^n + \frac{1-\varepsilon_w^t}{1+\frac{1}{\tilde{J}_i}}\vec{g}_{iw}^t]\vec{v}_w \\ &\quad +\tfrac{1}{2}m_i\{2[(1+\varepsilon_w^n)\vec{g}_{iw}^n + \frac{1-\varepsilon_w^t}{1+\frac{1}{\tilde{J}_i}}\vec{g}_{iw}^t]\vec{v}_w + (1-(\varepsilon_w^n)^2)(\vec{g}_{iw}^n)^2 + \frac{1-(\varepsilon_w^t)^2}{1+\frac{1}{\tilde{J}_i}}(\vec{g}_{iw}^t)^2\} \end{aligned}$$

Finally,

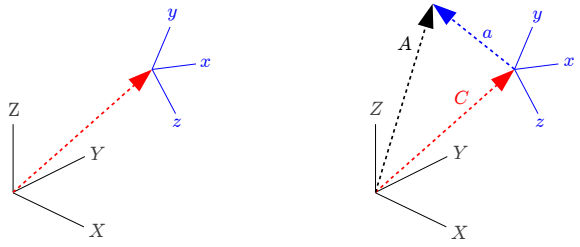
$$D_v = \tfrac{1}{2}m_i\{[1-(\varepsilon_w^n)^2](\vec{g}_{iw}^n)^2 + \frac{1-(\varepsilon_w^t)^2}{1+\frac{1}{\tilde{J}_i}}(\vec{g}_{iw}^t)^2\} \tag{A.84}$$

This equation resembles the one for the milling dose for the sphere-sphere case.

Appendix B

The roto-translational model (RTM)

The RTM is one of the tools from linear algebra that perform transformations between system of coordinates. The RTM is a mechanism to correlate two systems of coordinates by one single mathematical expression. One system is located and rotated in space using a second system as frame of reference. The first system is known as the relative system of coordinates whereas the second system is known as the absolute system of coordinates. They are also known as the non-inertial and inertial systems respectively when the absolute system is always fixed in space. This is the case in this study. The last set of names is the one used here. Figure B.1a illustrates this concept. The figure shows two Cartesian systems whose axis are identified with the usual letters x, y, z. The names in the inertial system are in capital letters whereas in the non-inertial system are in lower case. Figure B.1b shows a point in space referenced to the inertial and non-inertial systems of coordinates with the vectors $\vec{A}$ and $\vec{a}$ respectively.



(a) (b)

Figure B.1: a) A set of two correlated system of coordinates. b) A point in space in reference to the two coordinate systems.

The RTM consist of a matrix and a vector. The matrix is used to compute rotation and receives the name of rotation matrix. The vector is used to perform translational

transformations and is called translational vector. This explain how the RTM received its name.

One of the greatest advantages of the RTM is the ability to express the location of a point as the sum of a series of rotations and translations. Moreover, those transformations can be relatives or absolutes. Figure B.2 illustrates some possibilities of transformation with reference to figure B.1a.

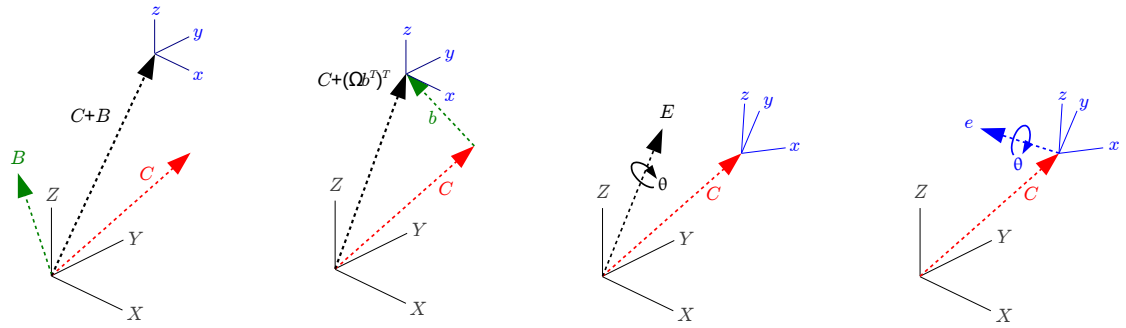


Figure B.2: Further relocation of relative systems of coordinates. a) Absolute translation, b) Relative translation, c) Absolute rotation, d) Relative rotation.

The rotation matrix is orthogonal, which means that is square with real entries whose columns and rows are orthogonal unit vectors. The rotation matrix in 3 dimensions is 3×3 in size. The rotation of a vector A using a rotation matrix $\bar{\Omega}$ is given by $(\bar{\Omega} \cdot A^T)^T$, where the superscript T stands for the transpose. Sometimes in the literature the rotation matrix is the inverse of the one used here. This creates a disagreement with the equations exposed here; however, if equations are adjusted accordingly the result will be the same. There are many ways to obtain the rotation matrix. The de-facto method to build up the rotation matrix in the computational world is through quaternions. More information can be found in most of the computer graphics texts.

The change of coordinates between systems is carried out with the following general formulas:

$$\begin{aligned} \vec{a}(x,y,z) &= [\vec{A}(X,Y,Z) - \vec{C}(X,Y,Z)] \cdot \bar{\Omega} \\ \vec{A}(X,Y,Z) &= \vec{C}(X,Y,Z) + \left[\bar{\Omega} \cdot \vec{a}(x,y,z)^T\right]^T \end{aligned} \tag{B.1}$$

where $\vec{a}$ is the position vector of a point in the non-inertial system, $\vec{A}$ is the position vector of the same point in the absolute relative system. $\vec{C}$ is the position vector of the location of the origin of the relative system of coordinates, $\bar{\Omega}$ is the rotation matrix. Figure B.1 helps to visualize the idea.

