DEGREE PROJECT IN MATHEMATICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2019

Suturing in Surgical Simulations

KIRAN BEERSING-VASQUEZ

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

Suturing in Surgical Simulations

KIRAN BEERSING-VASQUEZ

Degree Projects in Scientific Computing (30 ECTS credits)
Degree Programme in Applied and Computational Mathematics (120 credits)
KTH Royal Institute of Technology year 2019
Supervisor at SenseGraphics AB: Julien Lenoir
Supervisor at KTH: Michael Hanke
Examiner at KTH: Michael Hanke

TRITA-SCI-GRU 2019:359
MAT-E 2019:85

Royal Institute of Technology
School of Engineering Sciences
KTH SCI
SE-100 44 Stockholm, Sweden
URL: www.kth.se/sci

Abstract

The goal of this project is to develop virtual surgical simulation software in order to simulate the suturing and knot tying processes associated with surgical thread. State equations are formulated using Lagrangian mechanics, which is useful for the conservation of energy. Solver methods are developed with theory based in Differential Algebraic Equations (DAEs) which concern governing Ordinary Differential Equations (ODEs) that are constraint with Algebraic Equations (AE). An implicit integration scheme and Newton's method is used to solve the system in each step. Furthermore, a collision response process based on the Linear Complementarity Problem (LCP) is implemented to handle collisions and measure their forces.

Models have been developed to represent the different types of objects. A spline model is used to represent the suture and mass-spring model for the tissue. They were both selected for their efficiency and base on real physical properties. The spline model was also chosen as it is continuous and can be evaluated at any point along the length. Other objects are also defined such as rigid bodies.

The Lagrangian multiplier method is used to define the constraints in the model. This allows for the construction of complex models. An important constraint is the suturing constraint, which is created when a sufficient force is applied by the suture tip on to the tissue. This constraint allows only a sliding point along the suture to pass through a specific point on the tissue.

This results in a virtual suturing model which can be built on for use in surgical simulations. Further investigations would be interesting to increase performance, accuracy and scope of the simulator.

Sammanfattning

Härdning i kirurgiska simuleringar

Det här projektet syftar till att utveckla mjukvara för virtuell simulering av kirurgi som involverar knytande av suturtråd. Lagranges ekvationer används för att härleda energibevarande tillståndsekvationer. Lösningsmetoderna grundar sig i teori från området Differential-Algebraiska Ekvationer (DAEer), som avser att kontrollera Ordinära Differentialekvationer (ODEer) med algebraiska bivillkor. Ett implicit integrationsschema och Newtons metod används för att lösa systemet i varje steg. Utöver det så implementeras en kollisionsresponsprocess baserad på det linjära komplementaritetsproblemet (LCP) för att hantera kollisioner och mäta deras krafter.

Modeller har utvecklats för att representera olika typer av objekt. En spline-modell används för att representera suturtråden och ett mass-fjäder system för vävnaden. Valet baserades på deras höga prestanda samt starka anknytning till objektens fysiska egenskaper. Spline-modellen valdes också då dess kontinuitet innebär att den går att evaluera för en godtycklig punkt inom dess domän. Andra objekt, såsom stela kroppar, finns också definierade.

Lagrangemultiplikator används för att definiera bivillkor i modellen. Detta tillåter konstruktionen av komplexa modeller. Ett viktigt bivillkor är suturbivillkoret som uppstår när tillräcklig kraft från spetsen på den kirurgiska nålen appliceras på vävnaden. Detta bivillkor tillåter att endast en glidande punkt längsmed suturen passerar genom en specifik punkt på vävnaden.

Detta resulterar i en virtuell modell för stygn som kan byggas vidare på för användning i kirurgiska simulationer. Det vore intressant med ytterligare undersökningar för att förbättra prestandan, precisionen och simulatorns omfattning.

Acknowledgements

There are many people whom I would like to give my thanks to here. Firstly I would like to thank Daniel Evestedt and Tommy Forsell for hiring me at SenseGraphics in the first place, along with Julien Lenoir, Wyatt Bardouille and Jan Östman who were also part of the hiring process.

I would especially like to thank Julien for his role as my tutor where he was able to patiently explain complex topics and provide technical guidance for the duration of this project. Those who worked on `SGPhysics` are also important contributors to the success of this project, including Jesper Andersson and Oscar Mårtensson. *Tack så mycket* to Michael Stahre for helping with Swedish translations. And as it's not all work and no play, I would also like to thank my valiant squash and table-tennis, "*pingis*", opponents: Markus Israelsson, Thibaut Buchert, Einar Dahl, Asmaa Ait Hadouch, Hao Zhong and Michael Carlie.

My deepest gratitudes go to the coordinators of the COSSE Joint Master programme Michael Hanke, Reinhard Nabben and Kees Vuik for giving me the opportunity to take part in this journey. In particular I would like to thank Michael Hanke for supervising this project, and guiding me throughout. Thank you to the professors and teaching assistants for providing the structure to develop my skills and knowledge in computational simulations.

I'd like to give a huge shout out to all of the new people I've met and friends made over the last two years. It has been very special. Congratulations to all those graduating this year.

And, of course, I would like to thank my parents, Sunil and Debora, and my brother, Adrian, for their encouragement and support over all of these years. Thanks guys!

Contents

Chapter 1

Introduction

1.1 About the Company

SenseGraphics AB (logo at Figure 1.1) is a Swedish company based in Kista, near Stockholm, Sweden. It was founded in 2004 and has around 20 employees. SenseGraphics offers cutting edge medical simulator software that is used in the field of surgical simulation. This has a wide range of applications, with the technology being used in areas such as robotic surgery, eye surgery, ultrasound interpretation, dentistry, minimally invasive interventions and anaesthetics. The simulators are used to train surgeons in a realistic environment without ever putting a patient at risk.

Figure 1.1: SenseGraphics AB logo. [Courtesy of SenseGraphics AB]

More specifically, SenseGraphics develops software for the modelling of the physics, the 3D rendering and the haptic feedback of these applications. One example is shown in Figure 1.2. The models are very realistic and versatile, and are simulated in real-time.

1.2 Problem

An important part of most surgeries is the use of surgical thread to stitch tissues together. In the medical field, this is called suturing. In order for surgeons to

practice suturing in robotic surgery, a suturing model must be implemented in SenseGraphics' virtual simulator. This model should provide a realistic, real-time simulation of suturing that can be reliably used as a training tool. In order to achieve this, numerical modelling techniques must be used.

There are a wide array of suture types available, being made of different materials and with different geometric properties such as diameter and resting configuration. Furthermore, different stitching techniques are used, that can either be continuous or interrupted, and of varying depths. One example of a stitching technique is shown in Figure 1.3. Different types of needles are also used. These can be straight or curved to different degrees, and of different diameters. Therefore the suturing model must be very flexible in order to account for these possibilities.

Another important part of the suturing process is the tying of knots in the suture. The knot is used to bind the suture to itself, while also binding the tissue. Once tightened, it should remain tight [2]. An example of a common knot used in surgery is shown in Figure 1.4. The simulation of knots is a highly complex problem, with many complex interactions and numerous constraints. The suturing model must therefore be robust enough to handle them in a real-time simulator.

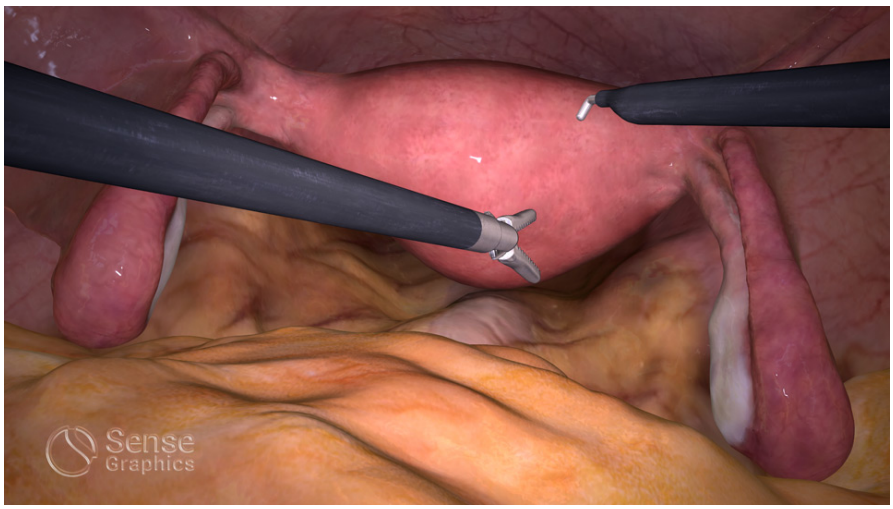
1.3 Aim and Objectives

The aim of this project is as follows:

> The design, development and implementation of a complete suturing simulation model.

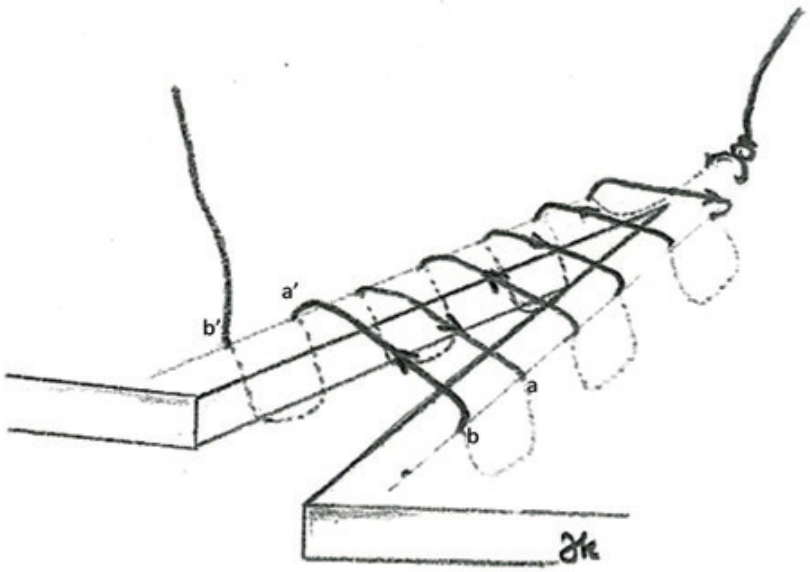
In order to achieve this aim, the following objectives were set:

- Develop a method to solve problems of constrained motion in physical space using theory from Differential Algebraic Equations (DAEs).
- Design and implement a spline model for the suture.
- Design and implement a model for the tissue.
- Implement a constraint-handling method for the models.
- Design and implement a model for suture knots (ligatures).
- Build example models and evaluate the implementation.



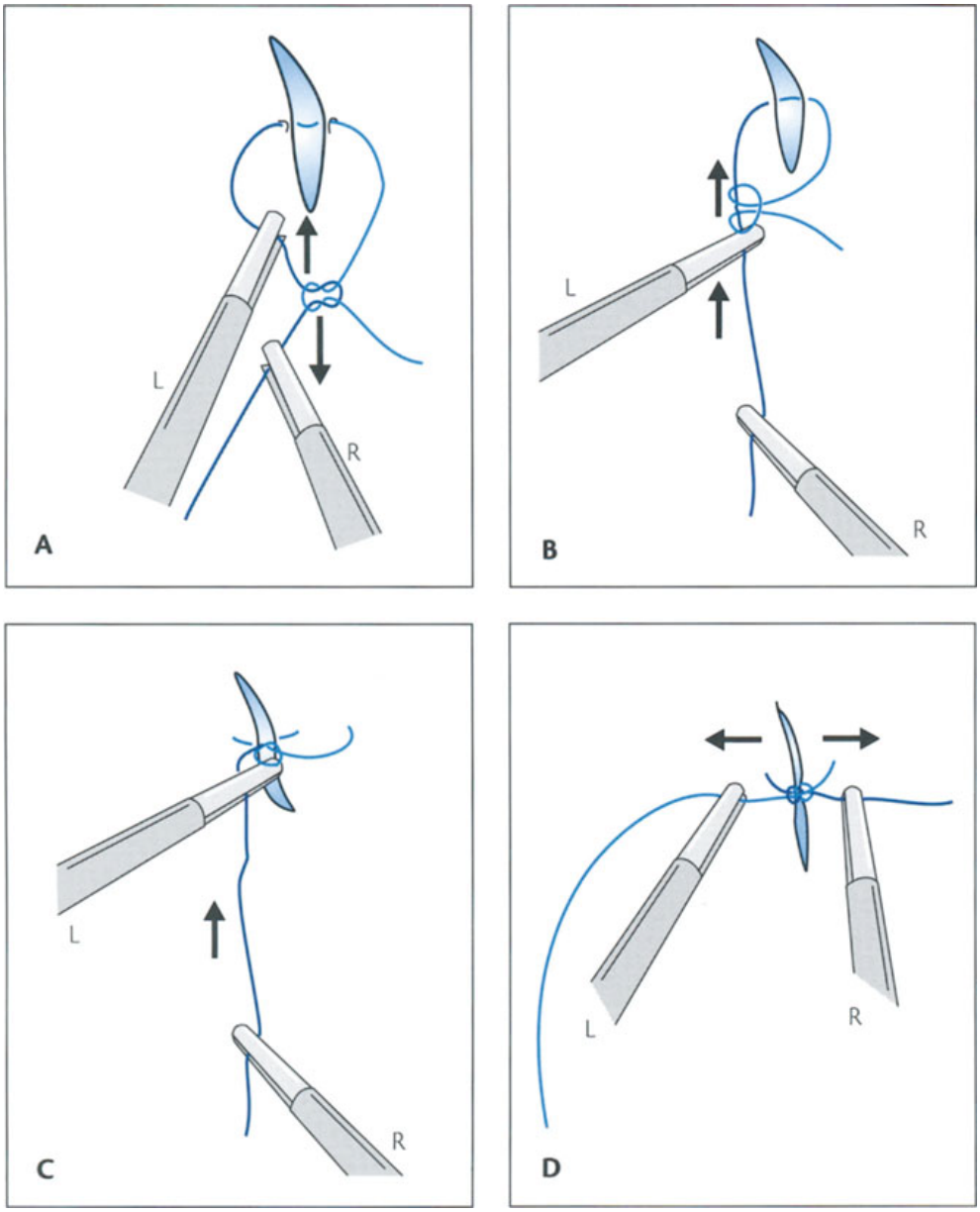
Source: SenseGraphics https://sensegraphics.com/

Figure 1.2: Real-time virtual simulation of a laparoscopic surgery. [Courtesy of SenseGraphics AB]



Source: Clinical and Experimental Otorhinolaryngology https://www.e-ceo.org/

Figure 1.3: Diagram of a Connell Suture [1]. [Reproduced in accordance with the Budapest Open Access Initiative definition of open access for scholarly, educational purposes]



Source: Springer https://link.springer.com/

Figure 1.4: Diagram of a slip knot conversion from Garber and Sackier [3]. The locking square knot is first changed to a slip knot (**A**). The knot is then slid down (**B**, **C**). The slip knot is then reconverted to a locking square knot. (L-left hand; R-right hand). [Reproduced with permission from Springer Nature]

This report details the process of following these objectives, with the results and the evaluation of the results.

In the next section, Chapter 2: Background, previous work in the field is investigated and assessed. Motivations for their inclusion or exclusion in this project are given. Following that, Chapter 3: Method outlines the procedure and content of the approach used to solve the problem. The underlying mathematical procedure and engineering-related contents are described. Next, Chapter 4: Implementation gives a practical description of how the method was applied prorgramatically. Chapter 5: Results describes the qualitative and quantitative results of the degree project. In Chapter 6: Discussion, the positive effects and the drawbacks are investigated, and the results of the degree project are evaluated. Finally Chapter 7: Conclusion summarises the work and achievements, and offers direction for future work.

Chapter 2

Background

2.1 Medical Application

Robot–assisted surgery is being used increasingly in the medical world leading to fewer complications and faster recovery times [4] [5]. A study from Elhage et al. [6] compared robot–assisted surgery (RAS) with the more traditional approaches of open–surgery (OS) and laparoscopic surgery (LAP) performed without robot assistance. Six surgeons were tasked with performing an in vitro simulated vesico–urethral anastomosis using each technique, and were assessed along several metrics. The results showed that RAS combines the best aspects of OS and LAP. Like OS, it has a short procedure time, produces a low number of errors and a low level of discomfort was reported. LAP on the other hand has a prolonged task time, and was the method which produced the highest number of errors and highest level of reported discomfort. Like LAP however, RAS has the advantages of minimally invasive surgery, resulting in lower complication rates and lower blood loss [4] [7]. Because of the relative increased cost of the equipment for these procedures, they are often used for more specialised surgeries where the patient is at greater risk of complications, but they are commonplace in hospitals around the globe and more cost–effective robotic platforms are on the horizon [5].

In order for surgeons to practice using these machines they can either use the robot on real physical test pieces, or they can practice in a virtual environment that simulates a real surgical procedure. One of the most famous robots used in robotic surgery is the da Vinci (Si and Xi) from Intuitive Surgical [1]. SenseGraphics technology was used to build the da Vinci Skills Simulator (see

[1] Intuitive | Robotic Assisted Systems | da Vinci Robot `https://www.intuitive.com/en-us/productsâĂŞand-services/da-vinci/systems`

Figure 1.2), which comes bundled in this robot. This simulator is an example of a virtual reality tool used to train surgeons in a safe environment directly on the console itself.

Studies show that improvements from practising in a virtual test environment are similar to those of using a physical, dry–lab, test environment for simple skills training [8]. Advantages of the simulated training environment however include its flexibility, no requirement for disposable one–time–use parts and almost limitless possibilities for different scenarios. The use of a physics engine, such as PhysX, can provide certain advantages such as allowing for a method of force feedback for simulation on haptic–enabled robots [9] [10]. On the other hand, building the physics engine from scratch enables greater flexibility in the implementation, greater optimisation for real–time calculations, and more scope to use different types of mathematical models to build the system as opposed to what "comes in the box". A virtual training environment can also provide performance metrics to the trainee. It has been shown that by using these metrics trainees can improve substantially, even if there is no mentor guiding the process [11], so the potential for self–improvement and self–skills–learning is huge.

2.2 Related Work

2.2.1 Suture Models

The modelling of medical sutures for virtual simulations has been attempted using many different approaches. All of these implementations fall under the broader topic of the simulation of 1D Deformable Objects (ODDOs), and can be grouped into five main categories. They are as follows:

1. Geometric chain models
2. Mass–spring models
3. Beam models
4. Cosserat models
5. Spline models

In actuality the term ODDO is somewhat of a misnomer as almost all suture models account for the thickness and therefore volume of the object. However it is still useful to think of it in these terms as the suture is often defined along

its length with forces applied to it in various directions. By thinking about it in these terms, it can be said that it is in fact defined in one dimension, with deformations applied in normal and cross–sectional directions.

In this section the five approaches listed above will be examined. An early paper is presented to give an idea of how the method works, then an example of a specific implementation in suturing simulations, or similar, is presented and evaluated. These implementations also have different methods of handling body interactions and model constraints, however these are discussed in the section which follows.

Geometric Chain Model

This is an efficient approach that is widely used, however it relies only on geometric information and no physics is directly modelled. It was first proposed by Brown, Latombe, and Montgomery [12] who present an algorithm called "Follow the Leader" (FTL) which is used to update the position of a simulated rope in each time step.

This model assumes no stretching, as only non–elastic ropes are considered. The discretised rope is composed of n straight rigid links connected by $n + 1$ spherical joins (nodes), forming a kinematic chain. The joints allow for two degrees of rotational freedom, and the links are all of the same unit length. Furthermore, each node may only move a maximum of δ in each step, so by ensuring δ is less than the radius, lagging is reduced.

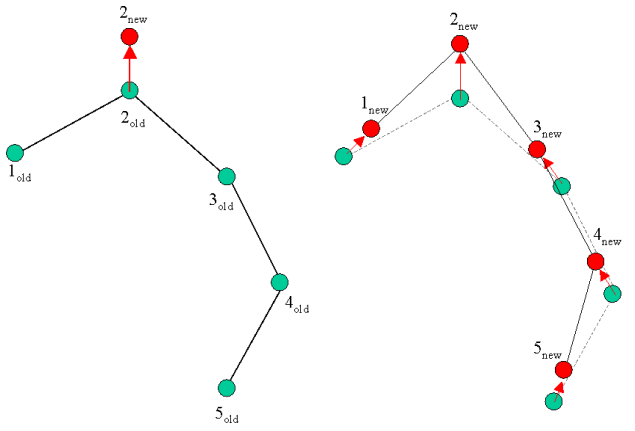
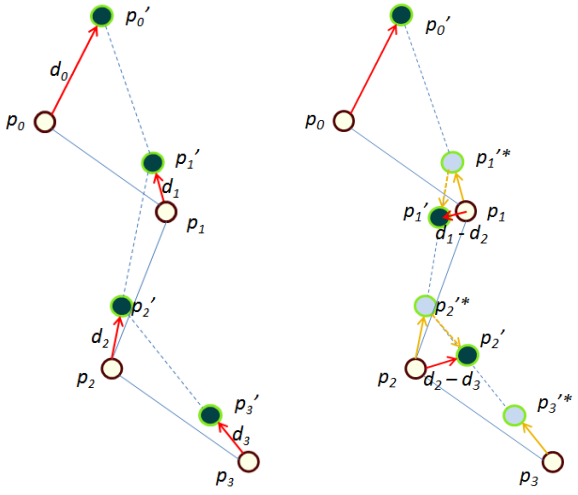


Figure 2.1: Example of FTL with (a) forced movement of grasped node 2, (b) update of position for nodes 1 and 3, and then node 4, and then node 5 [12]. [Reproduced with permission from Springer Nature]

This approach introduces the FTL algorithm, where movement applied to one node propagates throughout the rest of the nodes, working in the following way. If, during a given time step, a grasped node, N_i, is displaced from a position x_i^{old} to x_i^{new}, then its neighbouring node, N_{i+1}, with initial position x_{i+1}^{old}, is moved along the line connecting x_i^{new} and x_{i+1}^{old}, such that the two neighbouring nodes are still unit distance apart from each other. The motion is propagated in both directions until the new positions of N_0 and N_n have been computed. This is shown in Figure 2.1. Note that bending and torsion effects are not modelled.

This approach has been used more recently by Müller, Kim, and Chentanez [13] for use in real–time hair simulation. This model improves the FTL algorithm by introducing a velocity correction, and is called "Dynamic Follow the Leader" (DFTL).

Instead of only considering moving nodes to a new position in such a way that only the distance from its original position is minimised, this paper adds a dynamic element. The corrected direction for node N_i is calculated by subtracting the change in position for node N_{i+1}, as shown in Figure 2.2.



(a) (b)

Figure 2.2: Example of DFTL with (a) standard static FTL (b) velocity correction and resulting state [14]. [Reproduced with permission from the Association for Computing Machinery]

This model is a better candidate for the dynamic simulation of hair strands, but it is only reliant on the position and velocity of the nodes. There are no physical laws that are balanced, or energies modelled.

Another implementation of a geometric model is that of Umetani, Schmidt, and Stam [15], which can simulate bending and twisting deformation efficiently using the framework of position based dynamics. This is a step closer to modelling the real physics of ODDOs as elastic properties are included in the model. Although the model is very simple, motion appears visually realistic and physically plausible. For this reason it is often used in surgical simulators as it frees up resources for more complex tasks. Despite this, some fundamental problems exist as the method does not derive from the understood physical behaviour.

Although this model is efficient, it is not a mathematically accurate model of the underlying physics. Therefore physics–based approaches should be considered over this approach. Physics–based approaches allow for the modelling of processes with a complexity and accuracy that is impossible to achieve with a purely geometric approach [16].

Mass–Spring Model

The first of the physics based models is the mass–spring model. An early implementation of the mass–spring model for ODDOs is that of Rosenblum, Carlson, and Tripp III [17], the diagram for which is depicted in Figure 2.3.