Only the relative transformations are implemented in this study. The following properties of vectors and matrices come handy in the deduction of the equations presented here:

- $(\vec{A}^T)^T = \vec{A}$
- $\vec{A}^T + \vec{B}^T = (\vec{A} + \vec{B}\,)^T$
- $(\vec{A} \cdot \vec{B})^T = \vec{B}^T \cdot \vec{A}^T$
- $\bar{\Omega}^T = \bar{\Omega}^{-1}$, since the rotation matrix is orthogonal
- $\bar{\Omega}^T\bar{\Omega} = \bar{\Omega}\bar{\Omega}^T = I$, thanks to the previous property

A further relative translation

Figure B.2b shows the vector $\vec{b}$ as a further relative translation of the non-inertial system of coordinates. Equations B.1 become:

$$\begin{aligned} \vec{a}(x,y,z) &= [\vec{A}(X,Y,Z) - \vec{C}(X,Y,Z)] \cdot \bar{\Omega} \\ \vec{A}(X,Y,Z) &= \vec{C}(X,Y,Z) + \left[\bar{\Omega} \cdot [\vec{a}(x,y,z) + \vec{b}(x,y,z)]^T\right]^T \end{aligned}$$

A further relative rotation

Figure B.2c shows a absolute rotation around $\vec{E}$ with an angle θ. A new rotation matrix, $\bar{\Omega}_{E,\theta}$, is constructed with $\vec{E}$ and the angle θ. Equations B.1 become:

$$\begin{aligned} \vec{a}(x,y,z) &= [\vec{A}(X,Y,Z) - \vec{C}(X,Y,Z)] \cdot \bar{\Omega}_{E,\theta} \cdot \bar{\Omega} \\ \vec{A}(X,Y,Z) &= \vec{C}(X,Y,Z) + \left[\bar{\Omega}_{E,\theta} \cdot \bar{\Omega} \cdot \vec{a}(x,y,z)^T\right]^T \end{aligned}$$

Appendix C

X3D specification

X3D is the ISO standard file format for representing 3D computer graphics[1]. A general overview of this format is found in Wikipedia[2]. The specification is freely available in Internet at the Web site of the Web3D Consortium[3]. Relevant parts of the document are brought here. Only specialized programs read the X3D format since this format is not common yet. A list of programs is available at the aforementioned websites. The program used in this thesis was Xj3d[4].

C.1 Coordinate system

Coordinates in X3D are specified as an (X,Y,Z) triplet in a right-handed, rectangular coordinate system[5]. Figure C.1 shows the default orientation of the axis in a computer screen. In the default position and orientation, the viewer is on the Z-axis looking down the negative direction of the Z-axis toward the origin with X+ to the right and Y+ straight up[6]

[1] ISO/IEC 19775-1.2:2008

[2] http://en.wikipedia.org/wiki/X3d

[3] www.web3d.org/x3d

[4] www.xj3d.org/

[5] www.web3d.org/x3d/specifications/ISO-IEC-19775-1.2-X3D-AbstractSpecification/Part01/components/rendering.html#Coordinates

[6] www.web3d.org/x3d/specifications/ISO-IEC-19775-1.2-X3D-AbstractSpecification/Part01/components/navigation.html#X3DViewpointNode

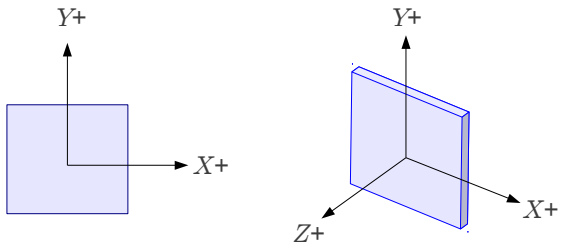
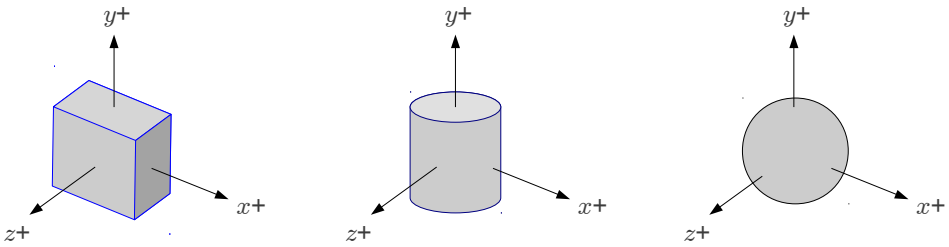


Figure C.1: Orientation of the default coordinate system in a computer screen.

C.2 3D nodes

Figure C.2 shows the geometrical entities, or nodes, used in the program developed in this study.



(a) Box (b) Cylinder (c) Sphere

Figure C.2: Some X3D nodes in 3D.

C.2.1 Box[7]

The Box node specifies a rectangular parallelepiped box. The box centre is located at (0,0,0) in the local coordinate system. The box is aligned with the axes of local coordinate system. By default, the box measures 2 units in each dimension, from -1 to +1. The size field specifies the extents of the box along the X, Y and Z-axes respectively and each component value shall be greater than zero. Figure C.2a illustrates the Box node.

[7] www.web3d.org/x3d/specifications/ISO-IEC-19775-1.2-X3D-AbstractSpecification/Part01/components/geometry3D.html#Box

C.2.2 Cylinder[8]

The Cylinder node specifies a capped cylinder. The cylinder centre is located at (0,0,0) in the local coordinate system and its central axis is oriented along the local Y-axis. By default, the cylinder is sized at -1 to +1 in all three dimensions. It means a height of 2 units and radius 1. Both radius and height shall be greater than zero. Figure C.2b illustrates the Cylinder node.

C.2.3 Sphere[9]

The Sphere node specifies a sphere. The sphere centre is located at (0,0,0) in the local coordinate system. The default radius measures 1 unit. Figure C.2c illustrates the Sphere node.

[8] www.web3d.org/x3d/specifications/ISO-IEC-19775-1.2-X3D-AbstractSpecification/Part01/components/geometry3D.html#Cylinder

[9] www.web3d.org/x3d/specifications/ISO-IEC-19775-1.2-X3D-AbstractSpecification/Part01/components/geometry3D.html#Sphere

Appendix D

How to use the software

The current configuration is ready to simulate the Spex 8000; but, with few changes the program will be able to simulate other devices. It is possible to select a different vial, a different trajectory, or even the actual collision response. The code was written to be flexible. The code is modular and distributed in several files. New files can be written and used instead of the current ones. Unlimited number of outputs can be created and attached to the program. A section is dedicated on how to expand the program.

The program is written in C++ and distributed in its source code. The code has to be compiled before it can be executed. Any C++ compiler will create an executable file. Most C++ programming editors or integrated development environments (IDE) have a built-in compiler. Since the code is distributed in source code, it can virtually run in every operative system; however, the code was tested for Linux only. It is up to the user to choose where and how compile the program. The software developed here was written in NetBeans editor. NetBeans is available for many operative systems and can be obtained at no cost at www.NetBeans.com[1].

The next step after the selection of the compiler and editor is to copy the code to a place with read/write and execution permissions. It requires about 400KB of space for the code itself; but, more space is required in order to save the results. The size of the results depends on how the program is configured to run. The output may range from few kilobytes to few megabytes per simulation.

The program does not have a graphical interface and does not read configuration files. The only way to modify the operating parameters of the program is through text files. Unfortunately, the code has to be compiled every time something is changed before it can be executed. All files in the code are in plain text so any text editor will open

[1] A C/C++ plug-in has to be loaded in order to use C++.

them; however, a C++ editor or IDE is recommended over a text editor because of the availability of useful programming features. A feature that show the properties and methods of objects is very valuable since the code is object oriented.

The program is easy to understand and use. The configuration file works as a collection of commands. The default configuration file is MA01.cpp and is found in the root of the program directory. The file is divided into three logical sections. The first section links external files, the second is a collection of commands, and the last one launch the program. Only the commands are to be changed since the rest is ready to simulate the Spex 8000.

Here is a step by step guide to start a simulation.

D.1 How to introduce values

Some of the commands of the program requires some values to be entered. Those values must conform certain rules in order to be recognized successfully. The instructions here indicate what kind of value it is required. Parameters surrounded by square brackets, i.e. [ ], are optional. Ellipsis (...) indicates that the instruction is incomplete and no format apply yet. The following are the possible formats:

$Text

Texts always has to be surrounded by quotation marks. E.g. “Eps086”.

#UInt

Values referred as #UInt are positive integer numbers like 0, 1, 2 and so on. The limit depends on the compiler, normally it is found under the parameter INT_MAX .

#Dbl

Values referred as #Dbl are real numbers like 1.4142. They can be negative as well. The range and precision of this type of number is found in the compiler documentation.

#Bool

Bool stands for boolean. A boolean value can be either true or false.

$Unit

Some parameters have the option to specify its unit of measurement. The metrical system will be assumed if no unit is specified. All computations within the program are performed using metrical units. Some commands specify the type of unit inside curly brackets ({}) when the type of unit can not be inferred from the name of the command. Units have to be written by its symbol exactly as specified in tables D.1 and D.2 .

Table D.1: Basic units for input.

LENGTH		MASS		TIME		ANGLE		ENERGY	
centimetre	cm	gram	g	hour	h	degree	deg	joule	J
decimeter	dm	kilogram	kg	minute	m	gradient	grad		
foot	ft	pound	lb	millisecond	ms	radian	rad		
inch	in	milligram	mg	second	s				
meter	m								
millimeter	mm								

The following units are derived from the previous units.

Table D.2: Extended units for input.

SPEED	ANGULAR SPEED	ACCELERATION	DENSITY
ft/s	Hz	ft/s2	g/cm3
m/s	rad/s	in/s2	kg/m3
	rpm	m/s2	lb/ft3

Unit symbols have to be surrounded by quotation marks. The following line is an example of the use of units with an arbitrary command:

```
setDensity(5.5, "g/cm3");
```

D.2 Basic instructions

D.2.1 Output folder

Optional. This command tells the program where to store the output files. The folder can be located in the hard disk, a USB memory, or any other place. The only requirement is that user have writing access. The name of the folder must conform OS rules. The application does not create any folder; therefore, a folder must exist before the program start running. Output files are stored by default in the same folder where the program is running. It is advisable to have a folder per simulation to avoid overwriting of the output files. To specify a folder use the following command.

```
Config->setOutputFolder($Text);
```

where $Text is the name of the folder. It can be a sub folder of the current program folder like "Eps084" or a folder in other location like "/usr/home/user1/test/Eps084".

D.2.2 Simulation length

The following options allow to stop a simulation:

- Maximum accumulated milling dose. This option was introduced to find the time that it takes to take the powder to a SHS-ready state (the introduction talks about it). It might be useful for other purposes.

  ```
  Config->setMaximumMillingDose(#Dbl, [$Unit{Energy}]);
  ```

- Maximum time. This option allows to produce time-specific outputs like trends or histograms.

  ```
  Config->setMaximumTime(#Dbl, [$Unit]);
  ```

- Maximum ball speed. This option was introduced to prevent unreal scenarios. Some user parameters might lead balls to acquire speeds higher than normal. This might be the case of a large COR. The maximum speed is checked only for the balls involved in a collision.

  ```
  Config->setMaximumBallSpeed(#Dbl, [$Unit]);
  ```

- Maximum number of collisions. This option was introduced for debugging purposes.

  ```
  Config->setMaximumCollisions(#UInt);
  ```

None of the options were configured by default. Hence, the user must select at least one. If more than one criterion is selected, the simulation stops when at least one condition is met.

D.2.3 Gravity

Optional. Gravity affects the trajectory of balls when they flight between collisions. Balls follow a straight path at constant speed with the effect of no gravity, whereas they have a parabolic trajectory and variable speed under the influence of gravity. The default value is no gravity. To introduce gravity use:

```
Config->setGravity();
```

An acceleration of 9.8 m/s^2 in the negative direction of the Y axis is introduced with this command. The program offers the user the ability of change the magnitude and direction of gravity with the following command:

```
Config->setGravity(#Dbl, #Dbl, #Dbl, [$Unit{acceleration}]);
```

where the first three parameters are the magnitude of the gravity in each direction. $Unit is a unit of acceleration and is optional.