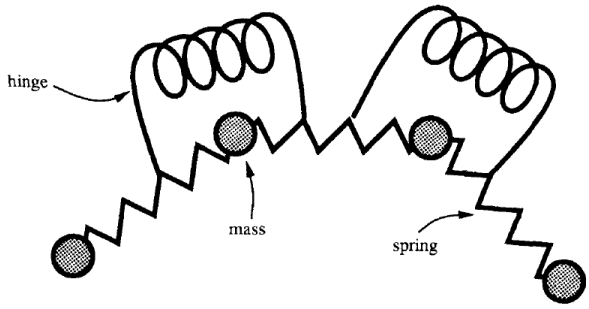


Figure 2.3: Example of mass–spring model [17]. [Reproduced with permission from John Wiley and Sons]

The figure shows point masses, with each pair of successive point masses connected by springs. The compression and extension of the spring connecting two masses determines the force on the masses, and is found using Hooke's law. The equation is as follows.

$$F_{\text{spring}} = k_{\text{spring}}(d - d_0)$$

where F_{spring} is the magnitude of the force on each point mass, k_{spring} is the spring constant (which controls the resulting strength of the force), d is the

current distance between the two point masses and d_0 is the initial distance (or the distance at rest) between the two point masses. If the spring is stretched then the force vectors at each mass point towards each other, and if the spring is compressed then the force vectors point away from each other. Furthermore, simple damping scheme is used by introducing a damping coefficient, D_{spring}, so the equation becomes

$$F_{\text{spring}} = k_{\text{spring}} (d - d_0) - D_{\text{spring}} (k_{\text{spring}} (d - d_0))$$

Figure 2.3 also shows hinges at the interior point masses. The force due to the hinge element is calculated in a similar way, resulting in the following equation.

$$F_{\text{hinge}} = k_{\text{hinge}}\theta - D_{\text{hinge}} (k_{\text{hinge}}\theta)$$

where F_{hinge} is the magnitude of the hinge force on the point mass in a direction that would straighten the ODDO, a direction perpendicular to the two neighbouring point masses, k_{hinge} is a hinge stiffness coefficient, D_{hinge} is the hinge damping coefficient and θ is the angular displacement of the hinge.

Finally the force due to gravity and an approximate aerodynamic drag force are included using the following equations.

$$\begin{aligned} F_{\text{gravity}} &= mg \\ F_{\text{drag}} &= vD_{\text{drag}} \end{aligned}$$

where F_{gravity} is the magnitude of the force due to gravity in the direction of gravity, m is the mass of the point in question, g is acceleration due to gravity (9.8m/s^2), F_{drag} is the force due to drag acting in the opposite direction to the velocity of the point mass, v, and D_{drag} is a drag damping coefficient. Additional hinge springs can be used to model torsion and provide bending resistance [18].

Once the forces have all been evaluated, the acceleration, a, of each point mass at a given time is calculated using the following equation.

$$a = F/m$$

where F is the sum of all forces acting on the point mass. This is then used to update the velocity of the point mass, and to calculate its new position for the next time step.

One example of this type mass–spring model being used to model sutures is by LeDuc, Payandeh, and Dill [19]. Suture models of varying complexities

are presented. The most complex of which includes point masses connected by longitudinal springs, dampers between the masses, torsion springs, torsion dampers and viscous damping effects. Ultimately, the authors chose a simpler model composed only of point masses connected by undamped longitudinal masses. A quasi–static integration method was used to update the positions of the nodes, and numerical viscous damping was used without increasing the complexity of the model or slowing down the calculation.

A second example from Wang et al. [18] models a higher level of complexity of the four degree of freedom (DoF) model. Three positional DoFs, x^1, x^2, x^3, determined by force, F, and one torsional angle DoF, q, determined by torsion force, τ. The forces modelled are as follows: stretching and compression, bending, twisting, dissipative friction force and contact forces (from the environment or self–collisions), while stretch and friction are used to define the torsional forces. Euler integration is then used to define the motion of all nodes at each time step, as follows, producing a realistic and efficient simulation.

$$(x_i, q_i)\,(t + dt) = p\,(F_i, \tau_i)$$

An example of the mass–spring model implemented for knot tying is by Phillips, Ladd, and Kavraki [20]. This will be analysed deeper in the constraints section.

The mass–spring model is straightforward to implement and computationally is rather efficient and easily parallelisable, as nodes rely only on their neighbours. However the model is still a big over–simplification of reality and is often composed of large discretisations. High curvature bending is only possible with a very fine decomposition. Furthermore, complex models, with many interconnected mass–springs, often suffer from low stability and high amounts of oscillations with insufficient damping. And finally Moore and Molloy [16] argue that the propagation of forces throughout the ODDO is delayed, however this is not the case if an implicit numerical solver is used. A method such as the Störmer–Verlet could be used, which is computationally efficient.

Beam Model

A simple implementation of modelling ODDOs using linked beam elements has been used for hair simulations by Anjyo, Usami, and Kurihara [21]. The hair strand is represented by an originally straight cantilever beam, discretised into k segments as shown in Figure 2.4.

The beam segments, s_i, are connected by nodes, p_i, and are all of length d. Any external global force, g, is distributed uniformly along the beam at

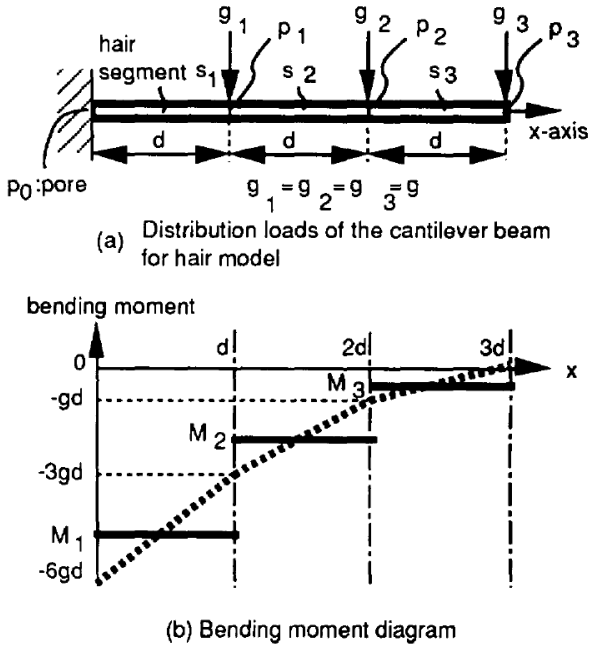


Figure 2.4: Example of cantilever beam model [21]. [Reproduced with permission from the Association for Computing Machinery]

the nodes as g_i. Assuming elastic deformation, the following equation defines bending.

$$\frac{\partial^2 y}{\partial x^2} = -\frac{M}{EI}$$

where the x–axis is along the resting beam direction and the y–axis is perpendicular to that. M is the bending moment at each node and EI is the flexural rigidity, which is the product of the material's Young's modulus, E, and the beam's second moment of inertia, I. So here we can see that the material properties are used directly, however only bending momentum deformation is modelled, and shearing force deformation is ignored. The moment at each node is calculated using the following equation.

$$\begin{aligned} M_i &= -\|g\| \, d \frac{\sum_{p=2}^{k-i+1} p_i + \sum_{p=1}^{k-i} p_i}{2} \\ &= -\|g\| \, \frac{d\,(k-i+1)^2}{2} \end{aligned}$$

From this the displacement, y_i, for node p_i can then be evaluated using the

following equation.

$$y_i = -\frac{1}{2}\left(\frac{M_i}{EI}\right)d^2$$

This is done successively from node p_1 to p_k with the deflection being relative to the tangent of the previous segment.

Dequidt et al. [22] present a much more advanced implementation of the beam model for simulating medical catheters based on the original work of Przemieniecki [23]. Guébert [24] applies the method used by Dequidt et al. [22] to medical sutures. This model takes into account bending, torsion and elongation by representing each element with an elementary stiffness matrix, $\bar{K}_e$, of size 12×12 which has all of the required equations for the different deformations. The global transformation matrix, Λ, is used to convert from local coordinates, $\bar{K}_e$, to global coordinates, K_e, using the following relationship.

$$K_e = \Lambda^T \bar{K}_e \Lambda$$

Note that the full stiffness matrix, K, is a banded matrix due to the geometric structure of the model. This global stiffness matrix is assembled by collecting each beam element and summing their contributions, K_e. This leads to the following equilibrium equation.

$$M\ddot{x} + D\dot{x} + Kx = f \tag{2.1}$$

where M is the mass matrix, which is diagonal due to the assumption of lumped masses at the nodes, D is the damping matrix, which is calculated from the Rayleigh damping equation, $D = \alpha M + \beta K$, and f is the external forces applied to the suture.

To solve the system of equations, an Euler implicit method is used to integrate the equilibrium equation. Since the beam model results in a tridiagonal block structure, an optimised inverse method is used to solve the system of equations. This method is efficient due to the optimisations, but these are reliant on the tridiagonal structure. With this method, the solvability is not dependent on the diagonal dominance of K_e. The solution is then computed with order $\mathcal{O}(n)$ instead of $\mathcal{O}(n^3)$.

This method is more similar to a corotational approach [25], as opposed to the incremental approach presented by Cotin et al. [26], who initially introduced the model for catheters. Cotin et al. [26]'s method of inverted mechanics uses the substructure of the stiffness matrix, K, to calculate $u = K^{-1}f$ without having to compute K^{-1} in its entirety. This method has that advantage

that it does not make the tridiagonal assumption, however it cannot represent geometric non-linearities, while the method from Dequidt et al. [22] can.

The beam approach is relatively straightforward for simple models and rather efficient, as [22], [24] and [26] show. However it is often a complex procedure to implement torsional stiffness. The approach has a high accuracy but it requires intensive calculations for more complex models.

Cosserat Model

The Cosserat rod model was first introduced by the brothers Cosserat and Cosserat [27], and is a special case of a more general theory of Cosserat continua which includes surfaces and points. Pai [28] was the first to implement this model in surgical simulations, and even computer graphics in general. Here, the configuration is described by the space curve $r(s)$ and a coordinate of frame "directors" at each point on the curve, as shown in Figure 2.5.

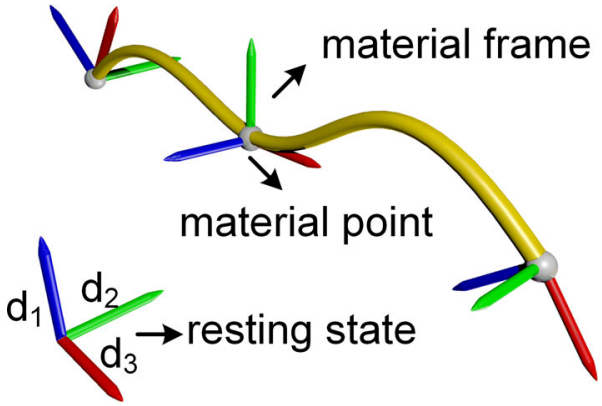


Figure 2.5: Example of a 2–segment Cosserat beam model [29]. [Reproduced under the terms of the Creative Commons CC BY license]

Each d_i is a vector, with d_3 along the tangent, and d_1, d_2 normal, such that $d_1 = d_3 \times d_2$. Collectively, these vectors are known as the Frenet–Serret frame. These directors are assembled in the coordinate frame $E(s) = [d_1\, d_2\, d_3\, r]\,(s)$. Deriving this along the abscissa results in the kinematic differential equation.

$$E' = \frac{dE}{ds} = \begin{bmatrix} [u] & v \\ 0 & 0 \end{bmatrix}$$

where u is the Darboux vector representing angular velocity, and v represents linear velocity. The Darboux vector is the result of differentiating the Frenet–Serret frame by the parametric abscissa, s. It must be noted that the linear and angular velocities are taken space–wise and not time–wise.

Note that the 3×3 matrix $[u]$ is the skew–symmetric matrix of the cross product, $u\times$. Constraints are handled by the stresses at s, defined by $\xi = \left(m^T, n^T\right)^T$, where m and n are the transmitted torque and force per cross–section area of the rod respectively. Deriving this along the abscissa results in the applied torque, τ, and the force, f, per unit length. Taking "$'$" to solely denote differentiation in relative coordinates results in the identity equation 2.3.

$$\frac{d\xi}{ds} = \eta = \left(\tau^T, f^T\right)^T \tag{2.2}$$

$$= \xi' + \begin{bmatrix} [u] & [v] \\ 0 & [u] \end{bmatrix} \xi \tag{2.3}$$

Combining these two leads to the stress differential equation.

$$\xi' = \eta - \begin{bmatrix} [u] & [v] \\ 0 & [u] \end{bmatrix} \xi$$

By using director vectors to calculate bending and torsion global displacements are modelled directly and nonlinear deformation effects can also be taken into account. Pai [28] only implements simple constitutive laws as follows, however it is possible to use the same idea for more complex models.

$$m = K\left(u - \hat{u}\right)$$
$$n = L\left(v - \hat{v}\right)$$

where $\hat{u}$ and $\hat{v}$ are the strains at rest, which could be straight or represent a coiled suture. For example Spillmann and Teschner [30] use the Finite Element Method (FEM) to model strain energy, and Bergou et al. [31] use the centreline for a discrete bending and twisting energies. Note that with Pai [28], the representation of the centreline is only implicit, which complicates the handling of collisions. Xu and Liu [29] combine the complexity of these approaches with Position-Based Dynamics (PBD) and a unified particle framework to create an accurate yet efficient simulation.

Kirchhoff models are related to Cosserat models, however they assume that there is no shear or tensile strain along the beam, meaning that the length remains constant (no stretching) and that the cross-section is always orthogonal to the centreline. Comparing the two approaches, the methods diverge with an increased cross-sectional area of the rod and for larger forces acting on it [32]. Material properties only have minimal effect on the discrepancies. So for thin ODDOs, subject to only small loads, both approaches produce similar results, and therefore the simpler Kirchhoff model can be considered.

Wang et al. [33] implement a thin Kirchhoff rod model to simulate surgical thread in an efficient way. It can operate as fast as 1ms per frame and is able to output axial forces, which can be used in haptic system. A geometric variational integration scheme is used that conserves energy and momentum.

Cosserat models produce realistic results and deform in a way that represents the actual behaviour of a suture with known properties. However, it is a particularly computationally expensive model, and due to its complexity it is difficult to constrain it away from its end points. This is largely due to coordinate parametrisaiton.

Spline Model

Splines present a precise and computationally lightweight approach in comparison to previous methods. With relatively few control points, a curve can be defined by interpolating between them with piece-wise polynomial basis functions, as it is defined continuously. The properties of any given point along the curve can be computed precisely, therefore it can be discretised into as many points as required without increasing the degrees of freedom of the model. This results in a more precise representation without the added complexity.

A spline curve, $\mathcal{C}$, has its degrees of freedom stored in the vector x_S. This vector contains all positional information. Assuming the spline is defined by $n+1$ control points, each control point selects from x_S to get its properties, p_i, which are the x, y and z space coordinates for that specific control point. The spline curve can then be defined with piece-wise polynomial basis functions as follows.

$$\mathcal{C} = \left\{ P\left(x_S, u\right) = \sum_{i=0}^{n} N_i\left(u\right) p_i \middle| \forall u \in [\text{begin}, \text{end}] \right\} \tag{2.4}$$

where u is the parametric abscissa along the curve and N_i is the basis function for a given control point. The basis functions N_i, and limits of u are dependent on the spline type. For example, a uniform B-spline would have the range $u \in [\text{degree}, n+1]$ (and a cubic uniform B-spline would have the range $u \in [3, n+1]$). The curve shown in Figure 2.6 is in fact a parametric spline and not a B-spline, however it does show the fact that the curve need not coincide with a particular control point in space. One simply interpolates between them. A difference is that uniform B-splines would not connect to the end nodes.

So even though the degrees of freedom are finite, the spline represents a continuous model.

An early implementation of dynamic splines in computer graphics was by Terzopoulos et al. [35], where they are used to describe general models in

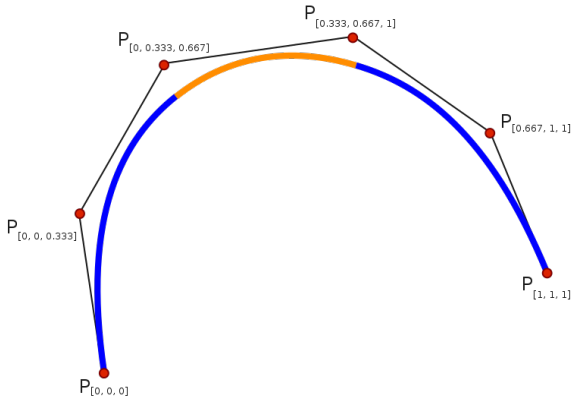


Figure 2.6: Parametric cubic spline [34]. [Reproduced under the Creative Commons Attribution-Share Alike 3.0 Unported license]

3d space that can undergo elastic deformation. This is done by solving their underlying differential equations. In the case of 1D models, the deformation energy is the strain energy, which is defined by three terms: the resistance of the curve to stretching, α, bending, β, and twisting, γ. It is formulated as follows.

$$\epsilon\left(r\right)=\epsilon\left(s,\kappa,\tau\right)=\int_{0}^{L}\alpha\left(s-\hat{s}\right)^{2}+\beta\left(\kappa-\hat{\kappa}\right)^{2}+\gamma\left(\tau-\hat{\tau}\right)^{2}du$$

with r representing the position of the curve, s, κ and τ representing the current stretching, bending, and twisting, and $\hat{s}$, $\hat{\kappa}$ and $\hat{\tau}$ representing the resting stretching, bending, and twisting. Note that all of these terms are dependent on $r\left(t\right)$, which is the position of the curve at time, t. The motion of the curve is then determined by representing the governing equations in Lagrangian form, which allows for the next state to be evaluated using a semi-explicit numerical integration scheme. The three terms of this represent the inertial force (due to the model's mass), the damping force (due to dissipation of forces) and the elastic force (due to deformation) as shown with the equation

$$\frac{\partial}{\partial t}\left(\mu\frac{\partial r}{\partial t}\right)+\gamma\frac{\partial r}{\partial t}+\frac{\partial\epsilon\left(r\right)}{\partial r}=f\left(r,t\right)$$

where $r\left(t\right)$ is the current position of the curve at time, t, μ is the mass density, γ is the damping density, and $f\left(r,t\right)$ is the external forces acting on the curve.

When presented as a constraint-free dynamic system, it takes the following

form.

$$\begin{bmatrix} M & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & M \end{bmatrix} \begin{bmatrix} \ddot{x}_x \\ \ddot{x}_y \\ \ddot{x}_z \end{bmatrix} = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} \tag{2.5}$$

where $\ddot{x}_i$ represents acceleration degrees of freedom in a given axis, F represents the sum of forces acting on the model and the generalised mass matrix, M, formed by $M_{ij} = m \int_{\mathbb{R}} N_i(s) N_j(s) ds$, where $N_i(s)$ is the spline's basis function for control point i at the parametric abscissa s, and mass m.

To encode the forces in the model, Lenoir et al. [36] considers two methods, one using springs to produce an efficient simulation, while another provides a strain energy by considering the continuity of the spline [37]. The paper [38] uses the method from the former to determine stretching, bending and twisting energies. Constraints are defined and added via the Lagrange multipliers technique. The system is then solved in each step with an implicit Euler numerical integration method.

Theetten [39] presents a comprehensive implementation for splines which is extremely rigorous in representing the underling mechanics, while also being flexible enough to be applied in many different problems with various constraint types, and efficient due to optimisation. A later paper [40] summarises the approach in English. Forces and moments are applied locally to the model by describing stretching, bending and twisting using the exact properties of the suture. Configurations are then described in terms of displacements and rotations. Furthermore, both elastic and plastic deformations are modelled. The plastic region is treated as perfectly plastic, in that it does not undergo any further elasticity. Finally the model breaks at a given yield stress. [40] also explores how the system can be constructed for efficiency.

2.2.2 Suture Constraints

Depending on the type of model chosen to represent the suture, different approaches must be taken to add constraints. Some models only allow for very rudimentary constraints, while others allow for any generic constraint dependent on position or velocity. In this section constraints relevant to splines will be examined.

Building from equation 2.5, Lenoir et al. [38] present a more rigorously defined spline model for suturing. Integrating the Lagrange multiplier method

for constraints into the model results in the following system.

$$\begin{bmatrix} M & 0 & 0 & -L_x^T \\ 0 & M & 0 & -L_y^T \\ 0 & 0 & M & -L_z^T \\ L_z & L_y & L_z & 0 \end{bmatrix} \begin{bmatrix} \ddot{x}_x \\ \ddot{x}_y \\ \ddot{x}_z \\ \lambda \end{bmatrix} = \begin{bmatrix} F_x \\ F_y \\ F_z \\ E \end{bmatrix}$$

where $\ddot{x}_i$ represents the acceleration of degrees of freedom in a given axis, $L = (L_x, L_y, L_z)$ is the constraint matrix, F represents the sum of forces acting on the model, E encodes the intensity of the violation of the constraint, and the generalised mass matrix, M, formed by $M_{ij} = m \int_{\mathbb{R}} N_i(s) N_j(s) ds$, where $N_i(s)$ is the spline's basis function for control point i at the parametric abscissa s, and mass m.

Lenoir et al. [38] present implementations of three constraints: the sliding point (allowing the spline to slide through a specific point in space), the sliding direction constraint (to ensure the suture is orthogonal to the organ at the point of entry) and the friction on sliding point constraint. The suture is simulated separately from the organ, therefore its constraints are simpler to handle. However for better results the whole system should be evaluated simultaneously.

In this approach, the position of the sliding constraint is determined by evaluating a new "free variable", s, which represents a parametric abscissa along the spline that, in turn, represents the position along the spline that is undergoing suturing. This free variable is added to the system with a small epsilon representing a fictitious inertial term, ϵ, to make the system simpler to solve. This adds artificial stiffness in order to reduce the differentiation index therefore the stability index also (see Section 3.4.3 for further understanding of this). As we will later see, this intentionally added error is not necessary for solving the system, however it is used in this case. The system could then be expanded to be as follows.

$$\begin{bmatrix} M & 0 & 0 & 0 & -L_x^T \\ 0 & M & 0 & 0 & -L_y^T \\ 0 & 0 & M & 0 & -L_z^T \\ 0 & 0 & 0 & \epsilon & -L_p^T \\ L_z & L_y & L_z & L_p & 0 \end{bmatrix} \begin{bmatrix} \ddot{x}_x \\ \ddot{x}_y \\ \ddot{x}_z \\ \ddot{s} \\ \lambda \end{bmatrix} = \begin{bmatrix} F_x \\ F_y \\ F_z \\ 0 \\ E \end{bmatrix} \tag{2.6}$$

The update each step, however, is actually evaluated in two parts: once for the "tendency" (how the model would update without constraints) and once for "correction" (the effects of the constraints). A better solution would be to evaluate the change in state *with* the constraints by considering it as a Differential Algebraic Equation (DAE).

Since updates to the position in space of the organ is computed before the suture, then the point in space through which the suture must pass is known. Therefore, the position constraint, g for the suture is defined as follows.

$$g(q, \dot{q}, t, s) = P(s, t) - P_0 \tag{2.7}$$

where q is the current state of all of the positional degrees of freedom for the suture, $\dot{q}$ is their current velocity, s is the relevant parametric abscissa for suturing, $P(s, t)$ is the relevant point along the suture and P_0 is the point on the organ through which the suture must pass. Next, the tangent of the suture is also fixed by using two unit vectors, u and v, that represent the normal of the plane of the organ, at the location of suturing. This is defined by two new constraints, e_1 and e_2, which ensure the dot product of these unit vectors with the tangent of the suture at point P, $T(s, t) = \frac{\partial P}{\partial s}(s, t)$, is equal to zero.

$$e_1(q, \dot{q}, t, s, u) = \frac{\partial P}{\partial s}(s, t) \cdot u = 0 \tag{2.8}$$

$$e_2(q, \dot{q}, t, s, u) = \frac{\partial P}{\partial s}(s, t) \cdot v = 0 \tag{2.9}$$

Theetten and Grisoni [41] present a development on this idea which is more robust, and allows for a wider range of constraint types. A set of Lagrangian constraints for 1-dimensional models are proposed which cover a large range of cases. These take the same general form of $g(\dots) = 0$, however are represented slightly differently in the full system. The structure of the full system is as before

$$\begin{bmatrix} K & J^T \\ J & 0 \end{bmatrix} \begin{bmatrix} \Delta^{n+1} X \\ -\lambda \end{bmatrix} = \begin{bmatrix} F \\ E \end{bmatrix} \tag{2.10}$$

where K is a stiffness matrix, F is the system force vector, $\Delta^{n+1} X$ is the displacement vector between the current state and the following one, E is the violation vector of the constraints, λ is the Lagrange multiplier vector and J is the derivative of the constraint vector equations, $g = 0$, with respect to the coordinates, also called the Jacobian of the constraints, i.e. $J = \frac{\partial g}{\partial x_i}$. In order to build this system, the differentiation of the constraint equations by the degrees of freedom must be determined for each constraint. This is done for all of the constraint types presented in the paper. Next, the magnitudes corresponding to the constraint forces, λ, must be determined, by efficiently using the violation, E.