D.2.4 Customizing balls

It is possible to customize every single ball. Balls are added with the current values of diameter, density, velocity, colour, gaps, and number of relocations, as described in the following subsections. Properties must be changed before a ball is introduced to the system. All variables are kept the same until they are explicitly changed. A future change to any ball property do not affect previously added balls. Balls can be introduced individually or in batches with the following command:

```
Config->addSpheres(#UInt);
```

where #UInt is the number of balls to add. It is mandatory for obvious reasons to add at least one ball to the system. The number of balls can be as large as the larger positive integer the compiler allows. This number is large enough to practically run any simulation. The number of balls is also limited by the size of the vial. There is a limit in the number of tries in the location of a ball, see D.2.4.6.

D.2.4.1 Balls dimensions

Optional. Geometrical properties of balls are changed with the same properties of spheres (section D.3.5). The default diameter for balls is 1 cm. The diameter will fall randomly in a range with the following option:

```
Config->setSphereRandomDiameter(#Dbl, [$Unit]);
```

the range will be the current diameter +/- #Dbl [$Unit].

D.2.4.2 Balls density

Optional. The default density for all balls is 8 g/cm^3 which corresponds to the density of steel. Density can be changed with the following command:

```
Config->setSphereDensity(#Dbl, [$Unit]);
```

$Unit is a unit of density and is optional. The mass of every ball is computed by the product of their volume and density.

D.2.4.3 Reduced moment of inertia

Optional. The reduced moment of inertia of balls can be changed with the following command:

```
Config->setSphereReducedMomentOfInertia(#Dbl);
```

The default values is 2/5 which is the value for solid spheres with uniform density.

D.2.4.4 Balls velocity

Optional. All balls start from rest at the beginning of the simulation. The initial velocity of balls can be changed with this command:

```
Config->setSphereVelocity(#Dbl, #Dbl, #Dbl, [$Unit]);
```

where the first three parameters are the magnitude of the velocity in each direction. $Unit is a unit of speed and is optional. Another way to set the initial velocity is by letting the program choose it randomly in magnitude and direction. The following command serves for this purpose:

```
Config->setSphereRandomSpeed(#Dbl, [$Unit]);
```

where #Dbl is the maximum allowable speed.

D.2.4.5 Balls colour.

Optional. The colour will be used by the outputs that produce graphics. The colour of the balls can be changed with the instructions given in section D.3.3. The Colour is changed with the following instruction

```
Config->setColour ...
```

D.2.4.6 Fine tune adjustment.

It is important to start the simulation with no bodies in contact since EDM accept only one collision at a time. A minimum distance should be guaranteed between balls, and between balls and walls.

The minimal initial distance between balls is 1 mm. The following is an optional command that modify this parameter:

```
Config->setInitialS2SphereGap(#Dbl, [$Unit]);
```

The minimal initial distance between walls and balls is also 1 mm. This distance is adjusted with this command:

```
Config->setInitialS2WallGap(#Dbl, [$Unit]);
```

The following command limit the number of possible relocations when a ball is located for the first time:

```
Config->setSphereMaxRelocations(#UInt);
```

D.2.5 Configuring the vial

The vial for the Spex 8000 is a small cylindrical canister. The following command create this container:

```
Collision::IVial01 * Container = new Collision::IVial01(&Interface);
```

This container object has one component inside. This component is a cylinder representing the inner surface of the canister. Section D.3.4 explains how to change properties of a cylinder. This cylinder can be accessed through:

```
Container->cylinder. ...
```

The walls of the vial are identified according to the figure 1.2 with the following ids:

- 0 for the right wall (bottom wall in figure C.2)
- 1 for the left wall (top wall in figure C.2)
- 2 for the cylindrical wall

D.2.6 Kinematics

The kinematics of the Spex 8000 is created with the following command:

```
Collision::IKinematics8000 * Kinematics = new
      Collision::IKinematics8000(&Interface);
```

Section 2.2.2 explains the details about the kinematics of the Spex 8000. l, θ_m, γ_m, and w are specified respectively as follow:

```
Kinematics->setArmLength(#Dbl, [$Unit]);
Kinematics->setMaxTheta(#Dbl, [$Unit]);
Kinematics->setMaxGamma(#Dbl, [$Unit]);
Kinematics->setAngularSpeed(#Dbl, [$Unit]);
```

The default values are the same as those in equation 2.9.

D.2.7 Coefficient of restitution (COR)

The COR is defined in terms of its normal and tangential components. One COR is specified for the ball-ball collisions and another COR for the ball-wall collisions. A constant value can be changed directly in the response module. The current COR is constant during all execution of the program. The following command shows how to do it for the ball-wall response:

```
Wresponse->setNormalCOR(#Dbl);
Wresponse->setTangentialCOR(#Dbl);
```

And, the following command shows how to do it for the ball-ball response:

```
Sresponse->setNormalCOR(#Dbl);
Sresponse->setTangentialCOR(#Dbl);
```

The Incrementally Slipping Friction Model proposed by Walton and Braun (1986) is implemented and included by default. The options available are normal COR, friction coefficient, and minimum tangential COR. The normal COR is constant through the program. Walton's model can be used and personalized with the following commands:

```
Collision::ICOR02 * COR = new Collision::ICOR02(&Interface);
COR->setNormalCOR(#Dbl);
COR->setFrictionCoeff(#Dbl);
COR->setMinimumTangCOR(#Dbl);
```

It is possible to create a new COR. Section D.4.10 explains how to add files.

D.3 Common elements

There are some elements that are common to many of the components of the program.

D.3.1 File name

Most of the outputs of the program go into text files. Every output has a file name by default. The extension of the file is not changeable; but, the name is. The name of the file can be changed with the following command.

```
setFilePrefix($Text);
```

All outputs are written by default in the folder mentioned in the section D.2.1. However, the output folder of the respective output can be changed as follows:

```
setOutputFolder($Text);
```

All outputs create a new file by default. Existing files will be overwritten. Some outputs allow append existing files rather than create new ones. Appending is useful to combine different outputs into a single file. Not all outputs accept appending. Appending is possible with the following command.

```
setAppendFile();
```

D.3.2 Timers

Timers are mostly used to indicate when to start and stop recording of data into the computer persistent memory. Most of the timers in the program are preset to start at time zero. A timer can be turned on or off with the following commands respectively:

```
turnOn();
turnOff();
```

A timer can be turned on at a specific time or iteration (collision) with the following commands respectively:

```
setInitialTime(#Dbl, [$Unit]);
setInitialIteration(#UInt);
```

The first occurring condition starts the output in the case both commands are given. Changing the initial time is useful to let the particles reach its equilibrium state before start recording data. Timers stop by default when the program stop. Outputs can also be stopped after certain amount of time or iterations with the following commands respectively:

```
setDuration(#Dbl, [$Unit]);
setIterations(#UInt);
```

The first occurring condition stops the output if both commands are given.

Timers have also a second counter designed to trigger periodic events. This is a feature used to facilitate trends. This timer indicates when a time span has passed. This timer is on for one collision only then it comes to an off status. The time span is defined by

```
setTimeInterval(#Dbl, [$Unit]);
```

This second timer starts simultaneously with the main timer. The second counter can be stopped when it reaches certain number of intervals. The following command serves for this purpose:

```
setMaximunRows(#UInt);
```

D.3.3 Colours

The colour of an entity can be changed in three ways. The first one is by using one of the following colours: black, blue, cyan, fuchsia, grey, green, lime, magenta, maroon, navy, olive, purple, red, teal, white, yellow. The colour name is to be introduced through the following command:

```
setColour($Text);
```

Another option is by using the combination of red, green, and blue, in this order. The amount of each colour ranges from 0 to 255.

```
setColour(#Dbl, #Dbl, #Dbl);
```

The last option also use the combination of red, green, and blue; but, as integers from 0 to 1:

```
setColour(#UInt, #UInt, #UInt);
```

D.3.4 Cylinders

Cylinders are used as geometrical representation of cylindrical vials. A cylinder is a geometrical entity with volume but no associated mass. Diameter and height of a cylinder can be changed respectively with the following commands:

```
setDiameter(#Dbl, [$Unit]);
setHeight(#Dbl, [$Unit]);
```

The colour can also be changed as specified in section D.3.3.

D.3.5 Spheres

Spheres are used to represent balls and contact points. As cylinders, spheres are geometrical entities with volume but no associated mass. The location of a sphere is referenced to its geometrical centre. Sphere radius or diameter can be changed with the following commands:

```
setRadius(#, [$Unit]);
setDiameter(#, [$Unit]);
```

The colour can also be changed as specified in section D.3.3.

D.4 Outputs

The program has several built-in outputs. All libraries of the available outputs have been already included in the main cpp program (look for some #include lines). The program do not produce any output by default. It is up to the user to select which outputs to use. Some outputs were created for debugging purposes only. To use an output it has to be created first, then customized, and finally added to the system. An output is created with its default parameters.

The following lines show an example of how to export statistics.

```
Collision::IOutputStatistics * Statistics = new
      Collision::IOutputStatistics(&Interface);
Config->addOutput(Statistics);
```

where *Statistics* is the name of the output object. It is just a name for reference and not the actual output file. Collision::IOutputStats is the type of object. The word *&Interface*

is the link to the program interface. It allows outputs to show messages in the screen or in a debug file.

All outputs can be programmed to start and stop recording data at any moment. The object Timer defined in section D.3.2 is implemented in each output. This timer is found under the name *timer*. By default, all outputs start automatically at time zero. Some of the outputs only last few collisions by default.

There is not limit in the number of outputs that can be used, and even more, it is possible to have more than one output of the same type. A file name should be given to every output, otherwise, they will be writing the same file. For example, it is possible to have one statistics for the first minute and another one for the rest of the running time. This can be illustrated in following lines:

```
e.g.>  Collision::IOutputStatistics * Statistics01 = new
           Collision::IOutputStatistics(&Interface);
       Statistics01->filename.setFilePrefix(“Statistics01”);
       Statistics01->timer.setDuration(1, “min”);
       Config->addOutput(Statistics01);
       Collision::IOutputStatistics * Statistics02 = new
           Collision::IOutputStatistics(&Interface);
       Statistics02->filename.setFilePrefix(“Statistics02”);
       Statistics02->timer.setInitialTime(1, “min”);
       Config->addOutput(Statistics02);
```

Many of the outputs export data to text files with CSV extension. CSV stands for Comma Separated Values. CSV files are like a spreadsheet table stored in a plain text file. Values in columns are separated by commas. Every row in the text file is a data record. CSV files can be opened in any text editor; but, they are best showed in spreadsheets programs like Excel or LibreOffice Calc. CSV can also be imported into databases, like MS Access or MySQL. Outputs have default names for the files they write. The name of the file can be changed with a component called *filename* and the instructions given in section D.3.1.

The graphical outputs export data in the format X3D. A short description of this format is found in appendix C. The program developed here produces X3D files but does not render their content in the screen; another program is needed to do so. The page www.web3d.org/products provide a list of various commercial, free, and open source viewers available out there. The graphical outputs were tested only with xj3d version 1.0 for Linux.

Many outputs require to specify a program parameter. Many parameters before and after every collision has been collected. Those parameters are divided in three groups according to the nature of the variable in C++. There are bools, unsigned integers, and doubles. Bools store true/false information, double stores real numbers, and unsigned integers are positive integers. The following is the convention used to specify the kind of parameter needed:

Id@B

It is a parameter of type bool. Table D.3 lists of available parameters of this type. Following line is an example of how to specify a ball-ball collision:

```
Collision::bBallBall
```

Id@D

It is a parameter of type double. Table D.5 shows all parameters of this type. Following line is an example of how to make reference to the milling dose per collision:

```
Collision::dMillingDose
```

Id@UI

It is a parameter For example to make reference to the of type unsigned integer. Table D.4 shows all parameters of this type. Following line is an example of how to make reference to the collision number:

```
Collision::iIteration
```

#Label

The name of the variables in the output files is not printed out automatically. This option allows to insert any custom text in order to have more control of the results.