Once all of the constraints have been added to the system in equation 2.10, the system is reordered as a banded matrix using the Cuthill-McKee algorithm

in order to reduce its bandwidth. So the overall improvements of this method include solving the tendency and corrections together and the use of a smart reorganisation of the matrix structure. This leads to very accurate results as well as an interactive simulation time, showing its suitability for real-time applications.

2.2.3 Tissue Model

As stated by Golec [42] in her PhD thesis, it is uncommon in existing methods of tissue simulation for high accuracy in simulating deformable bodies and real-time performance to go hand-in-hand. Therefore compromises are often made on some level either on the resolution or on the behaviour to make approximations that ensure real-time performance. This can range from large approximations such as pre-rendering deformations, to smaller ones such as assuming linear elastic deformations. However the simulation of deformable bodies is a broad field with many approaches.

Mass-Spring

The idea of constructing a Mass-Spring-based tissue model is to create often very complex geometry using point masses and connecting them together with springs, each given a certain stiffness. This simulates the elastic behaviour of soft tissue. Similarly to mass-spring sutures, configurations can be set to account for stretching, bending and twisting of the tissue.

In order to define the model with n points, the position of all points must be set, q_i, as must the mass of all points, m_i, the resting length of each spring connecting two masses, l_i, could be determined from the initial distance of the two masses, however the spring stiffness, k_i, must be set for each spring. Then any change to the length of the spring, δ, can be used to determine the deformation energy transferred to the spring as potential energy.

$$E_{\text{pot}}(\delta) = -\frac{1}{2}k\delta^2$$

As can the force due to the deformation.

$$F(\delta) = -\frac{\partial E_{\text{pot}}(\delta)}{\partial \delta} = k\delta$$

Problems with the mass-spring model include its susceptibility to wrinkling and buckling instabilities under compressive forces, which are especially pronounced for more complex models, and high amounts of oscillations. An

implementation by Golec [42] overcomes a lot of these issues to produce a highly accurate yet efficient mass-spring soft tissue model. The approach is a hybrid of correction force methods which is generic and produces physically accurate deformations. This shows that there is scope for using mass-spring systems in simulating soft tissue.

Finite-Element Model

The Finite-Element Method (FEM) is the most widely used approach for high-precision scientific applications. The idea is to numerically evaluate approximations of the real solutions of the model's underlying Partial Differential Equations (PDEs) which represent its physical behaviour such as elasticity and viscosity. They are the dynamic equations of motion and are build up with basis functions. When the model is treated as continuous this leads to a very high level of precision. Sifakis and Barbič [43] go into a lot of detail of how this can be achieved.

SOFA is a simulation frameworks that is widely used in medical simulations [44] [45]. Several implementations of FEMs are defined with certain variations, allowing the user to model objects including soft tissues. This has been used in many medical applications.

Others

Many other approaches to simulating soft tissue have been used, including Positional Based Dynamics [46], Meshless Deformations [47] and Unified Particle Framework [48] however they are beyond the scope of this project, as the focus is on the suturing aspect. The purpose of the preliminary tissue model is to be used to validate the suturing model.

2.2.4 Needle Model

In most suturing models, the needle tends to have the same underling model as the suture, but with different material properties of course. One example is by Guébert [24] who details the precise behaviour expected of the needle. This is based on previous work with Duriez et al. [49] where the behaviour of a needle during insertion is established, and a model is proposed to simulate this behaviour with constraints.

Another approach is to simulate the needle and the suture as two separate objects, and to simply link them with a constraint. The disadvantage here is that a new constraint must be added, which adds another condition that must

be resolved in each step. However the advantage of this method is that the two parts can be made very discontinuous in terms of material properties, without adding extra complexity or stiffness to the system.

2.2.5 Needle/Tissue Interaction

The paper [49] describes how the puncturing forces, the precise path of the needle tip due to steering and the resulting friction are all precisely modelled for beam elements. The constraints are formulated as with Lagrangian multipliers, using the synthetic formulation for the needle, n, and the tissue, t.

$$M_n\dot{v}_n = p_n - F\left(q_n, v_n\right) + H_n^T\lambda \tag{2.11}$$

$$M_t\dot{v}_t = p_t - F\left(q_t, v_t\right) + H_t^T\lambda \tag{2.12}$$

where q is the vector of generalised degrees of freedom, M is the mass matrix, v is the velocity vector, F is the vector of forces resulting from internal viscoelastic properties, p is the external forces, λ is the Lagrangian multiplier vector representing the constraint forces when multiplied by H^T (see Section 3.4.2 for more details). First the unconstrained system is computed. Next the constraint forces, λ, are computed for the violation by solving the constraint laws. Finally the model is corrected such that the system described by equation 2.11 and equation 2.12 holds.

The amount of correction required is calculated in each step by solving the constraint laws. In order to find the constraint force value a method similar to the Gauss-Seidel algorithms is used. In each iteration of the algorithm alternating constraints are visited to compute new values of λ_i. This leads to a unique solution of λ when the correction required falls below a certain threshold. This approach of solving constraints was later integrated into SOFA [45].

This model resulted in a simulation of needle insertion into soft tissue that is capable of representing complex behaviour. By validating it against the physical model of the needle and tissue, Moreira et al. [50] found only a small difference in behaviour. This lead to the conclusion that it is a feasible option to be used in surgical simulations.

Chapter 3

Method

3.1 Underlying Physics

In order to build a model to simulate real world physical behaviour, certain choices must be made with regards to the underlying physics. The formulation of the laws of mechanics will determine choices later on about how to implement different features. The system needs to be capable of modelling both rigid body and soft body dynamics. Therefore the Newton and the Newton-Euler formulations of mechanics are insufficient for this purpose.

Lagrangian mechanics is essentially another way of interpreting Newton's laws of motion, however it describes evolution based on energies (kinetic, potential...). It is a more general, flexible and mathematically sophisticated formulation than Newton and Euler's. It is flexible enough to model rigid body and soft body dynamics. For this case, it was therefore decided to use Lagrangian mechanics as the formulation for the physical laws of the system.

3.1.1 Lagrangian Mechanics

The fundamental formula of Lagrangian mechanics for a system with no external forces is as follows.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_j}\right) = \frac{\partial L}{\partial q_j} \text{ for all } j \in \{0, ..., n\} \tag{3.1}$$

with the Lagrangian, L, (defined by the energies of the system), $n+1$ degrees of freedom, q_j, which are also known as "generalised coordinates". These coordinates can represent anything from angles and positions, to control points for a spline (points that don't belong to the object) [51].

The idea of the Euler-Lagrange (EL) formulation is to split the external forces into a sum of potential and non-potential forces.

$$L = E_{kin} - E_{pot} \tag{3.2}$$

Hamilton's principle of least action dictates that the integral of this quantity is invariant with time. This quantity is known as the action, S. Mathematically expressed, this is stated as follows.

$$S = \int_{t_0}^{t_1} (E_{kin} - E_{pot})\, dt \rightarrow \text{stationary!} \tag{3.3}$$

This is true at all times. The action does not change value throughout trajectory through state space, and can therefore be used to define this trajectory. This is shown in Figure 3.1, where the system takes the most efficient path between two states.This definition can be used to derive both the weak and the strong form of the equations of motion.

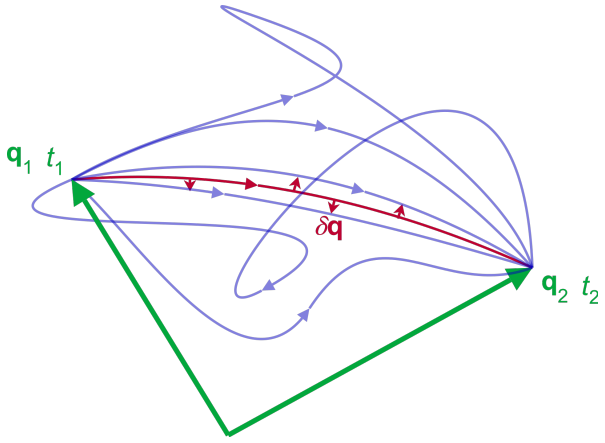


Figure 3.1: Hamilton's Least Action Principle [52]. [Reproduced under the Creative Commons CC0 1.0 Universal Public Domain Dedication]

In order to use this the kinetic energy and potential energy must be well-defined for the model either for the whole model (strong form) or at the infinitesimal scale (weak form). The kinetic and potential energy are both, of course, dependent on the degrees of freedom. Once they have been defined, the model can then be determined to be complete. Note that constraints in the model must be holonomic, meaning they can be described on the position-level, and independently defined. Next these definitions are inserted into equation 3.1. Note that by definition potential energy is only dependent on positional information, so $\frac{\partial E_{pot}}{\partial \dot{q}_j}$ goes to zero. If there are any external forces acting

on the model, these are added to the system as Q_j to the corresponding degree of freedom. The system is presented as follows.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_j}\right) - \frac{\partial L}{\partial q_j} = Q_j \tag{3.4}$$

$$\frac{d}{dt}\frac{\partial E_{kin}}{\partial \dot{q}_j} - \frac{\partial E_{kin}}{\partial q_j} + \frac{\partial E_{pot}}{\partial q_j} = Q_j \tag{3.5}$$

Finally, the degrees of freedom are collected into a system of equations that can be solved using a numerical solver method to update the system for the next step.

Potential Energies

Over the course of a simulations, energy will be transferred to objects in a way that deforms them. To model this, elastic deformations can be modelled by using the amount of deformation using Hooke's law. For example, the potential energy stored in a spring can be defined using the amount of deformation from its rest state, x. Hooke's law tells us that the force required to stretch the spring will be directly proportional to the amount of stretch.

$$f_{\text{deformation}} = -kx$$

where $f_{\text{deformation}}$ is the force required for the deformation, x is the amount of deformation and k is the spring stiffness. Therefore the change in potential energy is defined by the work done to stretch the spring by x. The potential energy is therefore defined by integrating the force required over the stretching distance as follows

$$E_{\text{pot, spring}} = \int_o^x k\rho \, d\rho$$

Assuming that the spring is perfectly elastic, the definition becomes

$$E_{\text{pot, spring}} = \frac{1}{2}kx^2$$

Similarly, a point mass can have potential energy due to gravity. By moving a point mass, m, a distance h in the y-direction (opposite to the direction of gravitational acceleration, g) the potential energy gain is

$$E_{\text{pot, gravity}} = mgh$$

Kinetic Energies

The definition for the kinetic energy for each object in the model follows the standard definition for kinetic energy.

$$E_{\text{kin}} = \frac{1}{2}mv^2$$

The precise definition will be dependent on the model. Differentiating this with respect to the velocities of the degrees of freedom, $\dot{q}_i$, and then with respect to time, t, leads to the first term of equation 3.5. The second term of equation 3.5 is found the same way as with the potential energy, by differentiating E_{kin} by the degrees of freedom, q_i.

3.2 Models

First the different models were designed and constructed. Each followed a rigorous mathematical model which was based on the analysis of other approaches. First the models of the objects in the system are described. These are also called "bodies" in the world of computer graphics.

3.2.1 Suture

Geometry

From the available choices of ODDOs the spline, as described in Section 2.2.1, was selected as the model for the suture. This was due to being highly precise and continuous wile being modelled by a relatively low number of control points. This means that it requires a low number of degrees of freedom when compared to other models.

The uniform cubic B-spline was chosen as the implementation to be able to model the suture so that it is defined up to the third derivative. By using equation 2.4 as the definition for the uniform cubic B-spline, it can be updated as follows by blending together the values of the four neighbouring control points, with the weight given by the basis functions.

$$\mathcal{C} = \left\{ P\left(x_S, u\right) = \sum_{i=0}^{n} N_i\left(u\right) p_i\left(x_S\right) \middle| \forall u \in [3, n+1] \right\}$$

where u is the parametric abscissa along the curve, the properties of the control points, p_i, are taken directly from the degrees of freedom, x_S, and the basis

functions, N_i, are those for the uniform cubic B-spline. The basis function for the cubic B-spline can be derived from the the Cox-de Boor [53] recurrence relation for B-splines. It starts with the definition for order-1 B-splines, $k = 1$, which is used to define the higher-order splines, $k > 1$. For a non-decreasing sequence of knots, $\{t_0, t_1, \cdots, t_{n+k}\}$, the basis functions for order-1 B-splines are defined by

$$N_{i,1}(u) \coloneqq \begin{cases} 1 & \text{if } u \in [t_i, t_{i+1}) \\ 0 & \text{otherwise} \end{cases}$$

For higher-order B-splines, $k > 1$, the basis functions are defined by

$$N_{i,k}(u) \coloneqq \left(\frac{u - t_i}{t_{i+k-1} - t_i}\right) N_{i,k-1}(u) + \left(\frac{t_{i+k-1} - u}{t_{i+k} - t_{i+1}}\right) N_{i+1,k-1}(u)$$

Note that for any given order of B-spline, these basis functions all satisfy

$$\sum_i N_{i,k}(u) = 1 \, \forall u \in [3, n+1] \tag{3.6}$$

Uniform cubic B-splines have a uniform knot sequence, so the knot sequence becomes

$$\{t_0, t_1, \cdots, t_{n+k}\} = \{0, 1, \cdots, n+k\}$$

By using this along with the recurrence relation for B-splines, the definition for a basis function for the cubic uniform B-spline becomes

$$N_{i,4}(u) = \frac{1}{6} \begin{cases} (u-i)^3 & \text{for } u \in [i, i+1) \\ (u-i)^2(i+2-u) + (u-i-1)^2(i+4-u) + (u-i)(u-i-1)(i+3-u) & \text{for } u \in [i+1, i+2) \\ (i+3-u)^2(u-i) + (i+4-u)^2(u-i-2) + (i+3-u)(i+4-u)(u-i-1) & \text{for } u \in [i+2, i+3) \\ (i+4-u)^3 & \text{for } u \in [i+3, i+4) \\ 0 & \text{otherwise} \end{cases}$$

For example, the region defining the start of the spline (between nodes 3 and 4) is defined by four basis functions as follows.

$$N_{i,4}(u)\Big|_{u\in[3,4)} = \frac{1}{6} \begin{bmatrix} (1-u)^3 \\ 4 - 6u^2 + 3u^3 \\ 1 + 3u + 3u^2 - 3u^3 \\ u^3 \end{bmatrix} \begin{matrix} \text{for } i = 0 \\ \text{for } i = 1 \\ \text{for } i = 2 \\ \text{for } i = 3 \end{matrix}$$

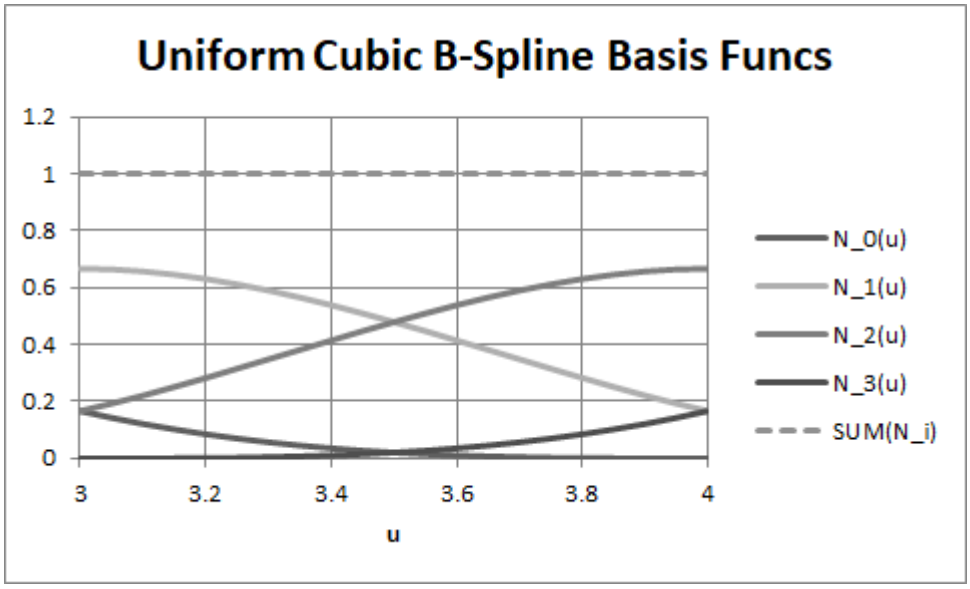


Figure 3.2: Uniform Cubic B-Spline Basis Functions.

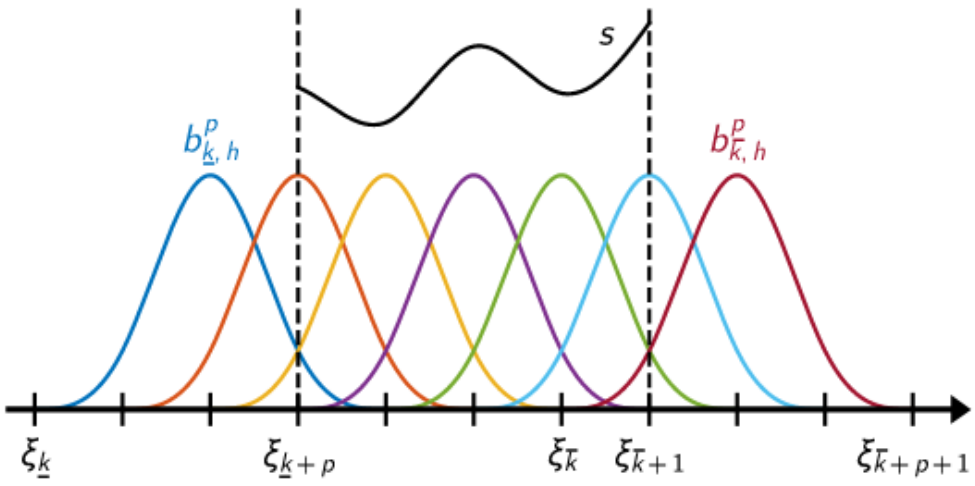


Figure 3.3: Uniform cubic B-splines basis functions forming a cubic spline [54]. [Reproduced under the Creative Commons Attribution-ShareAlike 4.0 International License]

which are shown in Figure 3.2.

Figure 3.3 shows how these basis functions combine to form a continuous curve, given that the coordinates of each control point are known.

The tangent of the curve can be found using the first derivative with respect to the parametric abscissa, u. This is defined by the following equation.

$$\partial_u \mathcal{C} = \tau_{\mathcal{C}} = \left\{ P'(x_S, u) = \sum_{i=0}^{n} N_i'(u)\, p_i(x_S) \middle| \forall u \in [3, n+1] \right\}$$

It is important to note that $P'(x_S, u)$ is not a unit vector as it also encodes stretching. $|P'(x_S, u)|$ determines the amount of stretching (or "extension") at a given parametric abscissa, u, and therefore dividing by this quantity gives the unit tangent at u. By extension the r^{th} derivative can be determined with the following equation.

$$\partial_{u^r} \mathcal{C} = \tau_{\mathcal{C}} = \left\{ P^{(r)}(x_S, u) = \sum_{i=0}^{n} N_i^{(r)}(u)\, p_i(x_S) \middle| \forall u \in [3, n+1] \right\}$$

Degrees of Freedom

The suture is defined by $n+1$ control points, which each have four degrees of freedom, stored in the vector q_i. This vector contains the x, y and z coordinates of the point, plus an angle which controls the twist of the spline. The full vector containing the degrees of freedom of the spline, x_S, therefore has the following form.

$$\begin{aligned} q_i &= \begin{bmatrix} p_i & p_i^\theta \end{bmatrix} = \begin{bmatrix} p_i^x & p_i^y & p_i^z & p_i^\theta \end{bmatrix} \\ x_S &= \begin{bmatrix} p_0^x & p_0^y & p_0^z & p_0^\theta & p_1^x & p_1^y & p_1^z & p_1^\theta & \dots & p_n^\theta \end{bmatrix} \\ &= \begin{bmatrix} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & \dots & x_{4n+3} \end{bmatrix} \end{aligned}$$

Kinetic Energy

As the simulation is evaluated using Lagrangian mechanics, the correct energies in the model must be determined. The kinetic energy of the model is defined as follows.

$$E_k = \iiint_3^{n+1} \frac{1}{2}\rho v^2 |p'| + \frac{1}{2} I_z \omega^2 |p'| \, du \, ds_1 \, ds_2$$

with material properties ρ, the mass density in $[kg \cdot m^{-3}]$, and I_z, the polar momentum of inertia in $[m^4]$. Furthermore, the definition relies on the parametric abscissa along the spline, u, and the curvilinear abscissae the directions of the cross sectional plane, s_1 and s_2.

Assuming a constant, orthogonal cross-section of radius, r, and that the mass density and inertia are also invariant along the curve, the kinetic energy can be simplified as follows. Note that $|p'|$ can be taken out of the integral as L_0, i.e. the initial total length of the suture.

$$E_k = \pi r^2 \, L_0 \left(\rho \int_3^{n+1} \frac{1}{2} v^2 \, du + I_z \int_3^{n+1} \frac{1}{2} \omega^2 \, du \right)$$

Since the cross section has been defined, I_z can now be entirely evaluated and therefore be considered constant, assuming that the cross section remains constant along the curve.

This leads to a definition for the first term of the Lagrangian equation, which is as follows, separating the positional degrees of freedom from the rotational degrees of freedom.

$$\begin{aligned}
\frac{d}{dt}\left(\frac{\partial E_k}{\partial \dot{q}_i}\right) &= \begin{bmatrix} \pi r^2 \, L_0 \, \rho \sum_{j=0}^{n} \left(\int_3^{n+1} N_i(u) N_j(u) \, du \right) & \ddot{p}_j(t) \\ \pi r^2 \, L_0 \, I_z \sum_{j=0}^{n} \left(\int_3^{n+1} N_i(u) N_j(u) \, du \right) & \ddot{\theta}_j(t) \end{bmatrix} \\
&= \sum_{j=0}^{n} \left(\pi r^2 \, L_0 \left(\int_3^{n+1} N_i(u) N_j(u) \, du \right) \begin{bmatrix} \rho & 0 & 0 & 0 \\ 0 & \rho & 0 & 0 \\ 0 & 0 & \rho & 0 \\ 0 & 0 & 0 & I_z \end{bmatrix} \right) \ddot{q}_j(t) \\
&= M \ddot{q}
\end{aligned}$$

where $\ddot{q}$ is the degrees of freedom's acceleration vector. From this we can determine the mass matrix, M, which is of size $4(n+1) \times 4(n+1)$. This matrix is composed of 4×4 blocks which have the following configuration.

$$M_{ij}^{(4\times4)} = \begin{bmatrix} M_{\text{position}} & 0 & 0 & 0 \\ 0 & M_{\text{position}} & 0 & 0 \\ 0 & 0 & M_{\text{position}} & 0 \\ 0 & 0 & 0 & M_{\text{rotation}} \end{bmatrix}$$

where

$$\begin{aligned}
M_{\text{position}} &= \pi r^2 \, L_0 \, \rho \int_3^{n+1} N_i(u) N_j(u) \, du \\
M_{\text{rotation}} &= \pi r^2 \, L_0 \, I_z \int_3^{n+1} N_i(u) N_j(u) \, du
\end{aligned}$$

Potential Energies and Internal Forces

In order to properly define the deformation energies of the suture, they must be formulated from its physical parameters. In order for them to be used in the Lagrange equations, they must then be differentiated by the degrees of freedom. The choice here was to use the definitions of the suture's internal forces in the Frenet frame to build the model. This is due to the fact that while being proportional to stresses, they are easier to compute. These are, in turn, defined by deformations.

The stretching, ϵ_s, bending , ϵ_b, and twisting, ϵ_t, deformation strains can be defined for any point along the curve of a suture and at any time by using the following definitions.

$$\epsilon = \begin{cases} \epsilon_s = 1 - \|r'\| \\ \epsilon_b = k = \frac{\|r' \times r''\|}{\|r'\|^3} = \frac{\mathcal{C}}{\|r'\|^3} \\ \epsilon_t = \theta' + \tau \end{cases}$$

where $\|r'\|$ is the norm of the local tangent vector, κ is the local curvature and τ is the geometric torsion of the curve, as defined by the Frenet frame.

A useful linear relationship which relates small stresses, σ, and strains, ϵ is from Hooke's law. It is stated as follows.

$$\sigma = \bar{E}\epsilon$$

where $\bar{E}$ is the 4^{th} order tensor called the stiffness tensor. This can be reduced to only two terms, the Young's modulus, E, and the Poisson's ratio, ν, from assuming that the suture's material is isotropic. Using the definition of the shear modulus, G, a relationship linking the longitudinal and transversal rates can be defined.

$$G = \frac{E}{2\,(1+\nu)}$$

The stresses can then be related to the strains with the following matrix.

$$\sigma = \begin{bmatrix} EA & 0 & 0 \\ 0 & EI & 0 \\ 0 & 0 & GJ \end{bmatrix} \epsilon \tag{3.7}$$

$$= H\epsilon \tag{3.8}$$

with Young's modulus, E, cross sectional area, A, cross sectional moment of inertia, I, and cross sectional polar moment of inertia, J. So now the energy

of the system can be defined at any point along the curve, and at any point in time.

$$E_p = \frac{1}{2}\epsilon^T \sigma$$

If it is assumed that the model starts with some strain, $\sigma^0 \neq 0$, then this leads to the following relationship.

$$E_p = \frac{1}{2}\left(\epsilon - \epsilon^0\right)^T H \left(\epsilon - \epsilon^0\right)$$

The potential deformational energies are independent from each other, as can be seen from the block diagonal structure of equation 3.7, therefore it can be separated into stretching, E_p^s, bending, E_p^b, and torsion, E_p^t.

$$E_p = E_p^s + E_p^b + E_p^t$$

Each of the deformation energies can be determined by integrating along the spline's arc length.

$$E_p^s(t) = \frac{1}{2}EA \int_0^L \left(\epsilon_s - \epsilon_s^0\right)^2 ds$$
$$E_p^b(t) = \frac{1}{2}EI \int_0^L \left(\epsilon_b - \epsilon_b^0\right)^2 ds$$
$$E_p^t(t) = \frac{1}{2}GJ \int_0^L \left(\epsilon_t - \epsilon_t^0\right)^2 ds$$

From this, the deformation forces can be evaluated.

$$f_i^s(t) = -\frac{\partial E_p^s(t)}{\partial q_i} = -EA \int_0^L \left(\epsilon_s - \epsilon_s^0\right) \frac{\partial \epsilon_s}{\partial q_i} ds$$
$$f_i^b(t) = -\frac{\partial E_p^b(t)}{\partial q_i} = -EI \int_0^L \left(\epsilon_b - \epsilon_b^0\right) \frac{\partial \epsilon_b}{\partial q_i} ds$$
$$f_i^t(t) = -\frac{\partial E_p^t(t)}{\partial q_i} = -GJ \int_0^L \left(\epsilon_t - \epsilon_t^0\right) \frac{\partial \epsilon_t}{\partial q_i} ds$$

Finally, the deformation stiffnesses can also be evaluated, as follows.

$$K_{ij}^s(t) = -\frac{\partial f_i^s(t)}{\partial q_j} = EA \int_0^L \left(\frac{\partial \epsilon_s}{\partial q_i} \otimes \frac{\partial \epsilon_s}{\partial q_j} + \left(\epsilon_s - \epsilon_s^0\right) \frac{\partial^2 \epsilon_s}{\partial q_i \partial q_j} \right) ds$$
$$K_{ij}^b(t) = -\frac{\partial f_i^b(t)}{\partial q_j} = EI \int_0^L \left(\frac{\partial \epsilon_b}{\partial q_i} \otimes \frac{\partial \epsilon_b}{\partial q_j} + \left(\epsilon_b - \epsilon_b^0\right) \frac{\partial^2 \epsilon_b}{\partial q_i \partial q_j} \right) ds$$
$$K_{ij}^t(t) = -\frac{\partial f_i^t(t)}{\partial q_j} = GJ \int_0^L \left(\frac{\partial \epsilon_t}{\partial q_i} \otimes \frac{\partial \epsilon_t}{\partial q_j} + \left(\epsilon_t - \epsilon_t^0\right) \frac{\partial^2 \epsilon_t}{\partial q_i \partial q_j} \right) ds$$

Note that instead of integrating over the length of the spline with s, the parametric abscissa, u, can also be used by using the relation $ds = |p'|du$, and changing the limits of the integral to those defined by the choice of spline function. The numerical integration method used to evaluate these forces and stiffnesses could be the trapezoidal method or a Gaussian quadrature for more precision.

Apart from deformation forces, it is possible to have other types of forces acting on the model. A major one is the gravitational force, which acts on all masses.

$$f_g = Mg$$

with the generalised gravity vector, g, for each of the degrees of freedom. Any other forces can be directly added to the model in this manner also.

A widely-used method of applying viscosity is with Rayleigh damping. It is defined as follows.

$$f_v = -\left(\alpha M + \beta K\right) v$$

with the velocity of the degree of freedom, v, mass matrix, M, stiffness matrix, K, Rayleigh mass coefficient, α, and Rayleigh stiffness coefficient, β. Rayleigh damping attenuates displacements proportionally to velocity and inertia. Low frequency vibrations are filtered with the mass coefficient, α, while high frequency vibrations are filtered through the stiffness coefficient, β.

3.2.2 Tissue

In order to simulate tissue for suturing, a mass-spring model was used. This was largely due to convenience, as much of what was required to build a mass-spring tissue model was available for use. Furthermore, the assumption of linear elasticity locally reduces complexity. The model is nonlinear geometrically as the strain is based on a distance. For this reason the force and stiffness is not constant over time, meaning they need to be re-evaluated in each step.

The development of a Finite Element Model (FEM) for tissue within `SGPhysics` would have taken much longer, as much of the work required for the mass-spring model was already complete. However, an FEM model would also have allowed for a more generalisable and versatile solution. Importing an FEM class from another package in the public domain, such as SOFA [45], goes against the design philosophy of `SGPhysics`, which is designed with the goal of having independence from external packages for flexibility and

architectural purposes. Furthermore, many of the lower-level features of packages such as SOFA are not available in the open-source version.

In order to construct an FEM class, base elements would have to be defined. 2D elements that sustain loading under bending stresses are *plates*, while elements that sustain tensile loading under in-plane stresses are *membranes*. The elements which encompass both are *shells*, and would be the appropriate choice for this application. By using 2D elements to construct a 3D model, one of the three dimensions is neglected in the modelling.

A mass-spring model still allows for a thorough investigation into simulating suturing and still provides physically-based behaviour while also being an interesting model to solve.

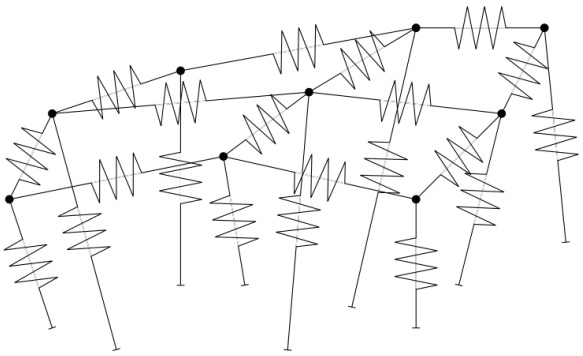


Figure 3.4: Schematic drawing of a mass-spring-model. The black dots represent mass points, the jagged lines between the dots are springs.

The tissue model is composed of point masses connected by springs in a way similar to Figure 3.4, but with a more rigorous structure. For a two-dimensional piece of tissue, point masses were laid out in a grid. Next, springs were added to connect neighbouring points, affecting longitudinal stretching. Next, springs were added to points diagonally adjacent to each other, which affects shearing. Finally springs were added to points that were two places away from each other (i.e. connecting every other point), which affects bending. From this, a complete model for a 2D section of tissue was defined.

The tissue therefore has the following material properties: mass (at each node), longitudinal stiffness, diagonal stiffness and bending stiffness. By adjusting these properties, the model can be made to represent different types of soft bodies with their own material properties.

Degrees of Freedom

Once the material properties have been defined and the model approximates a real tissue, then the only things that are variables that must be updated are the positions in space of the mass points. If the model is in 3D space, then each of these points will have three degrees of freedom, i.e. their x, y and z coordinates. So a tissue with $n+1$ mass points will have the following degrees of freedom stored in the matrix x_T.

$$
\begin{aligned}
q_i &= \begin{bmatrix} p_i \end{bmatrix} = \begin{bmatrix} p_i^x & p_i^y & p_i^z \end{bmatrix} \\
x_T &= \begin{bmatrix} p_0^x & p_0^y & p_0^z & p_1^x & p_1^y & p_1^z & \dots & p_n^x & p_n^y & p_n^z \end{bmatrix} \\
&= \begin{bmatrix} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & \dots & x_{3n} & x_{3n+1} & x_{3n+2} \end{bmatrix}
\end{aligned}
$$

Geometry

The geometry of the tissue is described as resulting from its underlying physical definition. It is essentially described by a set of 3D points and therefore triangles can be drawn connecting the points to form a complete surface. In order to refer to a point on this surface, the ID of a point can be used to refer to a specific defined point.

If an arbitrary point on the surface is required then barycentric coordinates can be used. In this case a point, P, on the surface of the tissue can be entirely defined from its three points (ABC) forming the triangle belonging to it and their respective weights, as shown in Figure 3.5. This weight is inversely pro-

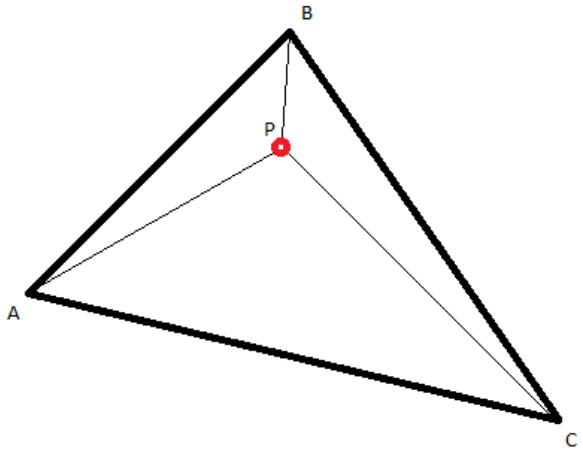


Figure 3.5: Point P defined from barycentric coordinates of the triangle ABC.

portional to the area formed by the point P and the opposite two points. This

influence is measured with two quantities, α and β, which when kept constant keep the relative position of P fixed to the surface of the tissue. It is defined as follows.

$$P = (1 - \alpha - \beta)\, A + \alpha B + \beta C$$

Note that for a given point, if the following properties hold,

$$0 \leq \alpha \leq 1 \tag{3.9}$$
$$0 \leq \beta \leq 1 \tag{3.10}$$
$$0 \leq (1 - \alpha - \beta) \leq 1 \tag{3.11}$$

then the point is inside the triangle.

3.2.3 Tools

In a given model tools could be built up using rigid body object types. That is, a geometric shape with position and rotation as its degrees of freedom. Position, as before, is described using x, y and z Cartesian coordinates, while rotations are described using quaternionic coordinates, $q = \begin{bmatrix} q_i & q_j & q_k & q_r \end{bmatrix}$. The rigid body therefore has seven degrees of freedom. The DoF time derivative is indirectly the set of linear and angular velocities. The angular velocities are the speeds of rotation about the x, y and z axes, therefore there are only six degrees of freedom in velocity.

Rigid bodies can be used as building blocks to create any kind of tool or other articulated object in the simulation. They can be combined together into systems of articulated rigid bodies. These bodies are attached to each other with joints, which are a type of constraints. Joints can be defined in a number of different ways, which differ by degrees of freedom or relative motion allowed.

The dynamics of a rigid body can be defined by combining the force and torque equations for each single rigid body, which are as follows.

$$\begin{aligned} F_i &= m_i a_i \\ T_i &= [I_R]_i\, \alpha_i + \omega_i \times [I_R]_i\, \omega_i \\ i &\in [1, n] \end{aligned}$$

for a system of n rigid bodies, where F_i is the force, T_i is the torque, α is the angular acceleration in vector form and ω is the angular velocity in vector form.

This leads to the Newton-Euler equation of motion for a rigid-body system with respect to the centre of mass.

$$\begin{bmatrix} F \\ \tau \end{bmatrix} = \begin{bmatrix} mI_3 & 0 \\ 0 & I_{\text{cm}} \end{bmatrix} \begin{bmatrix} a_{\text{cm}} \\ \alpha \end{bmatrix} + \begin{bmatrix} 0 \\ \omega \times I_{\text{cm}}\omega \end{bmatrix}$$

where F is the total force on the centre of mass, τ is the torque about the centre of mass, m is the mass, I_3 is the 3×3 identity matrix, a_{cm} is the acceleration of the centre of mass, I_{cm} is the moment of inertia about the centre of mass, ω is the angular velocity and α is the angular acceleration.

3.3 Constraints

In order to model interactions between the bodies, specific constraints were developed. These range from simple constraints such as fixing parts of two models to be coincident at all times, to more complex such as the suturing process between the suture thread and the tissue. More generally, a constraint is a relation between a subset of the degrees of freedom, which must always hold. Here the process for creating these constraints is described.

3.3.1 Fixed Points

In order to start defining the limits and constraints of the model, a good first start is the constraint to fix two points together. For example, if one of the points is an arbitrary point in space and the other is part of an object, then this will fix that point of the object to that point in space. In certain cases, it may be useful for the distance to be non-zero, such as in the case of the pendulum. The pendulum could however be modelled as a single rigid body with a rotation constraint at one end and a point mass added to the other.

The basic constraint has the following form.

$$g(x_0, x_1) = x_0 - x_1 = 0$$

where x_0 and x_1 are the respective points. If a point is defined by an object, or "body", then its definition is completely dependent on the body type. If it is a tissue, for example, then the point could be defined by the position of a specific node. It could also be defined by using barycentric coordinates to interpolate the relative position between multiple points. If the body were a suture, on the other hand, then the parametric abscissa, u, would be required to define a point along the spline.

3.3.2 Fixed Tangent

Fixed Tangent and Extension

Following from the logic of the previous constraint, it would seem logical to define a fixed tangent direction constraint in the same manner. The constraint for the suture could then be defined as follows.

$$g\left(x_t\right) = \tau\left(x_t\right) - \tau_0 = 0$$

where $\tau\left(x_t\right)$ is the tangent vector at the specified abscissa and τ_0 is the initial, or desired, value to fix the tangent vector to. The problem with this, as discussed in Section 3.2.1, is that this value also encodes the amount of stretching. So in order to fix the tangent without affecting stretching, then this tangent must be normalised by the amount of stretching. However a more elegant solution is to use the dot product to ensure that the angle formed with another vector remains constant.

Fixed Tangent Direction Only

In many cases, the vector of the tangent which the suture must be constrained to is not directly known but must be determined from a plane. This is the case with a suture entering tissue. The tissue is defined by points and therefore it is simple to map normal vectors along its surface. A fixed tangent constraint based on orthogonal vectors thus seems reasonable.

$$g\left(x_t\right) = \begin{cases} \tau\left(x_t\right) \cdot u_0 = 0 \\ \tau\left(x_t\right) \cdot u_1 = 0 \end{cases}$$

where τ, u_0 and u_1 are orthonormal. The vectors u_0 and u_1 each define a plane. This constraint definition ensures the tangent of the curve exists on both of these planes. That is, the tangent, τ, is normal to both u_0 and u_1, forming 90°angles. For a different fixed angle (i.e. a non-90°angle), the section concerning the full suturing constraint, Section 3.3.4, goes into more detail.

3.3.3 Joints

Joins are constraints that only allow motions parallel to certain axes, or restricts the movement of bodies to specific rotations. They can be defined using the dot product for 1-degree-of-freedom joints (i.e. parts that can only rotate about a single axis) or a fixed zero distance for 3-degrees-of-freedom joints (i.e. joints such as a ball-in-socket joint where two parts are joined at a specified point).

3.3.4 Suturing: Sliding Point and Fixed Tangent

The most important constraint type for this project is the actual suturing constraint. This is the suture passing through a point on the tissue, sliding through a specific point. This is therefore related to a fixed-point-in-space constraint between the suture and the tissue. The model becomes more complex however when we consider that this point actually slides along the suture over time. To resolve this, a new "free variable" is added to the model, u, which represents the parametric abscissa at which the suturing constraint is defined. This variable can be determined by solving the system as a whole, with the constraints correctly defined in the system. The position in space of the spline at u can then be evaluated by interpolating between the spline's piecewise functions for u, as follows.

$$P\left(x_S, u\right) = \sum_{i=0}^{n} N_i\left(u\right) p_i\left(x_S\right) \tag{3.12}$$

for u in the correct range, $u \in [\text{degree}, n+1]$, where there are $n+1$ control points, and the properties of the control points, p_i, are given by the degrees of freedom, x_S, and the piece-wise polynomial basis functions N_i are defined by u, and the choice of spline function (in this case, uniform cubic B-spline).

The suturing constraint is defined as follows.

$$g\left(x_T, x_S, u\right) = \begin{cases} P\left(x_S, u\right) - T\left(x_T\right) = 0 \\ \frac{AB(x_T)}{|AB(x_T)|} \cdot \frac{P'(x_S,u)}{|P'(x_S,u)|} = \alpha_0 \\ \frac{AC(x_T)}{|AC(x_T)|} \cdot \frac{P'(x_S,u)}{|P'(x_S,u)|} = \alpha_1 \end{cases} \tag{3.13}$$

where $P\left(x_S, u\right)$ is the position on the spline at the parametric abscissa, u. $T\left(x_T\right)$ is the point in the tissue where the suturing is occurring, as defined by positions of the point masses of the tissue, x_T. Lines 2 and 3 of this constraint will be explained below.

This first line of equation 3.13 ensures that the specified point along the suture and the specified point on the tissue are coincident. As previously described, the point along the suture is defined from a given parametric abscissa, u, which is initialised when suturing is activated to correspond to the tip of the suture. This occurs when the contact force of the suture tip on the tissue is above a specified threshold (see Section 3.4.7 on collision response for more information). At this point in time, the point on the tissue is also defined. As Figure 3.6 shows, only a section of the tissue is considered - that is, the 3 points forming the triangle in contact with the suture: A, B and C.

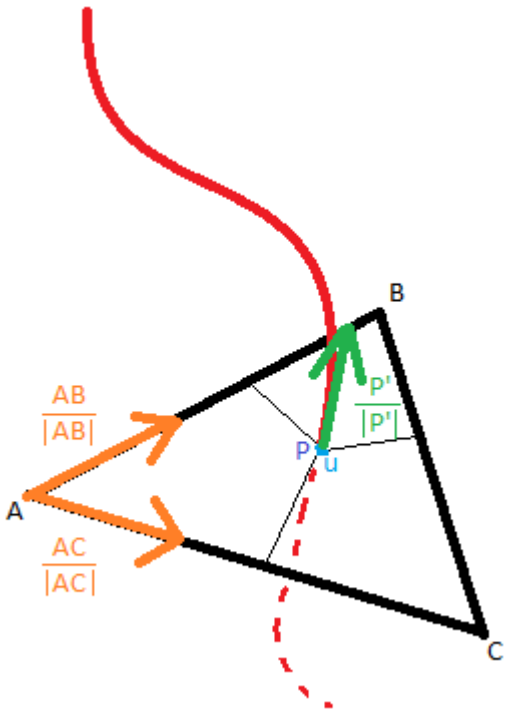


Figure 3.6: Suturing constraint. The active triangle of the tissue is shown in black, and the thread undergoing suturing is shown in red. They are coincident at point P, with the green arrow showing the unit tangent of the suture at this point, and the orange arrows showing the unit tangent at the surface of the tissue.

The location on the triangle of the suture tip can be defined from the three points using barycentric coordinates, as follows

$$T(x_T) = (1 - \alpha - \beta)\, p_A + \alpha p_B + \beta p_C$$

where p_A, p_B and p_C are the points of the triangle, and α and β are the constants that define the relative effect of the points on defining the suture point. α and β are kept as constant for the whole of the suturing process so that the suturing point on the tissue is constant relative to the defining triangle. The constraint can then be defined as the difference of these two points equals zero for the x, y and z directions. Therefore the first line of equation 3.13 is actually composed of three instances for the x, y and z directions respectively.

Now that the point has been fixed, the tangent of the suture must also be fixed. This is defined by lines 2 and 3 of equation 3.13, where $AB(x_T)$ is the line connecting point A and point B of the triangle as defined by positions of the point masses of the tissue, x_T, likewise for $AC(x_T)$. $P'(x_S, u)$ is the tangent of the spline at u, with control points defined by coordinates in 3D space, x_S. Similarly to equation 3.12, it is defined by

$$P'(x_S, u) = \sum_{i=0}^{n} N_i'(u)\, p_i(x_S)$$

which is the first derivative of equation 3.12 with respect to the parametric abscissa, u. Note that $P'(x_S, u)$ is not in fact a unit vector, but the length $|P'(x_S, u)|$ defines how stretched/elongated the curve is for any given parametric abscissa, u. Therefore in order to get the unit vector tangent direction at u, $P'(x_S, u)$ must be divided by $|P'(x_S, u)|$, as shown in equation 3.13.

The direction of this vector must be kept constant with respect to the triangle. The simplest way to define the direction in 3D space of the triangle is to use the points that are already defined. These are the points A, B and C. The unit normal vectors along the surface of the triangle can then be defined, as shown in figure 3.6, to be $\frac{AB}{|AB|}$ and $\frac{AC}{|AC|}$, as defined in the three space dimensions.

In order to define and fix the relative angle between the vectors, the dot product is used. When the suturing constraint is initialised, the dot product of the unit suture tangent is taken with each of the tissue surface tangents, and the resulting constant values are evaluated and stored as α_0 and α_1. These values are stored in the constraint and used to ensure that the angle formed by the suture and the tissue is kept constant for the duration of the suturing. This matches the observed physical behaviour of sutures passing through tissue, and also ensures a logical continuity between each step of the simulation.

Note that the new degree of freedom variable, u, is added to the system when the constraint is added. The side of the suture which has pierced the tissue is known, so therefore u is initialised to be the parametric abscissa which defined that end of the suture. For now, $\dot{u}$ is initialised to zero, but it could be estimated using a combination of the force at the suture tip, and the velocity of the point on the tissue where suturing is occurring (determined using the three points of the triangle, and the point's barycentric coordinates). If u is outside of the acceptable range then this means that the suture has completely passed through the hole. Therefore the constraint is destroyed and the variable u is removed from the system.

3.3.5 Knot

A knot occurs when a suture interacts with itself forming a particular structure. This can be detected using self-collisions of sections of the thread.

Self-collisions of the suture are called edge-edge collisions. Once two edge sections of the suture are determined to be in close-enough proximity in space, then a continuous collision detection algorithm is used by comparing the configuration in consecutive time steps. Then a collision response approach is used to resolve the collisions (see Section 3.4.7).