#Stat

The flags *#Stat* serve to indicate the type of statistical operation for those outputs that gather data in time. The following list gives the identification for average, maximum, minimum, percentage, rate, and sum respectively:

```
Collision::sAve
Collision::sMax
Collision::sMin
Collision::sPercentage
Collision::sRate
Collision::sSum
```

Another feature of the program allows to have more control of when an output should be active. Some outputs can be limited to some variables. For instance, sometimes is wanted to collect data from ball-ball collisions while ignoring ball-wall collisions. A component called restrictions is added to some outputs for this purpose. The outputs can be restricted to bool and/or integer values with the following commands respectively:

```
addRestrictionBool(Id@Bool, #Bool);
addRestrictionUInt(Id@UI, #UInt);
```

The following lines show an example of how to constrain data collection to the second wall of a vial:

```
e.g.>   addRestrictionBool(bBallBall, false);
        addRestrictionUInt(iSecond, 1);
```

D.4.1 Snapshots

This output is created with the following command

```
Collision::IOutputSX3D * X3DSnapshots = new
      Collision::IOutputSX3D(&Interface);
```

The contact point of a collision can be shown as a sphere. The contact point is a component under the name of *contactPoint*. The contact point is a sphere, and as such, its properties can be changed as described in section D.3.5. The following property makes contact point visible:

```
showContactPoint([#Bool]);
```

One example of a snapshot is presented in section 4.3.1.

D.4.2 Animations

This output is created with the following command

```
Collision::IOutputX3D * X3DAnimation = new
      Collision::IOutputX3D(&Interface);
```

Animations created with this output are limited to kinematics that have periodic motions or no motion at all. The trajectory of the vial is displayed as an interpolation of a series of points. Those points are computed from the period time of the kinematics and not from the motion of the vial itself. Constant speed motions of vial or walls like pistons can be animated when the length of the animation is equal or shorter than the length of its trajectory. The higher amount of points the smoother of the movements. However, beyond a certain number of points the eye cannot notice the difference. A minimum of 24 coordinate changes per minute is recommended. The default value is 60. This parameter is optional and can be changed with the following property

```
setAnimationFPM(#UInt);
```

The number of points is sometimes so low that the motion of the vial lacks of smoothness. A minimum number of points is set to 8 by default to prevent this from happening. The following command allows to change this number:

```
setAnimationMinFrames(#UInt);
```

The playback speed of the animation can be changed with the following command:

```
setAnimationSpeed(#Dbl);
```

A number higher than one increases the playback speed whereas a number lower than one decreases it. A value equal to 1 means no change of speed and is the default value. The default value is set to 0.001. This value is slow enough to appreciate the Spex 8000 with detail.

One example of an animation is presented in section 4.3.2.

D.4.3 Statistics

This output is created with the following command:

```
Collision::IOutputStatistics * Statistics = new
      Collision::IOutputStatistics(&Interface);
```

At least one collision variable must be specified to use this output. Booleans and double variables can be used. To produce statistics of a boolean and double variables use the following commands respectively:

```
addBool(Id@B, [$Label], [#Stat], [#Stat], [#Stat]);
addDouble(Id@D, [$Label], [#Stat], [#Stat], [#Stat], [#Stat],
      [#Stat], [#Stat]);
```

Bool variables only accept percentage, rate, and sum; while, double variables accept all of them.

One example of *statistics* is presented in section 4.3.5.

D.4.4 DFrecuency

This output is created with the following command:

```
Collision::IOutputDFrecuency * NVFrecuencies = new
      Collision::IOutputDFrecuency(&Interface);
```

The variable to analyze is selected with the following command:

```
setDouble(Id@D, [$Label]);
```

The total range and the number of intervals is defined with the following commands:

```
setInitialValue(#Dbl);
setFinalValue(#Dbl);
setNumberOfIntervals(#UInt);
```

The result of this output is a composite table presented with minimum information. More sections or information can be added with extra command. A tittle for the table can be introduced with the following command:

showTitle($Text);

A header can be introduced with the following command:

showHeader($Header);

where $Header is the header identification. There are four possible headers: number of each interval, lower bound of each interval, upper bound, and mean of the interval. The corresponding labels are:

```
Collision::OutputDFrecuency::headerIntervalNumber
Collision::OutputDFrecuency::headerLowerBounds
Collision::OutputDFrecuency::headerUpperBounds
Collision::OutputDFrecuency::headerMeans
```

More columns of the output table can be introduced with the following command:

```
showColumns($Column);
```

where $Column is the column identification. Three columns are available. Two columns show the count of times a value goes off the given interval. Another columns show the total per row. Following lines are corresponding labels.

```
Collision::OutputDFrecuency::columnLowerBound
Collision::OutputDFrecuency::columnUpperBound
Collision::OutputDFrecuency::columnTotalsPerRow
```

More information can be added to the output table. The following command allows to introduce more information to the results:

```
showSection($Section);
```

where $Section is the label of the section. The result can be expanded to include frequencies per wall, and also can include the percentage of each frequency. The following lines are the corresponding labels

```
Collision::OutputDFrecuency::sectionWallDetails
Collision::OutputDFrecuency::sectionPercentages
```

One example of this output is presented in section 4.3.4.

D.4.5 Trends

A report of trends of different variables are produced with the following command:

```
Collision::IOutputTrends * Trends = new
      Collision::IOutputTrends(&Interface);
```

The time is always shown in the output by default. The time of the interval is specified with the command *setTimeInterval* of the timer as described in section D.3.2. The default time interval is 30 seconds. The output is limited to 32000 records. To change this last value use the command *setMaximunRows* of the timer.

Trends of a boolean variable are exported the following command:

```
addBool(Id@B, #Stat, [$Label]);
```

Trends for booleans only accept sum and rate as the statistical mode.

Trends of double variable are exported the following command:

```
addDouble(Id@D, #Stat, [$Label]);
```

Trends for doubles% accept all statistics except percentage.

One example of this output is presented in section 4.3.6.