3.4 Mathematical Approach

3.4.1 Ordinary Differential Equations

In order to reduce the 2nd order ODE

$$Ma = F$$

to a 1st order ODE, the terms are re-written using a state vector, Y

$$Y = \begin{bmatrix} x \\ v \end{bmatrix}$$

where x is the positional state of all of the variables of the objects in the system (with the size of this referred to as `xDoF`, or "positional degrees of freedom"), and $\dot{x}$ is the time derivative of that, i.e. the change in state of all of the variables of the objects in the system. More specifically, v is the velocity and its size is referred to as `vDoF`, or "velocity degrees of freedom"). It can be assumed in most cases that $v \equiv \dot{x}$ however this is not the case for all models (such as rigid

bodies). Due to the fact that for some cases $v \not\equiv \dot{x}$, a transformation function, F_x, is used in the general case, using the following equation

$$\dot{x} = F_x\left(x, v\right)$$

however as this is not the case for the suture and tissue models, it will be assumed throughout that $v \equiv \dot{x}$. Therefore the use of the transformation function will be ignored. On the other hand, rigid multi-body systems are defined by seven `xDoF` (three positional degrees of freedom, and four quaternionic degrees of freedom), and six `vDoF` (three for the linear velocity, and three for the angular velocirty). Therefore the transformation function is required. However this type of problem is left to a future project.

The governing equation and the constraints combined into one larger system, which can be written as:

$$\bar{M} \cdot \dot{Y} = F\left(t, \bar{Y}\right) \tag{3.14}$$

which is a more general way of writing

$$\begin{bmatrix} I & 0 \\ 0 & M \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v \\ F \end{bmatrix} \tag{3.15}$$

where F represents the forces acting on the model.

This is the simplest formulation of a multi-body system, which is in fact an *unconstrained* system. The addition of certain types of rheonomous (time dependent) and scleronomous (time independent) constraints will lead to constraint forces applied to the system, as shown in Section 3.4.2. This simple formulation is nothing but the Newton equation, which states that force equals mass times acceleration.

3.4.2 Lagrange Multipliers

The constraints defined in Section 3.3 each define a manifold of allowed free motion. The addition of these constraints to the unconstrained system will produce forces, known as constraint forces, which are responsible for ensuring the constraints are satisfied. Once these constraints are added to the unconstrained system shown in Equation 3.14, the system extends to the following.

$$\left.\begin{aligned} \bar{M} \cdot \dot{Y} &= F\left(\bar{Y}\right) - F_c\left(\bar{Y}, \lambda\right) \\ 0 &= g\left(\bar{Y}\right) \end{aligned}\right\}$$

where $g\left(\bar{Y}\right)$ is a vector-valued function that describes the n_c constraints in the system, and $F_c\left(\bar{Y}, \lambda\right)$ are the additional forces on the system.

The existence of the additional forces ensures that the solution exists on the constraint manifold, and act in a direction orthogonal to the manifold, as shown by the d'Alembert principle. This leads to the following definition for the constraint forces.

$$F_c\left(x, v, \lambda\right) = G\left(x, v\right)^T \lambda$$

where

$$G\left(x, v\right) \coloneqq D_x g\left(x, v\right) = \frac{\partial}{\partial x} g\left(x, v\right)$$

is the differentiation matrix (Jacobian matrix) of g with respect to x, and λ is known as the Lagrange multiplier, which is of size n_λ.

The system shown in Equation 3.15 can also be expressed in matrix form with the constraints added, as shown here.

$$\begin{bmatrix} I & 0 & 0 \\ 0 & M & G^T \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{v} \\ \lambda \end{bmatrix} = \begin{bmatrix} v \\ F(x, v) \\ g\left(x, v\right) \end{bmatrix} \tag{3.16}$$

For holonomic constraints, the third line of the system described in equation 3.16 can be derived with respect to time twice and remain approximately equivalent [55]. The reason for this is explained in Section 3.4.3. If it is desired to have a symmetrical matrix on the left hand side of the equation, it is possible to do that by separating the constraint into the part that resembles, or is equal to G, and then the rest of it, ξ.

$$g(x) = 0 \tag{3.17}$$

$$\dot{g}(x) = G(x)\dot{x} = G(x)v = 0 \tag{3.18}$$

$$\ddot{g}(x) = G(x)\dot{v} + \left(\frac{d}{dt}G(x)\right) v = 0 \tag{3.19}$$

$$=: G(x)\dot{v} + \xi\left(x, v\right) \tag{3.20}$$

When added to the system, this becomes

$$\begin{bmatrix} I & 0 & 0 \\ 0 & M & G^T \\ 0 & G & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{v} \\ \lambda \end{bmatrix} = \begin{bmatrix} v \\ F(x, v) \\ -\xi\left(x, v\right) \end{bmatrix} \tag{3.21}$$

Note that equation 3.3 can also be updated to include the constraint forces acting on the system. It is as follows.

$$S = \int_{t_0}^{t_1} \left(E_{kin} - E_{pot} - G^T\lambda\right) dt \rightarrow \text{stationary!}$$

3.4.3 Differential Algebraic Equations

General Formulation

The ODE formulation of the problem is useful for evaluating the change of state over time, but they do not account for state restrictions. These state restrictions are usually algebraic equations (AEs). Examples of these are given in Section 3.3.

Ordinary Differential Equations	**Differential Algebraic Equations**	Algebraic Equations
$\dot{x} = 2x + 3y$	$\dot{x} = 2x + 3y$	$2x + 3y = 0$
$\dot{y} = 4x - y$	$0 = 4x - y$	$4x - y = 0$
"Change of state over time"	**"Both!"**	"State restrictions"

DAEs are a special class of problem which use ODEs to describe the change in state over time, and uses AEs to describe how the state is restricted. In other words, DAEs are systems of equations that contain ODEs and AEs.

A Differential Algebraic Equation is a system of equations containing differential equations and algebraic constraints.

More rigorously, the general form of a DAE is defined as follows.

$$F(t, x, y) = 0 \tag{3.22}$$

where $y(x) = \dot{x}$
with $F : \mathbb{I} \times \mathbb{D}_x \times \mathbb{D}_y \to \mathbb{C}^m$
where $\mathbb{I} = [\underline{t}, \bar{t}], \mathbb{I} \subseteq \mathbb{R}$
and $\mathbb{D}_x, \mathbb{D}_y \subseteq \mathbb{C}^n$ are open
and $m, n \in \mathbb{N}$

Note:
$f' = \frac{\partial f}{\partial x}$
$\dot{f} = \frac{\partial f}{\partial t}$

The trick we want is to find these values of position and velocity for a given time. By thinking of the problem this way we run into an issue. In reality one actually solves the equation $F(t, x, \dot{x}) = 0$ instead of equation 3.22. This is the difference between a function definition and an equation. Here $\dot{x}$ is ambiguous as it is at once a derivative of a differentiable function $x \to \mathbb{C}^n$ with respect to it's argument $t \in \mathbb{I}$, and also used as an independent variable of F. This arises from wanting F to determine a function x that solves the system of equations, with the idea of the system being:

$$F(t, x(t), \dot{x}(t)) = 0$$

for all $t \in \mathbb{I}$. By thinking of problems like this we begin to get into the mindset for solving DAEs.

Example

One of the main fields of research for DAEs has been its application to multi-body physics. Multi-body physics describes dynamic behaviour of interconnected rigid or flexible bodies, each of which may undergo large translational and rotational displacements. One of the simplest problems in multi-body physics is that of the 2D pendulum, shown in Figure 3.7.

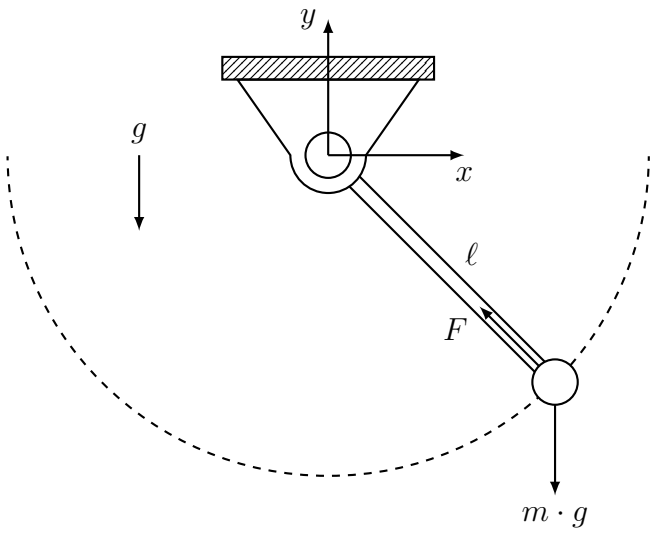


Figure 3.7: Pendulum example, with x and y axis directions, g as acceleration due to gravity, m as the point mass at the tip of the pendulum, ℓ as the length of the pendulum, and F as the restoration force.

For any given moment in time the motion is independent of the mass, m. The pendulum can thus be described with a DAE of the form:

$$\left.\begin{aligned} \ddot{x} &= -2x\lambda \\ \ddot{y} &= -g - 2y\lambda \end{aligned}\right\} \quad \leftarrow \text{State equations from Lagrangian Mechanics}$$

$$0 = x^2 + y^2 - \ell^2 \quad \leftarrow \text{Position constraints}$$

with
λ, Lagrange multiplier (constraint forces)
g, gravity
ℓ, length of pendulum

The top two equations are the state equations are are derived from the Lagrangian formulation of the laws of mechanics. This is a formulation based around the idea of conservation of energy. The symbol λ is the Lagrangian multiplier, and can be thought of as the forces on the model that arise from the constraints.

The bottom equation, on the other hand, is the position constraint. It defines the limited region in which the end-point of the pendulum can exist. This is defined using the pendulum's x and y components, and Pythagoras' Theorem.

By looking at the system, we can see a few things about it:

- 2^{nd} order differential equation: This is from the second derivatives of position (acceleration) being present in the state equations
- nonlinear constraints: The position variables in the constraint equation have a squared term
- semi-explicit: Expressed in two separate parts, one with a function of position describing derivatives of the position (the state equations) and another function of position which equals zero (the algebraic equations)

Indexes

In order to properly analyse DAEs, many index concepts have been developed that assist with questions of existence, uniqueness and stability of solutions, and to suggest approaches to find a solution. What follows is a list of the most commonly used index types, however only the first three will be explored briefly here.

- Differentiation index
- Strangeness index
- Perturbation index
- Nilpotency index
- Geometric index
- Tractability index
- Structural index
- ...

One can think of the index as an integer reflecting the complexity of the internal structure of the system [56].

For a better overview of different index types, see Mehrmann [57].

The **Differentiation Index**, ν, is the number of times all or part of the system must be differentiated in order to determine $\dot{x}$ in terms of t and x as a continuous function. With this definition, a derivative array can be built such that the solution x is uniquely defined for all initial values with $\ell = \nu$.

$$F_\ell(t, x, \dot{x}, ..., x^{(\ell+1)}) = \begin{bmatrix} F(t, x, \dot{x}) \\ \frac{d}{dt}F(t, x, \dot{x}) \\ \vdots \\ \left(\frac{d}{dt}\right)^\ell F(t, x, \dot{x}) \end{bmatrix} = 0$$

This new formulation is known as the "inflated system" which represents the "underlying ODE". For instance, the pendulum example described in Section 3.4.3 has $\nu = 3$. The major drawback of this definition is that it requires unique solvability.

Once the derivative array has been formed, then the following Jacobians can be defined.

$$M_\ell \left(t, x, \dot{x}, ..., x^{(\ell+1)}\right) = F_{\ell;x,\dot{x},...,x^{(\ell+1)}} \left(t, x, \dot{x}, ..., x^{(\ell+1)}\right)$$
$$N_\ell \left(t, x, \dot{x}, ..., x^{(\ell+1)}\right) = -\left(F_{\ell;x} \left(t, x, \dot{x}, ..., x^{(\ell+1)}\right), 0, ..., 0\right)$$

So for a linear DAE of the form $E\dot{x} - Ax = f$, this leads to the inflated differential algebraic equation:

$$M_\ell \dot{z}_\ell = N_\ell z_\ell + g_\ell$$

where

$$(z_\ell)_j = x^{(j)}, \text{ for } j = 0, ..., \ell$$
$$(g_\ell)_j = f^{(j)}, \text{ for } j = 0, ..., \ell$$
$$M_\ell = \begin{bmatrix} E & & & & \\ \dot{E} - A & E & & & \\ \ddot{E} - 2A & 2\dot{E} - A & E & & \\ \vdots & & \ddots & \ddots & \\ E^{(\ell)} - \ell A^{(\ell-1)} & \cdots & \cdots & \ell\dot{E} - A & E \end{bmatrix}$$
$$N_\ell = \begin{bmatrix} A & 0 & \cdots & 0 \\ \dot{A} & 0 & \cdots & 0 \\ \ddot{A} & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ A^\ell & 0 & \cdots & 0 \end{bmatrix}$$

The **Strangeness Index**, μ, is a generalisation of the differentiation index for over- and underdertermined systems. It indicates the number of global transformations required to reformulate the DAE into two parts:

- one part which states all constraints
- another part which describes the dynamical behaviour

For instance, the pendulum example described in Section 3.4.3 has $\mu = 2$.

It can be thought of like how an algorithm to solve faces of a Rubik's Cube must be done X times. Once X global transformations have been done on the (inflated) system of equations, then the system is strangeness-free, so the combined Jacobian is nonsingular and a solution can be determined. The original formulation has strangeness $\mu = X$. For a more formal, mathematical description of strangeness see Kunkel and Mehrmann [58], p74.

The **Perturbation Index**, κ, is a measure of sensitivity of solutions with respect to perturbations of the problem [58]. κ is the smallest number such that the perturbation η, defined by

$$F(t, \hat{x}, \dot{\hat{x}}) = \eta$$

satisfies

$$\|\hat{x} - x\| \leqslant C \left(\|\underline{\hat{x}} - \underline{x}\|_\infty + \|\eta\| + \|\dot{\eta}\| + ... + \left\|\eta^{(\kappa-1)}\right\| \right)$$

with a constant C independent of $\hat{x}$.

Specific Formulation

Constraints of the following form are introduced into the ODE system.

$$g(t,x) = 0$$

The system is now defined as

$$\begin{cases} M(t,x) \cdot \ddot{x} + G(t,x)^T \cdot \lambda = f(t,x) \\ g(t,x) = 0 \end{cases} \tag{3.23}$$

where

$$G(t,x) = D_x g(t,x)$$

is the Jacobian matrix of $g(t,x)$ with respect to x. One may think of it akin to $\frac{\partial}{\partial x}$.

Equation 3.23 can alternatively be formulated as follows.

$$\begin{bmatrix} M & G^T \\ G & 0 \end{bmatrix} \begin{bmatrix} \dot{v} \\ \lambda \end{bmatrix} = \begin{bmatrix} F \\ g \end{bmatrix} \tag{3.24}$$

This formulation will be useful for later.

Definition 3.4.1. Hessenberg DAE of index ν: [58] A DAE is of Hessenberg from of index $\nu \geq 2$, if has the following form

$$\begin{aligned} \dot{x}_1 &= f_1(x_1, \ldots, x_{\nu-1}, x_\nu) \\ \dot{x}_2 &= f_2(x_1, \ldots, x_{\nu-1}) \\ \dot{x}_3 &= f_3(x_2, \ldots, x_{\nu-1}) \\ &\vdots \\ \dot{x}_{\nu-1} &= f_{\nu-1}(x_{\nu-2}, x_{\nu-1}) \\ 0 &= f_\nu(x_{\nu-1}) \end{aligned}$$

with

$$\frac{\partial f_\nu}{\partial x_{\nu-1}} \cdot \frac{\partial f_{\nu-1}}{\partial x_{\nu-2}} \cdots \frac{\partial f_2}{\partial x_1} \cdot \frac{\partial f_1}{\partial x_\nu}$$

nonsingular for all relevant points $(x_1, \ldots, x_\nu)$. Furthermore, a DAE of the from

$$\begin{aligned} \dot{x} &= f(x,y) \\ 0 &= g(x,y) \end{aligned}$$

is a Hessenberg DAE of index 1, provided that the Jacobian, $\frac{\partial g(x,y)}{\partial y}$, is nonsingular.

Reducing the Index of DAEs

A small perturbation index, κ, reduces the sensitivity of the system. Furthermore this ensures only small iteration errors in Newton's method, and therefore the robustness of the solvers. Problems can be reformulated to reduce the index *before* computation leading to robust and efficient dynamic simulation of constrained systems. For instance, instead of solving a multi-body system with constraints on the position level (with differentiation index-3, $\nu = 3$) in each step, the problem can be reduced to an index-1 problem first, with constraints described on the acceleration level. This is done to reduce the degree of ill conditioning.

The system of equations governing a constrained multi-body physics can be expressed in the following form, with constraints on the position level.

$$\begin{aligned} \dot{x} &= v \\ M(x) + G(x)^T \lambda &= F(x, v) \\ g(x) &= 0 \end{aligned}$$

were the mass matrix $M(x)$ is symmetric and positive definite, and the constraints are (locally) independent such that the Jacobian matrix $G(x)$ has full row rank. Setting $x_1 = v$, $x_2 = x$ and $x_3 = \lambda$ leads to the following formulation

$$\begin{aligned} \dot{x}_1 &= f_1(x_1, x_2, x_3) = M(x_2)^{-1} \left(F(x_2, x_1) - G(x_2)^T x_3 \right) \\ \dot{x}_2 &= f_2(x_1, x_2) = x_1 \\ 0 &= f_3(x_2) = g(x_2) \end{aligned}$$

which is a Hessenberg DAE of index 3.

Now if the system of equations is written as in 3.14, with constraints on the position and velocity levels (by differentiating the constraint equations), the system is as follows.

$$\begin{aligned} \bar{M} \cdot \dot{\bar{Y}} &= F\left(\bar{Y}\right) \\ \bar{g}\left(\bar{Y}\right) &= 0 \end{aligned}$$

This is clearly a Hessenberg DAE of index 1.

In order to connect the reduction of the differentiation index with the reduction of the perturbation index, a more formal definition of the the perturbation index is required, as it was first defined by Hairer, Roche, and Lubich [59].

Definition 3.4.2. Perturbation Index: [58] The DAE

$$F(t, x, \dot{x}) = 0$$

with solution x, has a perturbation index κ along x on $[\,\underline{t},\, \bar{t}\,]$ if κ is the smallest number such that for all functions $\tilde{x}$ satisfying the perturbed DAE

$$F(t, \tilde{x}, \dot{\tilde{x}}) = \delta(t)$$

the defect $\delta(t)$ is satisfied by the estimate

$$\|\tilde{x} - x\| \leq c \left(\|\tilde{x}_0 - x_0\| + \max_{\underline{t} \leq \tau \leq \bar{t}} \|\delta(\tau)\| + \cdots + \max_{\underline{t} \leq \tau \leq \bar{t}} \left\|\delta^{(\kappa-1)}(\tau)\right\| \right)$$

for a constant c independent of x, for all $t \in [\,\underline{t},\, \bar{t}\,]$.

A DAE is said to have a perturbation index, $\kappa = 0$ if the estimate

$$\|\tilde{x} - x\| \leq c \left(\|\tilde{x}_0 - x_0\| + \max_{\underline{t} \leq \tau \leq \bar{t}} \left\| \int_{\underline{t}}^{\tau} \delta(s) ds \right\| \right)$$

holds. The perturbation index shows the sensitivity of the system to instabilities. In other words, the lower the perturbation index on the system, the lower the error resulting from numerical inaccuracies.

Differentiating the final equation of the Hessenberg DAE, $f_\nu(x_{\nu-1})$, j times, leads to the following formulation.

$$\begin{aligned}
\dot{x}_1 &= f_1(x_1, \ldots, x_{\nu-1}, x_\nu) \\
\dot{x}_2 &= f_2(x_1, \ldots, x_{\nu-1}) \\
\dot{x}_3 &= f_3(x_2, \ldots, x_{\nu-1}) \\
&\vdots \\
\dot{x}_{\nu-1} &= f_{\nu-1}(x_{\nu-2}, x_{\nu-1}) \\
0 &= g^{(j)}(x_{\nu-i-j}, \ldots, x_{\nu-1})
\end{aligned}$$

and this DAE would have perturbation index $\kappa = \nu - j$. Hence, by reducing the differentiation index, the perturbation index is also reduced, which leads to a more stable mathematically equivalent formulation. This new formulation, however, permits additional solutions for given a initial state. By integrating to get back up to the position level, numerical errors are introduced. Therefore a projection or stabilisation step is used to ensure all constraints are satisfied.

Drift-Off

The disadvantage of using these index-reduction techniques is the appearance of the drift-off effect. This is due to large constraint residuals, $g(x)$, which increase as time, t, increases.

The analytical solution to the index-2 formulation of the problem actually satisfies $g(x)$ for all t, however due to discretisation and round-off errors, the numerical solution is bounded by a small constant, ϵ, which grows linearly with time, t.

$$\|\hat{g}(t_n, x) - g(t_n, x)\| \leq \|\hat{g}(t_0, x) - g(t_0, x)\| + \int_{t_0}^{t_n} \epsilon \, dt = \epsilon \cdot (t_n - t_0)$$

The numerical solution to the index-1 formulation therefore suffers from a quadratic error growth.

$$\|\hat{g}(t_n, x) - g(t_n, x)\| \leq \epsilon \cdot (t_n - t_0)^2$$

The solution, x, therefore drifts from the solution manifold where the constraints are satisfied. In Section 3.4.6 the method to resolve this issue is described. This is the use of a projection method in each time step.

3.4.4 ODE Integration Schemes

As an overview of different methods, the main first order integration schemes are reviewed here. These are the most fundamental methods of simulating a dynamic model in each step. Higher order methods were considered but required a much larger overhaul of the existing software architecture.

Explicit Euler Scheme

This approach uses only known quantities of the previous step. The next step is computed using the following explicit formula.

$$\begin{aligned} v_{n+1} &= v_n + hf(x_n, t) \\ x_{n+1} &= x_n + hv_n \end{aligned}$$

where h is the time step size. Note that changing derivatives are not taken into account. However, computational efficiency arises from only being required to compute f in each step.

Semi-Explicit Euler Scheme

Instead of taking the velocity of the previous time step, the semi-explicit Euler approach is to first compute the velocity for the current time step, v_{n+1}, and use that to compute the position at the current time step, x_{n+1}.

$$\begin{aligned} v_{n+1} &= v_n + hf(x_n, t) \\ x_{n+1} &= x_n + hv_{n+1} \end{aligned}$$

An advantage of this method is that the energy of the system is much closer to being conserved. Like the explicit Euler scheme, however, a small time step must be used to ensure stability.

Implicit Euler Scheme

In this approach, the system is updated using $\Delta x = x_{n+1} - x_n$ and $\Delta v = v_{n+1} - v_n$. The constraint-free system then becomes

$$\begin{aligned} \Delta v &= hf(x_{n+1}, v_{n+1}, t_n) \\ \Delta x &= hv_{n+1} \end{aligned}$$

where f is evaluated using Newton's method. This can be done by linearising the problem. The solution then results from finding the root.

$$f(x_{n+1}, v_{n+1}, t_n) \approx f_n + \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial v}\Delta v \tag{3.25}$$

The system can thus be evaluated as a numerical linear algebraic system of the form

$$A\Delta v = b$$

which can be solved with an LU decomposition of a matrix with complete pivoting, leading to a highly-accurate factorisation result, or with an approximated method (which can be computed to a desired accuracy) such as conjugate gradient, which can be parallelised to reduce computation time. Furthermore, implicit Euler schemes are unconditionally stable. This means that an arbitrary time step can be used, while explicit schemes are only stable for small time steps. Smaller time steps are still important however for reducing the number of iterations required to converge and exhibiting dynamical phenomena such as vibrations and oscillations.

3.4.5 Linearisation of the Equations of Motion

Referring to the process of linearisation from Eich-Soellner and Führer [55], the system is built using the time dependent functions F and g, which are the applied forces and constraints respectively. The time-dependency of the applied forces is described using an input/forcing function $u(t)$, and time-dependency of the constraints is described by an adaptive kinematic excitation, $z(t)$, such that the system becomes

$$\dot{x} = v \tag{3.26}$$
$$M(x)\dot{v} = F\left(x, v, u(t)\right) - G(x)^T\lambda \tag{3.27}$$
$$0 = g(x) - z(t) \tag{3.28}$$

It is assumed that $u(t)$ and $z(t)$ are small deviation from their nominal values, which are assumed to be zero.

$$u_N(t) = 0$$
$$z_N(t) = 0$$

The nominal solutions, $x_N(t)$, $v_N(t)$ and $\lambda_N(t)$, have initial values $x_N(0)$, $v_N(0)$ and $\lambda_N(0)$. Then set the following.

$$x(t) = x_N(t) + \Delta x(t)$$
$$v(t) = v_N(t) + \Delta v(t)$$
$$\lambda(t) = \lambda_N(t) + \Delta\lambda(t)$$

This allows the system described by equations 3.26 - 3.28 to be expanded in a Taylor series around the nominal solution and around $u(t)$ and $z(t)$. Assuming that all products of changes of variables (i.e. variables with a Δ) are negligible, this leads to the following.

$$\Delta\dot{x} = \Delta v \tag{3.29}$$
$$M(t)\Delta\dot{v} = -K(t)\Delta x - D(t)\Delta v - G(t)\lambda + B(t)u(t) \tag{3.30}$$
$$0 = G(t)\Delta x - z(t) \tag{3.31}$$

with the mass matrix, $M(t)$, the stiffness matrix, $K(t)$, damping matrix, $D(t)$,

constraint matrix, $G(t)$, and input matrix, $B(t)$, where

$$M(t) \coloneqq M(x_N(t)) \tag{3.32}$$

$$K(t) \coloneqq \left(\dot{v}_N^T \frac{\partial}{\partial x} M(x) - \frac{\partial}{\partial x} F\left(x, v_N(t), 0\right) + \frac{\partial}{\partial x} G(x)^T \lambda_N \right)_{x=x_N(t)} \tag{3.33}$$

$$D(t) \coloneqq -\left(\frac{\partial}{\partial v} F\left(x_N(t), v, 0\right) \right)_{v=v_N(t)} \tag{3.34}$$

$$G(t) \coloneqq \left(\frac{\partial}{\partial x} g(x) \right)_{x=x_N(t)} \tag{3.35}$$

$$B(t) \coloneqq \left(\frac{\partial}{\partial u} F\left(x_N(t), v_N(t), u\right) \right)_{u=0} \tag{3.36}$$

where $\frac{\partial F}{\partial x}$ is the Jacobian in terms of positions and $\frac{\partial F}{\partial v}$ is the Jacobian in terms of velocities.

By merging a lot of the excess notation, this can be written more simply as follows.

$$\dot{x} = v \tag{3.37}$$

$$M\dot{v} = -Kx - Dv - G^T\lambda + Bu \tag{3.38}$$

$$Gx = z \tag{3.39}$$

3.4.6 Solving Differential Algebraic Equations

By examining the structure of the matrix equation 3.24, some useful properties emerge. Using the bordering method (which is a special case of the usual method for block inversion), then a matrix of the form

$$\begin{bmatrix} S & H^T \\ H & 0 \end{bmatrix} \tag{3.40}$$

has the following inverse, assuming that S is invertible

$$\begin{bmatrix} S & H^T \\ H & 0 \end{bmatrix}^{-1} = \begin{bmatrix} S^{-1} - S^{-1}H^T V H S^{-1} & S^{-1}H^T V \\ VHS^{-1} & -V \end{bmatrix} \tag{3.41}$$

with

$$V = \left(HS^{-1}H^T\right)^{-1}$$

Therefore the inversion of the left-hand matrix in equation 3.24 leads to the following solution:

$$\begin{bmatrix} \dot{v} \\ \lambda \end{bmatrix} = \begin{bmatrix} M^{-1} - M^{-1}G^T VGM^{-1} & M^{-1}G^T V \\ VGM^{-1} & -V \end{bmatrix} \begin{bmatrix} F \\ g \end{bmatrix}$$

with

$$V = \left(GM^{-1}G^T\right)^{-1}$$

Alternatively, the mass matrix can be kept on the left-hand side as a factor of the velocity term. This keeps the formulation in the form of $forces = mass \times acceleration$, or $F = ma$. This leads to the following solution formulation.

$$\begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \dot{v} \\ \lambda \end{bmatrix} = \begin{bmatrix} I - M^{-1}G^T VGM^{-1} & G^T V \\ VGM^{-1} & -V \end{bmatrix} \begin{bmatrix} F \\ g \end{bmatrix} \tag{3.42}$$

with

$$V = \left(GM^{-1}G^T\right)^{-1}$$

DAE Projection Method

This is one simple method for solving the DAE system of equations. It involves solving the the system as an ODE on the acceleration level and then projecting the solution onto the position and velocity constraint manifold. More sophisticated methods are presented in the following sections.

At this point it is useful to define the following matrices that are constant for a given state.

$$A = I - H^T \left(GM^{-1}G^T\right)^{-1} GM^{-1}$$
$$B = G^T \left(GM^{-1}G^T\right)^{-1}$$

Therefore the new formulation of the governing ODE (which has the constraints on the acceleration level) to solve is

$$\begin{bmatrix} I & 0 \\ 0 & M \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v \\ AF\left(x, v\right) + B\ddot{g}\left(x, v\right) \end{bmatrix}$$

which can be solved using a simple backward Euler method

$$\begin{bmatrix} I & 0 \\ 0 & M \end{bmatrix} \begin{bmatrix} x(t+dt) - x(t) \\ v(t+dt) - v(t) \end{bmatrix}$$
$$= h \begin{bmatrix} v(t+dt) \\ AF\left(x(t+dt), v(t+dt)\right) + B\ddot{g}\left(x(t+dt), v(t+dt)\right) \end{bmatrix}$$

where dt is the size of the time step. For the full system, this is equivalent to

$$\bar{M} \cdot (Y(t+dt) - Y(t)) = hF(Y(t+dt))$$

This is a non-linear equation in $Y(t+dt)$ which is solved using a Newton-Raphson method, taking the following form.

$$F_{\text{NR}}(y) = M \cdot (y - Y(t)) - hF(y) = 0$$

This is done by calculating a series of y_k with $y_0 = Y(t)$, as shown below. Note that y_0 is the solution to the system if and only if there are no forces acting in the model, therefore there is no motion and the state need not be updated.

$$F_{\text{NR}}(y_k + \Delta y_k) = F_{\text{NR}}(y_k) + \frac{dF_{\text{NR}}}{dy}(y_k)\,\Delta y_k = 0$$
$$\Delta y_k = -\left(\frac{dF_{\text{NR}}}{dy}(y_k)\right)^{-1} F_{\text{NR}}(y_k)$$

with

$$F_{\text{NR}}(y_k) = M \cdot (y_k + Y_k) - hF(y_k)$$
$$\frac{dF_{\text{NR}}}{dy}(y_k) = M \cdot (y_k) - h\frac{dF}{dy}(y_k)$$

Next, the solution is projected onto the solution space for the position and velocity constraints, as described by Cline and Pai [60]. This is the "post-stabilisation" step. The solution has been solved on the acceleration-level (index-1), so it still requires correction on the position and velocity levels due to numerical integration being used to get these values. The system to solve after integration is as follows.

$$\begin{bmatrix} M & G^T \\ G & 0 \end{bmatrix} \begin{bmatrix} \dot{v} \\ \lambda \end{bmatrix} = \begin{bmatrix} F_v \\ \ddot{g} \end{bmatrix}$$

which leads to

$$\Delta Y = M^{-1}G^T \left(GM^{-1}G^T\right)^{-1} (-g(Y))$$

can be used to update the state using

$$Y_{k+1} = Y_k + h \cdot \Delta Y_{k+1}$$

Now as many constraints are dependent on the interaction of two bodies (with mass matrices M_0 and M_1 respectively) and with two types of constraints

(one on position, $g(x)$, and one on velocity $h(v)$), the system to solve after integration can be written in the following form.

$$\begin{bmatrix} I & 0 & 0 & 0 & G_{0x}^T & H_{0x}^T \\ 0 & M_0 & 0 & 0 & 0 & H_{0v}^T \\ 0 & 0 & I & 0 & G_{1x}^T & H_{1x}^T \\ 0 & 0 & 0 & M_1 & 0 & H_{1v}^T \\ G_{0x} & 0 & G_{1x} & 0 & 0 & 0 \\ H_{0x} & H_{0v} & H_{1x} & H_{1v} & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_0 \\ \Delta v_0 \\ \Delta x_1 \\ \Delta v_1 \\ -\lambda_X \\ -\lambda_V \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -g\left(x_0, x_1\right) \\ -h\left(x_0, v_0, x_1, v_1\right) \end{bmatrix}$$

For a two-body system, this is obtained using the following.

$$\begin{aligned} M^{-1} &= \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & M_0^{-1} & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & M_1^{-1} \end{bmatrix} \\ G &= \begin{bmatrix} G_{0x} & 0 & G_{1x} & 0 \\ H_{0x} & H_{0v} & H_{1x} & H_{1v} \end{bmatrix} \\ G^T &= \begin{bmatrix} G_{0x}^T & H_{0x}^T \\ 0 & H_{0v}^T \\ G_{1x}^T & H_{1x}^T \\ 0 & H_{1v}^T \end{bmatrix} \\ M^{-1}G^T &= \begin{bmatrix} G_{0x}^T & H_{0x}^T \\ 0 & M_0^{-1}H_{1v}^T \\ G_{1x}^T & H_{1x}^T \\ 0 & M_1^{-1}H_{1v}^T \end{bmatrix} \\ GM^{-1}G^T &= \begin{bmatrix} G_{0x}G_{0x}^T + G_{1x}G_{1x}^T & G_{0x}H_{0x}^T + G_{1x}H_{1x}^T \\ H_{0x}G_{0x}^T + H_{1x}G_{1x}^T & H_{0x}H_{0x}^T + H_{0v}M_0^{-1}H_{0v}^T \\ & + H_{1x}H_{1x}^T + H_{1v}M_1^{-1}H_{1v}^T \end{bmatrix} \end{aligned}$$

DAE Solver: BDF1, index1, without lambda variable

A more sophisticated method would be to solve the constraint's second derivative at the same time as the ODE, i.e. $\ddot{g}$. Note that after differentiating the constraints twice the constraint can be separated and expressed similarly to the method for equation 3.21.

$$\ddot{g} = A\dot{v}_0 + B\dot{v}_1 - \xi\left(x_0, v_0, x_1, v_1\right) = 0 \tag{3.43}$$

This leads to the following extended formulation for multi-body systems.

$$\begin{bmatrix} I & 0 & 0 & 0 & 0 \\ 0 & M_0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & M_1 & 0 \\ 0 & A & 0 & B & 0 \end{bmatrix} \begin{bmatrix} \dot{x}_0 \\ \dot{v}_0 \\ \dot{x}_1 \\ \dot{v}_1 \\ \dot{\lambda} \end{bmatrix}$$

$$= \begin{bmatrix} F_{x_0} \\ F_{v_0} - \left(\frac{dF_{x_0}}{dv}\right)^T G_{x_0}^T \lambda \\ F_{x_1} \\ F_{v_1} - \left(\frac{dF_{x_1}}{dv}\right)^T G_{x_1}^T \lambda \\ \xi\left(x_0, v_0, x_1, v_1\right) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{dF_{x_0}}{dx_0} & \frac{dF_{x_0}}{dv_0} & 0 & 0 & 0 \\ \frac{dF_{v_0}}{dx_0} & \frac{dF_{v_0}}{dv_0} & 0 & 0 & -\left(\frac{dF_{x_0}}{dv}\right)^T G_{x_0}^T \\ 0 & 0 & \frac{dF_{x_1}}{dx_1} & \frac{dF_{x_1}}{dv_1} & 0 \\ 0 & 0 & \frac{dF_{v_1}}{dx_1} & \frac{dF_{v_1}}{dv_1} & -\left(\frac{dF_{x_1}}{dv}\right)^T \\ \frac{d\xi}{dx_0} & \frac{d\xi}{dv_0} & \frac{d\xi}{dx_1} & \frac{d\xi}{dv_1} & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ v_0 \\ x_1 \\ v_1 \\ \lambda \end{bmatrix}$$

which can be described as an implicit DAE in the form of

$$F\left(y, \dot{y}\right) = 0$$

This method is an attempt to solve the implicit DAE by first embedding the constraint into the acceleration, which leads to the following ODE.

$$\begin{bmatrix} M_0 & 0 \\ 0 & M_1 \\ A & B \end{bmatrix} \begin{bmatrix} \dot{v}_0 \\ \dot{v}_1 \end{bmatrix} = \begin{bmatrix} F_{v_0} - \left(\frac{dF_{x_0}}{dv}\right)^T G_{x_0}^T \lambda \\ F_{v_1} - \left(\frac{dF_{x_1}}{dv}\right)^T G_{x_1}^T \lambda \\ \xi\left(x_0, v_0, x_1, v_1\right) \end{bmatrix} \tag{3.44}$$

$$= \begin{bmatrix} \frac{dF_{v_0}}{dx_0} & \frac{dF_{v_0}}{dv_0} & 0 & 0 & -\left(\frac{dF_{x_0}}{dv}\right)^T G_{x_0}^T \\ 0 & 0 & \frac{dF_{v_1}}{dx_1} & \frac{dF_{v_1}}{dv_1} & -\left(\frac{dF_{x_1}}{dv}\right)^T G_{x_1}^T \\ \frac{d\xi}{dx_0} & \frac{d\xi}{dv_0} & \frac{d\xi}{dx_1} & \frac{d\xi}{dv_1} & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ v_0 \\ x_1 \\ v_1 \\ \lambda \end{bmatrix} \tag{3.45}$$

Equation 3.45 can alternatively be reduced to be in terms of $\dot{v}_0$, $\dot{v}_1$ and λ, as follows.

$$\begin{bmatrix} M_0 & 0 & \left(\frac{dF_{x_0}}{dv}\right)^T G_{0_x}^T \\ 0 & M_1 & \left(\frac{dF_{x_1}}{dv}\right)^T G_{1_x}^T \\ G_{0_x}\frac{dF_{x_0}}{dv} & G_{1_x}\frac{dF_{x_1}}{dv} & 0 \end{bmatrix} \begin{bmatrix} \dot{v}_0 \\ \dot{v}_1 \\ \lambda \end{bmatrix} = \begin{bmatrix} F_{v_0} \\ F_{v_1} \\ \xi \end{bmatrix} \tag{3.46}$$

As the matrix here is of the same from as 3.40, then its inverse has the form of 3.41. Therefore the solution to this linear system is as follows. Note that the mass matrices have been kept on the left hand side of the equation, as with equation 3.42, and only the first two rows are computed (the upper block), as lambda is not used here.

$$\begin{bmatrix} M_0 & 0 \\ 0 & M_1 \end{bmatrix} \begin{bmatrix} \dot{v}_0 \\ \dot{v}_1 \end{bmatrix} = \left[I - H^T\left(HS^-1H^T\right)^{-1} HS^{-1}\right] \begin{bmatrix} F_{v_0} \\ F_{v_1} \end{bmatrix} + \left[S^{-1}H^T\left(HS^{-1}H^T\right)^{-1}\right] \left[\xi\right] \tag{3.47}$$

where

$$S = \begin{bmatrix} M_0 & 0 \\ 0 & M_1 \end{bmatrix}$$
$$H = \begin{bmatrix} G_{0_x}\frac{dF_{x_0}}{dv} & G_{1_x}\frac{dF_{x_1}}{dv} \end{bmatrix}$$

In order to simplify notation, here the following matrices are defined in a manner that supersedes equation 3.43.

$$\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} = A = \left[I - H^T\left(HS^-1H^T\right)^{-1} HS^{-1}\right]$$
$$\begin{bmatrix} B_0 & B_1 \end{bmatrix} = B = \left[S^{-1}H^T\left(HS^{-1}H^T\right)^{-1}\right]$$

Note that A is a 2×2 block matrix, and that B is a 1×2 block matrix since S is a 2×2 block matrix, and that H is a 1×2 block matrix. So by taking these definitions, and permuting the rows such that the positions are together and the velocities are together, this leads to the following system.

$$\begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & M_0 & 0 \\ 0 & 0 & 0 & M_1 \end{bmatrix} \begin{bmatrix} \dot{x}_0 \\ \dot{x}_1 \\ \dot{v}_0 \\ \dot{v}_1 \end{bmatrix} = \begin{bmatrix} F_{x_0} \\ F_{x_1} \\ A_{00}F_{v_0} + A_{01}F_{v_1} + B_0\xi \\ A_{10}F_{v_0} + A_{11}F_{v_1} + B_1\xi \end{bmatrix}$$

This ODE is then solved using Backward Euler.

$$\begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & M_0 & 0 \\ 0 & 0 & 0 & M_1 \end{bmatrix} \begin{bmatrix} \dot{x}_0(t+dt) - \dot{x}_0(t) \\ \dot{x}_1(t+dt) - \dot{x}_1(t) \\ \dot{v}_0(t+dt) - \dot{v}_0(t) \\ \dot{v}_1(t+dt) - \dot{v}_1(t) \end{bmatrix}$$

$$= h \begin{bmatrix} F_{x_0}(x_0(t+dt), v_0(t+dt)) \\ F_{x_1}(x_1(t+dt), v_1(t+dt)) \\ \begin{array}{l} A_{00}F_{v_0}(x_0(t+dt), v_0(t+dt)) \\ + A_{01}F_{v_1}(x_1(t+dt), v_1(t+dt)) \\ + B_0\xi(x_0(t+dt), v_0(t+dt), x_1(t+dt), v_1(t+dt)) \end{array} \\ \begin{array}{l} A_{10}F_{v_0}(x_0(t+dt), v_0(t+dt)) \\ + A_{11}F_{v_1}(x_1(t+dt), v_1(t+dt)) \\ + B_1\xi(x_0(t+dt), v_0(t+dt), x_1(t+dt), v_1(t+dt)) \end{array} \end{bmatrix}$$

This is a nonlinear problem, however the problem is linearised by using

successive linearisation in each step, as follows.

$$\begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & M_0 & 0 \\ 0 & 0 & 0 & M_1 \end{bmatrix} \begin{bmatrix} \dot{x}_0(t+dt) - \dot{x}_0(t) \\ \dot{x}_1(t+dt) - \dot{x}_1(t) \\ \dot{v}_0(t+dt) - \dot{v}_0(t) \\ \dot{v}_1(t+dt) - \dot{v}_1(t) \end{bmatrix}$$

$$= h \begin{bmatrix} F_{x_0}(x_0(t), v_0(t)) + \frac{dF_{x_0}}{dx_0}\Delta x_0 + \frac{dF_{x_0}}{dv_0}\Delta v_0 \\ F_{x_1}(x_1(t), v_1(t)) + \frac{dF_{x_1}}{dx_1}\Delta x_1 + \frac{dF_{x_1}}{dv_1}\Delta v_1 \\ A_{00}\left[F_{v_0}(x_0(t), v_0(t)) + \frac{dF_{x_0}}{dx_0}\Delta x_0 + \frac{dF_{x_0}}{dv_0}\Delta v_0\right] \\ + A_{01}\left[F_{v_1}(x_1(t), v_1(t)) + \frac{dF_{x_1}}{dx_1}\Delta x_1 + \frac{dF_{x_1}}{dv_1}\Delta v_1\right] \\ + B_0\xi\left[\begin{array}{l}(x_0(t), v_0(t), x_1(t), v_1(t)) \\ + \frac{d\xi}{dx_0}\Delta x_0 + \frac{d\xi}{dv_0}\Delta v_0 + \frac{d\xi}{dx_1}\Delta x_1 + \frac{d\xi}{dv_1}\Delta v_1\end{array}\right] \\ A_{10}\left[F_{v_0}(x_0(t), v_0(t)) + \frac{dF_{x_0}}{dx_0}\Delta x_0 + \frac{dF_{x_0}}{dv_0}\Delta v_0\right] \\ + A_{11}\left[F_{v_1}(x_1(t), v_1(t)) + \frac{dF_{x_1}}{dx_1}\Delta x_1 + \frac{dF_{x_1}}{dv_1}\Delta v_1\right] \\ + B_1\xi\left[\begin{array}{l}(x_0(t), v_0(t), x_1(t), v_1(t)) \\ + \frac{d\xi}{dx_0}\Delta x_0 + \frac{d\xi}{dv_0}\Delta v_0 + \frac{d\xi}{dx_1}\Delta x_1 + \frac{d\xi}{dv_1}\Delta v_1\end{array}\right] \end{bmatrix}$$

$$\begin{bmatrix} I - h\frac{dF_{x_0}}{dx_0} & 0 & -h\frac{dF_{x_0}}{dv_0} & 0 \\ 0 & I - h\frac{dF_{x_1}}{dx_1} & 0 & -h\frac{dF_{x_1}}{dv_1} \\ -hX_{(000)} & -hX_{(010)} & M_0 - hX_{(000)} & -hX_{(010)} \\ -hX_{(101)} & -hX_{(111)} & -hX_{(101)} & M_1 - hX_{(111)} \end{bmatrix} \begin{bmatrix} \Delta x_0 \\ \Delta x_1 \\ \Delta v_0 \\ \Delta v_1 \end{bmatrix}$$

$$= h \begin{bmatrix} Fx_0\left(x_0(t), v_0(t)\right) \\ \\ Fx_1\left(x_1(t), v_1(t)\right) \\ \\ A_{00}Fv_0\left(x_0(t), v_0(t)\right) + A_{01}Fv_1\left(x_1(t), v_1(t)\right) \\ +B_0\xi\left(x_0(t), v_0(t), x_1(t), v_1(t)\right) \\ \\ A_{10}Fv_0\left(x_0(t), v_0(t)\right) + A_{11}Fv_1\left(x_1(t), v_1(t)\right) \\ +B_1\xi\left(x_0(t), v_0(t), x_1(t), v_1(t)\right) \end{bmatrix}$$

where

$$X_{(ijk)} = A_{ij}\frac{dF_{x_j}}{dx_j} + B_k\frac{d\xi}{dx_j}$$

which is

$$\frac{M \cdot \Delta Y}{h} = F(Y) + \frac{dF}{dY} \cdot \Delta Y$$
$$\Leftrightarrow \left[M - h\frac{dF}{dY}\right] = h \cdot F(Y)$$

DAE Solver: BDF1, index1, with lambda variable

If instead of taking equation 3.44 and reducing it to its top block (as done in equation 3.47), it is kept in full, in terms of $\dot{x}_0$, $\dot{x}_1$, $\dot{v}_0$, $\dot{v}_1$ and $\dot{\lambda}$ then this can be expressed as implicit DAE of the form

$$F_{\text{NR}}\left(Y, \dot{Y}\right) = 0$$

or as a semi-explicit DAE

$$F_{\text{NR}}\left(Y, \dot{Y}\right) = M\left(Y\right) \cdot \dot{Y} - F\left(Y\right) = 0$$

The BDF of order 1 is a Backward Euler method which solves the following

non-linear problem.

$$F_{\text{NR}}\left(Y, \frac{Y - Y_i}{h}\right) = M\left(Y\right) \cdot \frac{\Delta Y}{h} - F\left(Y\right) = 0$$
$$\frac{dF_{\text{NR}}}{dY} = \frac{M}{h} - \frac{dF}{dY_n}$$

The non-linear solver will iteratively solve the following in each step.

$$\begin{aligned} & F_{\text{NR}}\left(Y_n\right) = 0 \\ \Leftrightarrow\ & F_{\text{NR}}\left(Y_{n-1}\right) \cdot \Delta Y = 0 \\ \Leftrightarrow\ & \frac{dF_{\text{NR}}}{dY} \cdot \Delta Y = -F_{\text{NR}}\left(Y_{n-1}\right) \end{aligned}$$

So the update to the state in each step for the implicit Euler can be computed using

$$\Delta Y = -\left(M - dt \cdot \frac{dF}{dY}\right)^{-1} \left(M \cdot \left(Y_k - Y_0\right) - dt \cdot F\right)$$

3.4.7 Collision Response

In order to solve collision responses and ensure contacts are correctly handled, a Linear Complementarity Problem (LCP) is solved [61]. This is a method of solving problems with complementarity conditions. That is, problems of the following form: given $M \in \mathbb{R}^{n \times n}$ and $q \in \mathbb{R}^n$, find $w = w_j \in \mathbb{R}^n$ and $z = z_j \in \mathbb{R}^n$ satisfying

$$w - Mz = q \tag{3.48}$$
$$w, z \geq 0 \tag{3.49}$$
$$w^T z = 0 \tag{3.50}$$

Equation 3.50 gives the name to this class of problem, and means that at least one of the pair w_j, z_j must be equal to zero. This very much applies to the case of collisions, as when there is no collision the distance between two objects is greater than zero, the collision response force force must be zero. Furthermore when there is a collision the distance between the two objects is zero and the collision response force must be positive.