D.4.6 Trends Inst

This kind of trend is similar to *Trends* in everything except that only the instantaneous values of the variables are printed. This output is created with the following command:

```
Collision::IOutputTrendsInst * TrendsInst = new
      Collision::IOutputTrendsInst(&Interface);
```

To show a boolean variable use the following command:

```
addBool(Id@B,[$Label], [$Text], [$Text]);
```

The $Text fields are the text to print if the variable is true, and the text to print if the variable is false.

To show double and integer variables use the following commands respectively:

```
addDouble(Id@D, [$Label]);
addInteger(Id@UI, [$Label]);
```

One example of this output is presented in section 4.3.7.

D.4.7 Trends Ball

This output is created with the following command:

```
Collision::IOutputTrendsBalls * TrendsBalls = new
      Collision::IOutputTrendsBalls(&Interface);
```

This output present no information by default. The following command instructs the program to include specific information.

```
addField($BallInfo, #Stat, [#Bool], [$Label]);
```

All $Stat identifiers work except rate. #Bool serves to know which ball have a maximum or minimum value of the variable; but, this only works when $Stat is *maximum or minimum*. $BallInfo is one of the following options

dDensity	(Density)
dMass	(Mass)
dReducedMOI	(Reduced moment of inertia)
dMOI	(Moment of inertia)
dTransSpeed	(Translational speed)
dRotSpeed	(Rotational speed)
dTransEnergy	(Translational energy)
dRotEnergy	(Rotational energy)
dTotalEnergy	(Total energy)

One example of this output is presented in section 4.3.8.

D.4.8 Trends Trace

This output prints out the position of a ball in a CSV file. This output was introduced for debbuging purposes. The following command creates this output:

```
Collision::IOutputTrace * Trace = new
       Collision::IOutputTrace(&Interface);
```

The only option is the Id of the ball. The *Id* starts at zero. It means the first ball have an *Id* of 0.

```
setBallId(#UInt);
```

D.4.9 Iterations

This output was created for debugging and helps to verify collision responses. It gives the values of almost all variables before and after collision; however, it does not check the accuracy of the response. CSV is the format used. This output is created with the following command:

```
Collision::IOutputIter * Iteration = new
       Collision::IOutputIter(&Interface);
```

There is not options for this output. But, this output have a timer and can be restricted.

D.4.10 Expanding the possibilities

The code can be extended beyond the Spex 8000. The program was designed to be flexible and expandable. Other shapes different than cylinders can used, or even containers with moving walls like pistons are accepted. Any trajectory of the vial is possible. Other parts of the program can also be changed.

The program is written in C++. The code was written using classes. A programmer will find that the hierarchy of the existing classes is very logic. There is one file per class and all files reside in the same directory. Instructions on the creation of specific files is out of scope in this work; however, general guidelines are given. It is highly recommended that the user use resident files as example. And is also recommended to use an IDE in order to increase productivity.

To create an object, like a vial, two classes have to be created. One for the user interface and another one for the object itself. Splitting the code of an object reduce its complexity while improving its readability and reliability. The interface asks the user for parameters and check them. The object itself is created once program finish checking the user input.

To use an interface class it has to be mentioned in the main file. For example, to refer a new vial in the file mainfile.cpp a command like this should be used.

#include "INewVial.h"

Notice that the letter I is preceding some files. This is the indication that those file are user interfaces. There is no need to refer the associated (the object itself) class in the main program; but, it has to be referred inside the interface class.

The coefficient of restitution (COR) can also be coded. It can be programmed to change in time or to be dependent on the conditions of the collision. The file COR02.h was created to serve as example. The first step is to create the object COR, e.g:

```
Collision::ICOR02 WCOR(&Interface);
```

and then introduce it to one or both of the responses. E.g.:

```
Wresponse.setCOR(&WCOR);
```

Collisions responses can be modified as well. There is one response for ball-ball collisions and another response for ball-vial collisions.

Prediction of the time of collision between the balls and the vial is performed in a stet-wise fashion. This gives freedom in the shape of the vial and its trajectory. The algorithm for prediction is a class itself that can be replaced by any other created by the user. Simply modify the following two lines in the code.

```
#include "Iwpredictor01.h"
Collision::IWPredictor01 Wpredictor(&Interface);
```

The set of units lies under the file *Conversions.h*. This file can be easily modified to adopt more units.

D.5 System variables

It is possible to retrieve information of the current collision. Many parameters before and after every collision has been collected.

The parameters are divided into groups according to the nature of the variable in C++. There are bools, unsigned integers, and doubles. Bools store true/false information, double stores real numbers, and unsigned integer are positive integers.

The following parameters of type bool are stored:

Table D.3: List of bool parameters.

NAME	DESCRIPTION
bBallBall	It is true if the collision happened between balls. It is false if it was a ball-vial collision
bCol	Always true. It is only used for the counting of collisions.
bGravity	Indicates if a gravity force is used.

The following unsigned integers are stored:

Table D.4: List of unsigned integer parameters.

NAME	DESCRIPTION
iBallsAmount	Amount of balls
iFirst	Id of one of the colliding walls. The counting start at zero
iIteration	Number of collision.
iSecond	Id of the other colliding ball, or the id of the wall.
iWallsAmount	Amount of walls of the vial. E.g. for a prism is 6, or for a cylinder is 3.

The following double variables are stored:

Table D.5: List of double parameters.