The LCP is obtained after the linearisation of the model on either one or both of the position level and the velocity level. The problem takes the follow-

ing form.

$$\begin{aligned} x \perp Ax + b \\ x \geq 0 \\ Ax + b \geq 0 \end{aligned}$$

where $A = HM^{-1}H^T$, where M is the system-level mass matrix, containing the mass matrices of all bodies in the system, and H is determined by the contact between respective bodies.

For the position level the contact constraint is

$$(x_0 - x_1) \cdot n \geq 0$$

where x_0 and x_1 are the positions of bodies 0 and 1, and n is the contact normal for the two bodies. Similarly, for the velocity level the contact constraint is

$$(v_0 - v_1) \cdot n \geq 0$$

where v_0 and v_1 are the velocities of bodies 0 and 1.

3.5 Procedure for Analysing

In each time step there may be several iterations of solving the complete system. This is both in the DAE state and constraint solver and in the collision response solver. These calculations are terminated once convergence has been achieved, or once the maximum number of allowed iterations has been reached.

Due to the use of implicit Euler integration, the following approach was used to solve for the error in each iteration.

$$\varepsilon = M(Y_k - Y_0) - hF$$

where ε represents the error for the whole system. This error can be divided into the error on the ODE level, $\varepsilon_{\text{system}}$, and the error on the algebraic constraint level, $\varepsilon_{\text{constraint}}$. This is done by simply extracting the values of ε that refers to each of the two.

As the solver is working with the index-1 formulation of the problem, the constraint violation can also be evaluated on the acceleration level. This is done by evaluating $\ddot{g}$ with the current state.

$$\varepsilon_{\text{violation}} = \ddot{g}\,(Y_k)$$

The model is said to have converged for the time step when the norm of each of these errors have fallen below pre-defined thresholds.

The collision response solver is set up to terminate either once all collisions have been solved (that is, the model had been updated to ensure there are no collision violations) or once a certain number of iterations has been reached.

3.6 Evaluation

In order to evaluate the success or otherwise of the method, several key metrics were investigated by building different test cases.

The main test is to investigate whether this is a physically accurate simulation therefore a comparison to real-world behaviour would validate the design.

Secondary metrics include how generalisable the implementation is, and how computationally efficient the solver method is. Generalisability is a useful goal in programming as it allows for further development to progress much more easily without much restructuring. Eventually this model is expected to run in real-time on surgical simulator equipment. Therefore, another goal is to have a simulation that can solve a standard system on the scale of hundredths of a second. This goal is also secondary to the main goal of building a physically and mathematically accurate simulation as this is the first iteration of this work.

Chapter 4

Implementation

4.1 SGPhysics

`SGPhysics` is a brand new physics engine developed internally within SenseGraphics in order to meet the back-end needs of the medical simulators without relying on tools developed externally. Its development coincided with this thesis project, so both projects were worked on in parallel, complementing each other. This approach allowed for flexibility on both sides to experiment with different approaches for particular problems, and to allow for points of comparison. It also allowed for a more structured and formal organisation of the different parts of each project, which is often not the case when using commercial physics engines.

4.2 Programmatic Implementation

There were many aspects of the implementation that were worked on over the course of this project. These can be grouped into several key areas which are described below.

- Solvers: methods of formulating the system and calculating its solution
- Models: methods of defining the objects in the simulation
- Constraints: methods of defining the constraints in the simulation
- Runtime: the organisation of the software as a whole

For the implementation part of this project, several tools and languages were used. Although there were few restrictions on what could be used, the

convention was to continue using the tools that were already being used by the SenseGraphics team.

- C++ programming language: a language that is widely used in the field of numerical computing as it can be extremely fast while offering many features and tools through libraries or otherwise.
- Python programming language: this language was used while experimenting with DAE solver methods due to its flexibility as a higher-level language than C++.
- Eigen library: a C++ library for linear algebra, matrix and vector operations and numerical solvers. It was relied on for implementing the mathematical processes outlined in Section 3.
- Visual Studio 2017 using Microsoft Visual C++ compiler (MSVC): this IDE was used to assist in writing the code. By default MSVC is used to compile the code.
- OpenGL: this API was used to visualise the output of the simulations and to build a GUI to interact with the model.
- Apache Subversion (SVN): this revision control system was used to manage code updates between all members of the team.
- CMake: this tool was used to build the software independently of the compiler for use after changes to the code.
- LaTeX: used for writing reports and creating presentations.

4.2.1 System and Solvers

There are several variations in different implementations of the DAE solver. Therefore a generic approach is used such that the user can pick how the system should be built and solved. These include formulating the system with a differentiation index of 1, 2 or 3. Furthermore certain functions related to the constraints, such as λ, can appear on the left or right hand side of the equation. This flexible approach is so that they can be compared to test the theory.

The generic implementation meant that there was flexibility in the choice of method, even for different constraints within the same model. It also meant that the different variations could be easily compared.

The system is then solved using the Implicit Euler method described in Section 3.4.6.

4.2.2 Models

As before, a generic class is used to define the bodies. The bodies themselves are defined in different ways, as described in Section 3.2.

The DoF (position and velocity) information for all bodies is stored in a generalised state vector, Y. Each body contains IDs and size information that allows very easy access to its DoF information from Y. This makes it both simple to understand the current state of a given body.

Each body also has all the necessary information to build its mass matrix, M and the force vector, F, for a given state, Y. Furthermore the Jacobian matrices $\frac{\partial F}{\partial x}$ and $\frac{\partial F}{\partial v}$ are defined for each body since they appear in the formulation of the problem described by equation 3.29 to equation 3.36.

4.2.3 Constraints

There were many different types of constraints that were designed and built, as described in Section 3.3. The implementation style of the constraints also took several iterations of development. These different implementations ranged in their level of flexibility, generalisability and complexity.

The final implementation was to define constraints described in Section 3.3 in terms of "part constraints", which are different depending on the body. For example, the fixed points constraint requires two points as inputs. Therefore the "part constraints" would be the way of defining this point for the particular body. The point on a suture, for instance, could be defined from its parametric abscissa, or the point on a section of tissue could either be defined from a point ID for a node or from barycentric coordinates to define an arbitrary point on the surface.

For each constraint, several objects must be defined so that they can be used by different solver methods. Only some are used by each method. The main definitions are as follows:

$$
\begin{array}{rrr}
 & & g, \dot{g}, \ddot{g}, \\
 & & \dfrac{\partial g}{\partial x}, \\
 & \dfrac{\partial \dot{g}}{\partial x}, & \dfrac{\partial \dot{g}}{\partial \dot{x}}, \\
\dfrac{\partial \ddot{g}}{\partial x}, & \dfrac{\partial \ddot{g}}{\partial \dot{x}}, & \dfrac{\partial \ddot{g}}{\partial \ddot{x}}, \\
 & & \xi
\end{array}
$$

where g is the constraint formulated on the position level, and ξ is as defined in equation 3.20.

Another useful definitions is

$$\frac{\partial \frac{\partial g}{\partial x}^T}{\partial x} \cdot u$$

which is a tensor-vector product, for a given vector multiplier, u, and tensor, $\frac{\partial \frac{\partial g}{\partial x}^T}{\partial x}$. In order to define this more easily, tensor theory can be used to decompose the tensor-vector product and then rebuild it.

Note that a matrix-vector product can be written using Einstein notation as

$$A \cdot v \coloneqq [A]_{ab} : [v]_b \tag{4.1}$$

where A is a matrix with dimensions in a and b, while v is a vector in dimension b. This results in a matrix described along dimension a. For example, if matrix A were of size 3×7, and vector v were of size 7×1, then the value 3 represents the size of A along dimension a, and the value 7 represents the size of both A and v along dimension b. Since both A and v are of the same size in dimension b, the dot product can take place and the resulting vector will be of size 3×1 as this is the size of A along dimension a.

Starting with the definition of the constraint, g, in dimension a

$$[g]_a$$

and its derivatives with their new dimensions

$$\left[\frac{\partial g}{\partial x}\right]_{ab}$$
$$\left[\frac{\partial g}{\partial x}^T\right]_{ab} = \left[\frac{\partial g}{\partial x}\right]_{ba}$$
$$\left[\frac{\partial \frac{\partial g}{\partial x}^T}{\partial x}\right]_{bac}$$

The target to reach is the following, but the dimension in which λ is defined is not known at this point.

$$\left[\frac{\partial \frac{\partial g}{\partial x}^T}{\partial x}\right]_{bac} : [\lambda]_?$$

however from the matrix-vector product, defined by equation 4.1, it can be said that

$$\left[\frac{\partial g}{\partial x}^T\right]_{ab} : [\lambda]_a$$

therefore

$$\left[\frac{\partial \frac{\partial g}{\partial x}^T}{\partial x}\right]_{bac} : [\lambda]_a$$

Now that the correct dimension has been determined, its size must also be determined. This can be derived from the tensor. The dimension of all DoF of the model is of size n, and the size of all the model's constraints is c.

$$\begin{aligned} g &\rightarrow \text{size } c \\ \frac{\partial g}{\partial x} &\rightarrow \text{size } c \times n \\ \frac{\partial g}{\partial x}^T &\rightarrow \text{size } n \times c \\ \frac{\partial g}{\partial x}^T : \lambda &\rightarrow \text{size } n \end{aligned}$$

so λ *must* be of size c.

Therefore

$$\begin{aligned} \left[\frac{\partial \frac{\partial g}{\partial x}^T}{\partial x}\right] &\rightarrow \text{size } n \times c \times n \\ \left[\frac{\partial \frac{\partial g}{\partial x}^T}{\partial x}\right]_{bac} : [\lambda]_a &\rightarrow \text{size } n \times n \end{aligned}$$

So in order to build the matrix resulting from this tensor-vector product, the following should be described analytically

$$\frac{\partial g}{\partial x}^T \lambda$$

then

$$\frac{\partial \frac{\partial g}{\partial x}^T \lambda}{\partial x_i}$$

are the rows of the matrix, with x_i as the degree of freedom for that row.

Once all of these objects have been defined for each of the constraints then they can be used by different solver methods.

4.2.4 Runtime

The execution of the simulation takes place in several stages. First the model must be set up, and all its contents defined. The first step is to instantiate all bodies in the model with all of their physical properties and other information such as colour, gravity and damping coefficients. Next the required "part constraints" are defined before defining the full constraints. Finally the initial state, Y_0, is defined for the model. This gives the initial position and velocity for all bodies in the system.

Once the initialisation method has been called, the collision meshes are updated. This makes it simpler to access the information required to assess collisions and collision responses. Next the "advance scene" loop is started with dt as the time step. This loop starts by running the chosen solver method. Once the method has run, using the required members of the bodies and constraints, the state is updated. Next a post stabilisation step occurs to negate drift-off, and the collision response method is called to resolve any undesired intersections. Note that any constraints that are created on-the-fly, such as the suturing constraint, are added as required.

Visualisation of the system is handled by OpenGL, which can also be used as an interface to interact with the model by moving parts of the model around. OpenGL provides a framework to visualise the model and act as a GUI.

4.3 Creation of Examples

Many scenarios were defined and analysed. They ranged from simple models for validation purposes, to more complex suturing models. Examples for these are shown in Chapter 5, Results.

Chapter 5

Results

5.1 Kinematic Analysis

In order to test if the models were reasonable, many were created with different configurations and constraints. These ranged from simple models such as structures built from connected rigid bodies, to more complex tissue and suture structures. Figure 5.1 shows some of these structures.

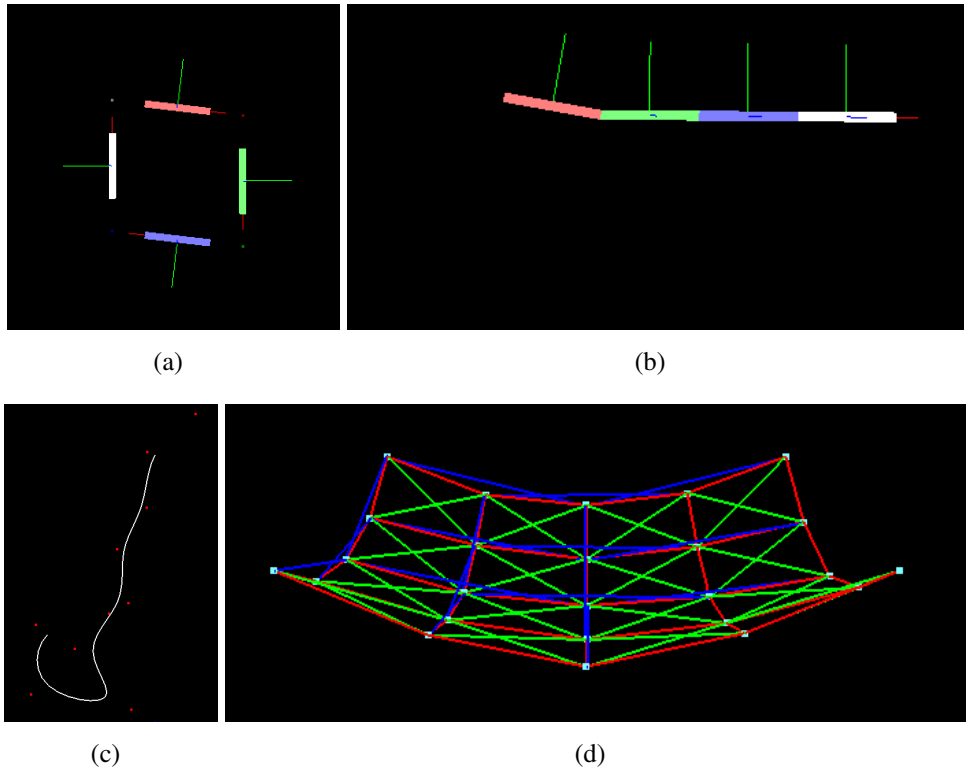
Of course, if the system is ill-defined then this will not lead to a well-defined structure. However it is tested and confirmed that the models tested were correctly defined.

It was found that when the simple models where correctly defined they behaved as expected, remaining correctly constrained and bodies did not penetrate each other when they were not supposed to.

5.2 Dynamic Simulation

In order to test how system behaves with time, forces are defined for the required parts of the model. These include external forces such as gravity. Once again, the behaviour of all of the simple models were validated as they behaved as expected.

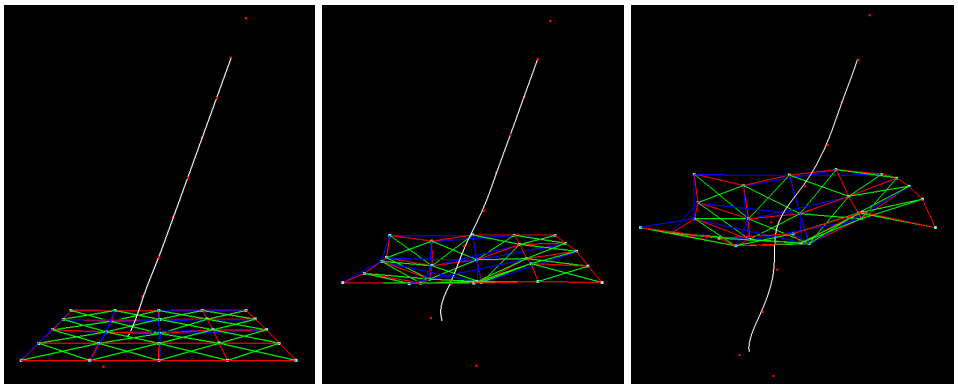
As stated in the method, the dynamic simulation was carried out for the problem formulated with constraints on the position, velocity and acceleration level. The error after using the Newton method to solve the system is shown in Table 5.1. It shows that formulating the problem as an index-1 problem is much more stable, and actually only a small amount of post-stabilisation is required to correct any drift-off.



(a) (b)

(c) (d)

Figure 5.1: Structures built for testing the model. (a) Closed loop of rigid bodies. (b) Quadruple pendulum. (c) Suspended suture. (d) Suspended tissue section.



(a) (b) (c)

Figure 5.2: Suturing simulation of a surgical thread being forced through a point in the tissue.

Table 5.1: Error analysis for system formulated on index-3 (position), index-2 (velocity) and index-1 (acceleration). The systems were solved with the Newton-Raphson non-linear solver, NR, and then projected back onto the constraint manifold using the Lagrange multiplier method in a projection step, PS.

	process [iteration]	$\|g\|$	$\|\dot{g}\|$	$\|\ddot{g}\|$
Index-3				
Step 0	NR [3]	1.388e-17	1.076e-06	3.229e-03
	PS [6]	1.132e-05	1.241e-14	1.251e-11
Step 9	NR [7]	1.593e-10	106.557	160622
	PS [9]	6.020e-02	40.4938	38853.7
Index-2				
Step 0	NR [2]	1.076e-09	1.579e-10	2.152e-03
	PS [6]	3.179e-13	5.085e-05	4.66e-11
Step 9	NR [5]	1.400e-02	6.708e-12	71088.2
	PS [9]	1.119e-14	16.4895	9.535e-10
Index-1				
Step 0	NR [4]	3.229e-09	2.152e-06	3.579e-13
	PS [4]	3.197e-11	4.690e-12	1.030e-06
Step 200	NR [4]	7.313e-06	2.025e-04	1.753e-12
	PS [4]	5.207e-13	7.073e-12	3.045e-03

An important example of a dynamic model is shown in Figure 5.2. It shows three snapshots of the suturing process.

The suture behaves as expected, passing through a single point in the tissue. The tissue, on the other hand, has certain instabilities which causes it to deform in unrealistic and nonphysical ways.

5.3 Performance

The emphasis of this project was not on performance, but rather on how the simulations function. It was, however, evaluated along two main metrics: the time taken to update to a new frame, and the rate of convergence in each step. Note that because of the approach taken, any degree of error could be chosen as the desired accuracy.

5.3.1 Refresh Times

The time step was varied and the consequences of this was examined. It was found to have an effect on the speed of the program. Table 5.2 shows the recorded times between frames of the simulation in release mode, which was for convergence to machine precision. It is clear that the addition of the suturing constraint greatly affected the amount of time required between frames. In both cases the time to calculate the state at the next step was greater than the actual size of the time step.

Table 5.2: Mean times between frames of the simulation with standard deviation, σ, for C++ implementation.

dt [s]	no suturing [s]	σ	suturing [s]	σ
0.001	0.107	0.006	0.459	0.045
0.005	0.186	0.030	0.588	0.061
0.010	0.188	0.012	0.596	0.033

5.3.2 Convergence Rates

Figure 5.3 shows the convergence rate of the DAE solver when a suturing constraint was not active, while Figure 5.4 shows the convergence rate of the DAE solver when a suturing constraint was active. The residual and violation are computed from

$$\text{residual} = \|M \cdot (Y_k - Y_0) - dt \cdot F\| \tag{5.1}$$

$$\text{violation} = \|\ddot{g}\,(Y_k)\| \tag{5.2}$$

In both cases convergence was with a rate of approximately ε^n, with n iterations.

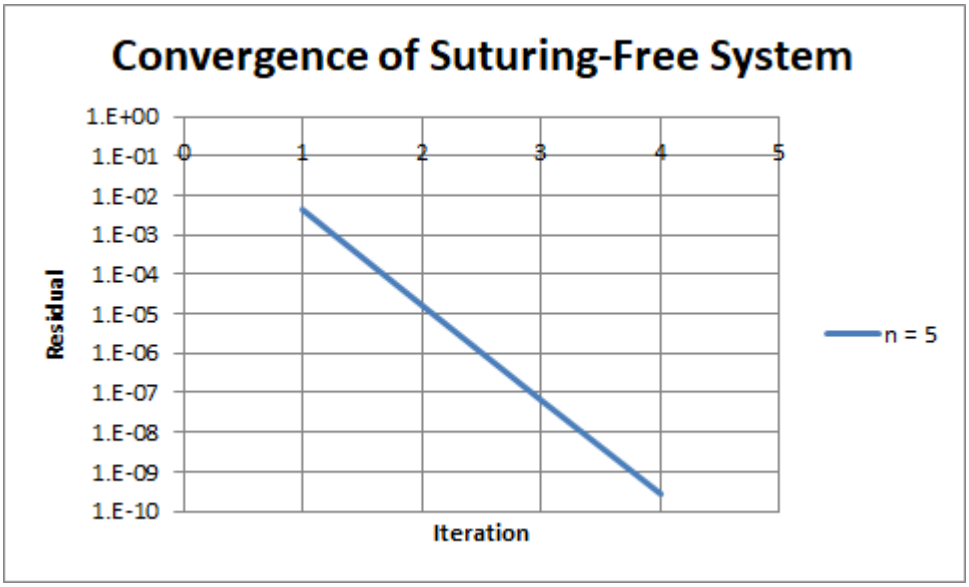


Figure 5.3: Convergence of solving the system in each step when suturing is not occurring for $dt = 0.001$ at step n, where t_n is the current time.

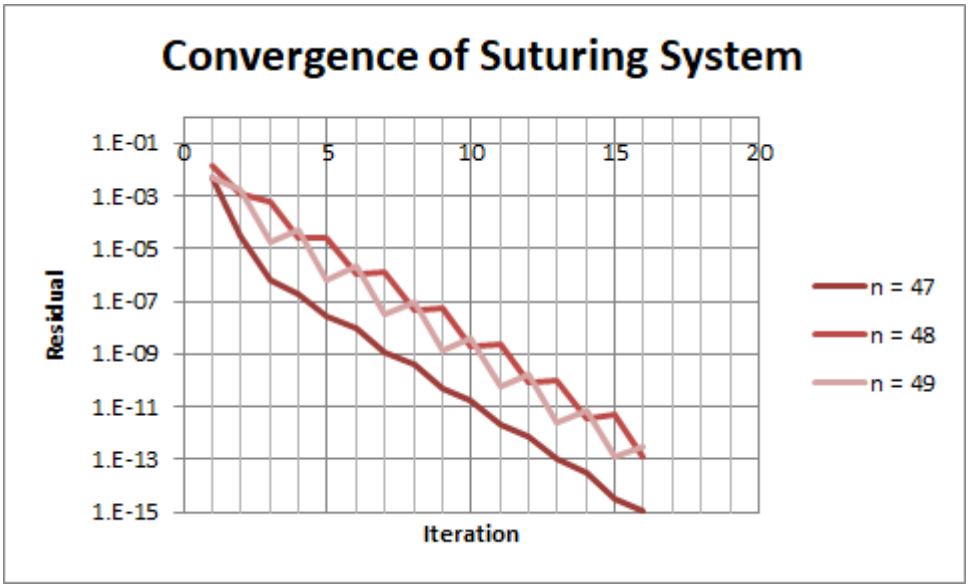


Figure 5.4: Convergence of solving the system in each step when suturing is occurring for $dt = 0.001$ at step n, where t_n is the current time. The different plots show the convergence for different frames in the simulation.

Chapter 6

Discussion

There are many aspects of this project and its outcome that deserve further reflection. These include positive effects and drawbacks, and an overall evaluation on the results of the degree project.