NAME	DESCRIPTION
dAngleNormal1After	Angle between the normal and the velocity of the first ball after collision
dAngleNormal1Before	Angle between the normal and the velocity of the first ball before collision
dAngleNormal1Change	Difference between the two previous variables
dAngleNormal2After	Angle between the normal and the velocity of the first ball after collision
dAngleNormal2Before	Angle between the normal and the velocity of the first ball before collision
dAngleNormal2Change	Difference between the two previous variables
dAngleNormalCpBefore	Angle between the normal and the relative velocity at the contact point before collision
dAngleNormalRelAfter	Angle between the normal and the relative translational velocity after collision
dAngleNormalRelBefore	Angle between the normal and the relative translational velocity before collision
dAngleNormalRelChange	Difference between the two previous variables
dEnergyBalance	Balance of total energy per collision. dEnergyTransferWall - dMillingDose
dEnergyRot1After	Rotational energy of the first colliding ball after collision
dEnergyRot1Before	Rotational energy of the first colliding ball before collision
dEnergyRot1Change	Difference between the two previous variables
dEnergyRot2After	Rotational energy of the second colliding ball after collision. 0 for B2V Collisions
dEnergyRot2Before	Rotational energy of the second colliding ball before collision. 0 for B2V Collisions
dEnergyRot2Change	Difference between the two previous variables
dEnergyRotTotalAfter	Total rotational energy after collision. dEnergyRot1After + dEnergyRot2After
dEnergyRotTotalBefore	Total rotational energy before collision. dEnergyRot1Before + dEnergyRot2Before
dEnergyRotTotalChange	Difference between the two previous variables
dEnergyTotal1After	Total energy of the first colliding ball after collision. dEnergyRot1After + dEnergyTras1After
dEnergyTotal1Before	Total energy of the first colliding ball before collision. dEnergyRot1Before + dEnergyTras1Before
dEnergyTotal1Change	Difference between the two previous variables
dEnergyTotal2After	Total energy of the second colliding ball after collision. dEnergyRot2After + dEnergyTras2After

NAME	DESCRIPTION
dEnergyTotal2Before	Total energy of the second colliding ball before collision. dEnergyRot12Before + dEnergyTras2Before
dEnergyTotal2Change	Difference between the two previous variables
dEnergyTotalAfter	Total energy of the colliding balls after collision. dEnergyTotal1After + dEnergyTotal2After
dEnergyTotalBefore	Total energy of the colliding balls after collision. dEnergyTotal1Before + dEnergyTotal2Before
dEnergyTotalChange	Difference between the two previous variables
dEnergyTransferWall	Energy transferred from the wall to the colliding ball. 0 for B2B Collisions
dEnergyTras1After	Translational energy of the first colliding ball after collision
dEnergyTras1Before	Translational energy of the first colliding ball before collision
dEnergyTras1Change	Difference between the two previous variables
dEnergyTras2After	Translational energy of the second colliding ball after collision. 0 for B2V Collisions
dEnergyTras2Before	Translational energy of the second colliding ball before collision. 0 for B2V Collisions
dEnergyTras2Change	Difference between the two previous variables
dEnergyTrasTotalAfter	Total translational energy after collision. dEnergyTras1After + dEnergyTras2After
dEnergyTrasTotalBefore	Total translational energy before collision. dEnergyTras1Before + dEnergyTras2Before
dEnergyTrasTotalChange	Difference between the two previous variables
dFreePath1	Trajectory between collisions of the first colliding ball.
dFreePath2	Trajectory between collisions of the second colliding ball. 0 for B2V Collisions
dFreePathAve	Average of the free path of the colliding balls. (dFreePath1 + dFreePath2) / 2
dMillingDose	Milling dose of the current collision.
dRot1AfterMag	Magnitude of the rotational velocity of the first colliding ball after collision.
dRot1BeforeMag	Magnitude of the rotational velocity of the first colliding ball before collision.
dRot1ChangeMag	Difference between the two previous variables
dRot2AfterMag	Magnitude of the rotational velocity of the second colliding ball after collision. 0 for B2V Collisions
dRot2BeforeMag	Magnitude of the rotational velocity of the second colliding ball before collision. 0 for B2V Collisions

NAME	DESCRIPTION
dRot2ChangeMag	Difference between the two previous variables
dTimeCurrent	Current time
dTimeDelta	Elapsed time from the last collision
dTimePrevious	Time of the previous collision
dVel1AfterMag	Magnitude of the translational velocity of the first colliding ball after collision.
dVel1BeforeMag	Magnitude of the translational velocity of the first colliding ball before collision.
dVel1ChangeMag	Difference between the two previous variables
dVel2AfterMag	Magnitude of the translational velocity of the first second ball after collision.
dVel2BeforeMag	Magnitude of the translational velocity of the first second ball before collision.
dVel2ChangeMag	Difference between the two previous variables
dVelCpBeforeMag	Magnitude of the relative velocity at the contact point before collision
dVelNormal1AfterMag	Magnitude in the normal direction of the translational velocity of the first colliding ball after collision.
dVelNormal1BeforeMag	Magnitude in the normal direction of the translational velocity of the first colliding ball before collision.
dVelNormal1ChangeMag	Difference between the two previous variables
dVelNormal2AfterMag	Magnitude in the normal direction of the translational velocity of the second colliding ball after collision.
dVelNormal2BeforeMag	Magnitude in the normal direction of the translational velocity of the second colliding ball before collision.
dVelNormal2ChangeMag	Difference between the two previous variables
dVelNormalCpBeforeMag	Magnitude in the normal direction of the relative velocity at the contact point before collision
dVelNormalRelAfterMag	Magnitude in the normal direction of the relative translational velocity after collision.
dVelNormalRelBeforeMag	Magnitude in the normal direction of the relative translational velocity before collision.
dVelNormalRelChangeMag	Difference between the two previous variables
dVelRelAfterMag	Magnitude of the relative translational velocity after collision.
dVelRelBeforeMag	Magnitude of the relative translational velocity before collision.

NAME	DESCRIPTION
dVelRelChangeMag	Difference between the two previous variables
dVelTangential1AfterMag	Magnitude in the tangential direction of the translational velocity of the first colliding ball after collision.
dVelTangential1BeforeMag	Magnitude in the tangential direction of the translational velocity of the first colliding ball before collision.
dVelTangential1ChangeMag	Difference between the two previous variables
dVelTangential2AfterMag	Magnitude in the tangential direction of the translational velocity of the second colliding ball after collision.
dVelTangential2BeforeMag	Magnitude in the tangential direction of the translational velocity of the second colliding ball before collision.
dVelTangential2ChangeMag	Difference between the two previous variables
dVelTangentialCpBeforeMag	Magnitude in the tangential direction of the relative velocity at the contact point before collision
dVelTangentialRelAfterMag	Magnitude in the tangential direction of the relative translational velocity after collision.
dVelTangentialRelBeforeMag	Magnitude in the tangential direction of the relative translational velocity before collision.
dVelTangentialRelChangeMag	Difference between the two previous variables