6.1 Existence and Uniqueness of Solutions

The models all provided numerically stable solutions for all of the simple cases shown previously in Figure 5.1. Solutions were found, providing existence and uniqueness of solutions to the system. One of these conditions was the initial condition must meet constraint equations and their time derivatives. This was very important in the setup of the model. If this were not the case then the system would be ill-defined and there would be no desirable solution in existence. So the initial values cannot be chosen arbitrarily, and must meet the conditions. If these conditions are met, then the model is *consistent*.

Let

$$E \coloneqq \begin{bmatrix} I & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{6.1}$$

$$A \coloneqq \begin{bmatrix} 0 & I & 0 \\ -K & -D & -G^T \\ G & 0 & 0 \end{bmatrix} \tag{6.2}$$

where E and A are square matricies. Then the system described by equations 3.37 to 3.39 can be described as a linear constant coefficient DAE as

follows.

$$E\dot{x}(t) = Ax(t) + f(t) \tag{6.3}$$
$$x(t_0) = 0 \tag{6.4}$$

The system can now be solved by model transformation (i.e. computing eigenvalues and eigenvector pairs).

6.1.1 Case 1: E nonsingular

This means that E is invertible, and the following can be computed. Let

$$\tilde{A} = E^{-1}A \tag{6.5}$$

then the system can be expressed as

$$\dot{x} = \tilde{A}x + Bu \tag{6.6}$$
$$x(t_0) = 0 \tag{6.7}$$

which has the solution

$$x(t) = e^{\tilde{A}(t-t_0)}x_0 + \int_{t_0}^{t} e^{\tilde{A}(s-t)}Bu(s)ds \tag{6.8}$$

where the matrix exponential function is defined in terms of the infinite series

$$e^{At} := I + \sum_{i=1}^{\infty} \frac{(At)^i}{i!} \tag{6.9}$$

6.1.2 Case 2: E singular, A nonsingular

Here the solution can be transformed as follows.

$$A^{-1}E\dot{x}(t) = x(t) + A^{-1}f(t) \tag{6.10}$$

Let J be the Jordan canonical form of the singular matrix $A^{-1}E$

$$A^{-1}E = TJT^{-1} \tag{6.11}$$

with

$$J = \begin{bmatrix} R & 0 \\ 0 & N \end{bmatrix} \tag{6.12}$$

with R representing the Jordan blocks with non-zero eigenvalues, and N the zero eigenvalue blocks. Therefore R is nonsingular and N is nipotent.

Definition 6.1.1. Nilpotency Index: [55] The nilpotency index of the DAE, ν, is defined as the index of nilpotency of N, i.e.

$$\nu = \text{ind}(N) \tag{6.13}$$

This is due to ν being the smallest number with $N^\nu = 0$ (and therefore $N^{\nu-1} \neq 0$) being identical to the size of the largest Jordan block of N.

Using the coordinate transformation $\bar{x} = T^{-1}x$ and premultiplying 6.10 by T^{-1} leads to

$$\begin{bmatrix} R & 0 \\ 0 & N \end{bmatrix} \begin{bmatrix} \dot{\bar{x}}_1 \\ \dot{\bar{x}}_2 \end{bmatrix} = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \end{bmatrix} T^{-1}A^{-1}f = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \end{bmatrix} \bar{f} \tag{6.14}$$

which can be decoupled into

$$R\dot{\bar{x}}_1 = \bar{x}_1\bar{f}_1 \tag{6.15}$$

$$N\dot{\bar{x}}_2 = \bar{x}_2\bar{f}_2 \tag{6.16}$$

By rearranging the system and differentiating equation 6.16, the underlying ordinary differential equation (UODE) can be determined.

$$\dot{\bar{x}}_1 = R^{-1}\left(\bar{f}_1 + \bar{x}_1\right) \tag{6.17}$$

$$\dot{\bar{x}}_2 = -\sum_{i=0}^{\nu-1} N^i \bar{f}_2^{(i+1)} \tag{6.18}$$

Backtransformation of equation 6.18 at t_0 give the conditions required for the initial conditions in order to be called *consistent*.

$$x\left(t_0\right) = T\bar{x}\left(t_0\right) = T\begin{bmatrix} \left(T^{-1}x\left(t_0\right)\right)_1 \\ -\sum_{i=0}^{\nu-1} N^i \bar{f}_2^{(i+1)}\left(t_0\right) \end{bmatrix} \tag{6.19}$$

Note that for this to hold, $\bar{f}_2$ must be differentiable $\nu - 1$ times, which may be different for each component. The space of solutions to the UODE is larger than that of the original DAE due to the loss of information from integration constants.

6.1.3 Case 3: E singular, A singular

The solvability of this case is closely tied to the regularity of the matrix pencil, $(\mu E - A)$, with $\mu \in \mathbb{C}$. This is called a *regular* pencil if $d(\mu) := \det(\mu E - A)$ is not identically zero. The solution can then be computed by solving the Drazin form of the system, or by converting it to state space form as described by Eich-Soellner and Führer [55]. These approaches may lead to a more numerically stable formulation of the problem than the methods investigated here.

6.1.4 Newton's Method

Since the models were usually nonlinear, the Newton method was used. This method is heavily dependent on the solvability of the Jacobian, $\frac{\partial F}{\partial x}$, and ensuring the solution can be linearised in each step. It is precisely the computation of this Jacobian which was the one of most costly parts of each step, however computing the linearisation step is also very costly.

Convergence of Newton's Method is given by the Newton Convergence Theorem [62] [63].

6.2 Suturing Behaviour

As shown in Figure 3.6 the overall behaviour of the suture was as expected. Inspection of the parametric abscissa, u, and its time derivative, $\dot{u}$, showed sensible behaviour. The suture passed through the point on the tissue where the piercing force threshold had been met.

There were however certain instabilities in the tissue, especially as the simulation progressed longer in time. Often the tissue underwent unusual and unexpected deformations. For instance, when the suture was initialised in a vertical position, and passed through orthongonally to the tissue, with a small ϵ as the piercing force threshold, parts of the tissue would deform even without frictional forces. This may be due to the structure of the tissue, which may be resolved by the use of a FEM model instead. A greater damping coefficient may also reduce the likelihood of these instabilities, however consideration must be taken to ensure the material properties still reflect real tissue.

6.3 Definitions of Constraints

Due to the large number of different approaches used to solve the models, many members of the constraint classes had to be created which described the constraint differentiated by different quantities. Depending on the process employed, some of these definitions would be used and not others. The formulation of these members may be difficult for more complex constraints. For instance, a dot product constraint has a single value as the result, however the part constraints are composed of three coordinates that define a vector. The implementation must therefore be flexible enough to handle this type of behaviour.

6.4 System Index

Index-2 and index-1 formulations tend to suffer from drift-off, thus requiring greater stabilisation each step on the position-level, as the position constraint is no longer included. However, these formulations have lower perturbation errors and are easier to solve. Table 5.1 shows how the reduction of the problem from position level constraints to acceleration level constraints makes the problem more similar to an ODE, so that problems can be solved straightforwardly by any ODE time integration method.

There will always be perturbations in the system arising from discretisation and truncation of values, and as shown in Section 3.4.3, formulating the system with lower index also reduces the perturbation index, as this system is a Hessengerg DAE. Therefore the benefits for formulating the system in this form outweigh the costs. Furthermore, the choice of a smaller step size reduces the violation of the constraint on the position level.

6.5 Step Size

The choice of the size of the time step had a great effect on the resulting behaviour of the simulation. This was the main reason an implicit method was employed, however caution was still required in the selection of the time step.

A time step that is too small risks simulation overruns, i.e. when the calculation time for a step forwards takes longer than the step itself. A time step that is too large may lead to lower accuracy and lower robustness, however this can be improved by choice of solver. Only a first-order implicit method was used here. A higher order method would be more robust and allow for larger time steps for the same stability. However, if the time step is too large then the simulation may not capture complexities such as oscillations or fine user input from the haptic devices.

The instabilities in the model may have been overcome with a higher-order method that was more stable for the same step size. This would involve retaining information about the state from previous steps. However, this could be avoided by using Runge-Kutta-type multi-stage methods. Furthermore, the implementation of an adaptive time step would allow for larger time steps when changes in the state are low, while allowing for smaller time steps when higher rates of change are detected. This makes the simulation less expensive to run overall, while maintaining accuracy when required.

6.6 Performance

As Table 5.2 shows, the simulation had relatively slow and sub-real-time performance. Since a direct LU decomposition of a matrix with complete pivoting was used to compute ΔY in each step, parallelisation was not taken advantage of. A method such as conjugate gradient or steepest descent could lead to a solution that is just as accurate (or at least accurate enough) in a much shorter amount of time. Furthermore, mostly dense matrices were used during computations. This is due to there being no single matrix class in Eigen that can handle both dense and sparse matrices. Switching to a method taking advantage of sparse matrix capabilities would not only yield better memory use but also better compute time. The choice of a solver method such as the Modified Newton method could also result in an increase in performance.

The successive linearisation method with LU-decomposition did however provide good convergence in each step, as shown by Figure 5.3 and Figure 5.3. In both cases the error seems to be descending with ε^n, with n iterations.

6.7 Project Scope

The aim of this project was to investigate both suturing and knot tying, however as it went on the focus was mainly on the suturing aspect. This was due to time and resource constraints, however out of the two areas of study suturing provides more areas of investigation from the perspective of physics simulation. Once the suture is in a knot-like configuration, a toolbox for knot detection, `SGKnots`, which was developed by Buchert [64], could be used to determine which type of knot had been tied, however this was not implemented.

Chapter 7

Conclusion

7.1 Summary of Achievements

As robot-assisted surgery becomes more advanced, so too does the demand for surgical simulators. These are software tools that allow a surgeon to practice complex operations in a realistic virtual environment without the need to have any patient present or the use of physical props. `SGPhysics` is a new physics engine in development. It is designed to meet the needs of simulating these virtual surgical procedures. Currently it does not include a model for suturing - the process of passing surgical thread through tissue in order to join it together. The development of a complete suturing simulation for `SGPhysics` was the goal of this project.

Different models were analysed and assessed for their suitability for the purpose of real-time virtual surgical simulations. The spline model was determined to be most suitable for representing sutures due to being mathematically very precise with only a few degrees of freedom. A mass-spring model was selected for the tissue for its efficiency. Other models were used to define rigid bodies which are also useful in surgical simulations. Each of these models were built into the physics engine `SGPhysics`.

The physics of the model were defined using Lagrangian mechanics in order to guarantee conservation of energy and due to its suitability in describing dynamic systems. Several solver methods were designed and built with the basis of DAE theory. They were analysed analytically and also tested against each other for verification. The most suitable method from the ones tested was to formulate the problem with the constraints defined on the acceleration level and to solve the system with Newton's method. By defining the system constraints on the acceleration level, this converted the system from an index-3 to

and index-1 system, which reduces the effect of perturbances on this type of stem.

A range of constraints were designed and implemented. They were defined using Lagrangian multipliers and added to the system. Because the system is defined using multibody DAE theory, constraints can be added directly to the system. The solution can then be computed, solving both the dynamic mechanics and the constraints simultaneously. This method is more stable than solving both separately, and means that the projection step has only minimal corrections to make. Furthermore, a collision response scheme was developed, based on the LCP, in order to handle collisions between bodies and measure the forces between objects.

Importantly, a suturing model was developed to simulate the behaviour of a suture passing through tissue as part of a surgery. This is a dynamic constraint that is created once a sufficient force is applied between the suture tip and the tissue. The creation and initial behaviour of the constraint was exactly as expected, however certain instabilities arose in the tissue which may be attributed to the choice of model for the tissue. Other models such as the rigid multibody systems could be defined and behaved as expected.

Overall the suturing model met the requirements although performance-wise the model could be improved. The Newton method for solving the system was accurate and stable, and the use of a direct solver instead of an iterative solver was probably justified due to the relatively lower number of variables. An iterative solver could exploit parallelisation, but this would only really be advantageous with systems of over 1000 or even $10{,}000$ variables, while the systems used here are in the hundreds. In the end, the goal of creating a suturing simulation was achieved and optimisation is left as a future task.

7.2 Future Work

As discussed above, an area of investigation is an optimisation task to ensure that the evaluation of the solution is occurring on the scale of tenths of a second. This could be partially achieved by using a method capable of parallelisation such as conjugate gradient or steepest descent. Similarly the numerical integration method could be expanded to allow for higher order methods. This would allow for solving more complex and stiff problems, and help avoid instabilities.

From a more practical perspective, the expansion of the program to include more types of bodies and constraints would make it possible to build more complex models. A very useful inclusion would be to develop a FEM model

for the tissue in order to accurately model physical behaviour such as elasticity and viscosity. It would also be interesting to investigate other suturing models such as the Cosserat model, as it is a physically-based continuous model, and to compare the differences in behaviour.

Finally, from an integration perspective, there are a number of different projects that could combine well with this one. Integration of `SGKnots` into `SGPhysics` would allow for detection and identification of knots based on the configuration of a suture. Integration of `SGPhysics` and `Robotic Surgery` would expand the number of devices that could run and interact with `SGPhysics` to include medical devices. These tasks, which are beyond the scope of this research thesis, would open up the work produced here to be used in medical simulators and expand their capabilities.

Bibliography

[1] Mehmet Haksever et al. “Modified Continuous Mucosal Connell Suture for the Pharyngeal Closure After Total Laryngectomy: Zipper Suture”. In: *Clinical and Experimental Otorhinolaryngology* 8.3 (2015), pp. 281–288. ISSN: 1976-8710. URL: https://doaj.org/article/a3046aa6bbb54583b68cd06d6e0df700.

[2] Howard Taylor and Alan W. Grogono. “The constrictor knot is the best ligature”. In: *Annals of the Royal College of Surgeons of England* 96.2 (2014), pp. 101–105. ISSN: 00358843.

[3] Shawn M. Garber and Jonathan M. Sackier. “Suturing and Tissue Approximation”. In: *Current Review of Minimally Invasive Surgery*. Ed. by David C. Brooks. New York, NY: Springer New York, 1998, pp. 179–187. ISBN: 978-1-4612-1692-6. DOI: 10.1007/978-1-4612-1692-6_18.

[4] Jim C Hu et al. “Comparative Effectiveness of Minimally Invasive vs Open Radical Prostatectomy”. In: *JAMA: The Journal of the American Medical Association* 302.14 (2009), pp. 1557–1564. ISSN: 0098-7484.

[5] Ramnath Subramaniam. “Current Use of and Indications for Robot-assisted Surgery in Paediatric Urology”. In: *European Urology Focus* 4.5 (2018), pp. 662–664. ISSN: 2405-4569.

[6] Oussama Elhage et al. “An assessment of the physical impact of complex surgical tasks on surgeon errors and discomfort: a comparison between robot-assisted, laparoscopic and open approaches”. In: *BJU International* 115.2 (2015), pp. 274–281. ISSN: 1464-4096. DOI: 10.1111/bju.12680.

[7] Vincenzo Ficarra et al. “Retropubic, Laparoscopic, and Robot-Assisted Radical Prostatectomy: A Systematic Review and Cumulative Analysis of Comparative Studies”. In: *European Urology* 55.5 (2009), pp. 1037–1063. ISSN: 0302-2838.

[8] Michael Amirian et al. “Surgical suturing training with virtual reality simulation versus dry lab practice: an evaluation of performance improvement, content, and face validity”. In: *Journal of Robotic Surgery* 8.4 (2014), pp. 329–335. ISSN: 1863-2483.

[9] Anderson Maciel et al. “Using the PhysX engine for physics-based virtual surgery with force feedback”. In: *International Journal of Medical Robotics and Computer Assisted Surgery* 5.3 (2009), pp. 341–353. ISSN: 1478-5951.

[10] Eusebio Ricardez et al. “SutureHap: Use of a physics engine to enable force feedback generation on deformable surfaces simulations”. In: *International Journal of Advanced Robotic Systems* 15.1 (2018). ISSN: 1729-8814.

[11] Gyusung Lee and Mija Lee. “Can a virtual reality surgical simulation training provide a self-driven and mentor-free skills learning? Investigation of the practical influence of the performance metrics from the virtual reality robotic surgery simulator on the skill learning and associated cognitive workloads”. In: *Surgical Endoscopy* 32.1 (2018), pp. 62–72. ISSN: 0930-2794.

[12] Joel Brown, Jean-Claude Latombe, and Kevin Montgomery. “Real-time knot-tying simulation”. In: *The Visual Computer* 20.2 (2004), pp. 165–179. ISSN: 0178-2789.

[13] Matthias Müller, Tae-Yong Kim, and Nuttapong Chentanez. “Fast Simulation of Inextensible Hair and Fur.” In: *VRIPHYS* 12 (2012), pp. 39–44.

[14] Zhuopeng Zhang and Shigeo Morishima. “Real-time Hair Simulation on Mobile Device”. In: *Proceedings of Motion on Games*. MIG ’13. Dublin 2, Ireland: ACM, 2013, 127:149–127:154. ISBN: 978-1-4503-2546-2. DOI: 10.1145/2522628.2522905.

[15] Nobuyuki Umetani, Ryan Schmidt, and Jos Stam. “Position-based Elastic Rods”. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’14. Copenhagen, Denmark: Eurographics Association, 2014, pp. 21–30. URL: http://dl.acm.org/citation.cfm?id=2849517.2849522.

[16] Patricia Moore and Derek Molloy. “A Survey of Computer-Based Deformable Models”. In: *International Machine Vision and Image Processing Conference (IMVIP 2007)*. IEEE, 2007, pp. 55–66. ISBN: 0769528872.

[17] Robert E. Rosenblum, Wayne E. Carlson, and Edwin Tripp III. “Simulating the structure and dynamics of human hair: Modelling, rendering and animation”. In: *The Journal of Visualization and Computer Animation* 2.4 (1991), pp. 141–148. DOI: 10.1002/vis.4340020410.

[18] Fei Wang et al. “Knot-tying with Visual and Force Feedback for VR Laparoscopic Training”. In: *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*. Vol. 6. Jan. 2005, pp. 5778–5781. DOI: 10.1109/IEMBS.2005.1615801.

[19] Matt LeDuc, Shahram Payandeh, and John Dill. “Toward Modeling of a Suturing Task.” In: *Graphics interface*. Vol. 3. Citeseer. 2003, pp. 273–279.

[20] Jeff Phillips, Andrew Ladd, and Lydia E Kavraki. “Simulated knot tying”. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*. Vol. 1. IEEE. 2002, pp. 841–846.

[21] Ken-Ichi Anjyo, Yoshiaki Usami, and Tsuneya Kurihara. “Simple method for extracting the natural beauty of hair.” In: *Computer Graphics (ACM)*. Vol. 26. 2. 1992, pp. 111–120. URL: http://search.proquest.com/docview/23801622/.

[22] Jérémie Dequidt et al. “Interactive Simulation of Embolization Coils: Modeling and Experimental Validation”. In: *Medical Imaging Computing and Computer Assisted Intervention -MICCAI’08*. Vol. 5241. Lecture Notes in Computer Science 1. Springer, 2008, pp. 695–702. ISBN: 354085987X.

[23] Janusz Stanislaw Przemieniecki. *Theory of matrix structural analysis*. Courier Corporation, 1985.

[24] Christophe Guébert. “Suture en chirurgie virtuelle : simulation interactive et modèles hétérogènes”. PhD thesis. Université des Sciences et Technologie de Lille - Lille I, Laboratoire d’Informatique Fondamentale de Lille, France, July 2010. URL: https://tel.archives-ouvertes.fr/tel-00561061.

[25] Thanh-Nam Le, Jean-Marc Battini, and Mohammed Hjiaj. “A consistent 3D corotational beam element for nonlinear dynamic analysis of flexible structures”. In: *Computer Methods in Applied Mechanics and Engineering* 269 (2014). ISSN: 0045-7825.

[26] Stephane Cotin et al. “New approaches to catheter navigation for interventional radiology simulation”. In: vol. 3750. 2005, pp. 534–542. ISBN: 3540293264.

[27] Eugene Cosserat and François Cosserat. *Théorie des corps déformables*. Paris: Librairie Scientifique A. Hermann et fils, 1909.

[28] Dinesh K. Pai. “STRANDS: Interactive Simulation of Thin Solids using Cosserat Models”. In: *Computer Graphics Forum* 21.3 (2002), pp. 347–352. ISSN: 0167-7055. DOI: 10.1111/1467-8659.00594. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.00594.

[29] Lang Xu and Qian Liu. “Real-time inextensible surgical thread simulation”. In: *International Journal of Computer Assisted Radiology and Surgery* 13.7 (2018), pp. 1019–1035. ISSN: 1861-6410. DOI: 10.1007/s11548-018-1739-1. URL: https://doi.org/10.1007/s11548-018-1739-1.

[30] Jonas Spillmann and Matthias Teschner. “CoRdE: Cosserat Rod Elements for the Dynamic Simulation of One-dimensional Elastic Objects”. In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’07. San Diego, California: Eurographics Association, 2007, pp. 63–72. ISBN: 978-1-59593-624-0. URL: http://dl.acm.org/citation.cfm?id=1272690.1272700.

[31] Miklós Bergou et al. “Discrete Elastic Rods”. In: *ACM SIGGRAPH 2008 papers*. Vol. 27. SIGGRAPH ’08 3. ACM, 2008, pp. 1–12. ISBN: 9781450301121.

[32] Mikhail Beliaev, Vladimir Lalin, and Vladimir Kuroedov. “Geometrically Nonlinear Rods Theory - Comparison of the Results Obtained by Cosserat-Timoshenko and Kirchhoff’s Rod Theories”. In: *Applied Mechanics and Materials* 725.Innovative Technologies in Development of Construction Industry (2015), pp. 629–635. ISSN: 1660-9336.

[33] Zhujiang Wang et al. “Real time simulation of inextensible surgical thread using a Kirchhoff rod model with force output for haptic feedback applications”. In: *International Journal of Solids and Structures* 113-114.C (2017), pp. 192–208. ISSN: 0020-7683.

[34] Wikimedia Commons. *File:Parametic Cubic Spline.svg — Wikimedia Commons, the free media repository*. [Online; accessed 20-March-2019]. 2018. URL: https://commons.wikimedia.org/w/index.php?title=File:Parametic_Cubic_Spline.svg&oldid=305157978.

[35] Demetri Terzopoulos et al. "Elastically deformable models". In: *Proceedings of the 14th annual conference on computer graphics and interactive techniques*. SIGGRAPH '87. ACM, 1987, pp. 205–214. ISBN: 0897912276.

[36] Julien Lenoir et al. "Surgical thread simulation". In: *ESAIM: Proceedings*. Vol. 12. EDP Sciences. 2002, pp. 102–107.

[37] Olivier Nocent and Yannick Rémion. "Continuous deformation energy for Dynamic Material Splines subject to finite displacements". In: *Computer Animation and Simulation 2001: Proceedings of the Eurographics Workshop in Manchester, UK, September 2–3, 2001*. Ed. by Nadia Magnenat-Thalmann and Daniel Thalmann. Vienna, Sept. 2001, pp. 87–97. ISBN: 978-3-211-83711-5. DOI: 10.1007/978-3-7091-6240-8_9.

[38] Julien Lenoir et al. "A Suture Model for Surgical Simulation". In: *ISMS*. 2004.

[39] Adrien Theetten. "Splines dynamiques géométriquement exactes : simulation haute performance et interaction". PhD thesis. Université des Sciences et Technologie de Lille - Lille I, Laboratoire d'Informatique Fondamentale de Lille, France, 2007, pp. v–182. URL: http://www.theses.fr/2007LIL10117/document.

[40] Adrien Theetten et al. "Geometrically exact dynamic splines". In: *Computer-Aided Design* 40.1 (2008), pp. 35–48. ISSN: 0010-4485.

[41] Adrien Theetten and Laurent Grisoni. "A robust and efficient Lagrangian constraint toolkit for the simulation of 1D structures". In: *Computer-Aided Design* 41.12 (2009), pp. 990–998. ISSN: 0010-4485.

[42] Karolina Golec. "Hybrid 3D Mass Spring System for Soft Tissue Simulation". PhD thesis. Université Claude Bernard - Lyon I, ED 512 École Doctorale en Informatique et Mathématiques, France, 2018. URL: https://tel.archives-ouvertes.fr/tel-01761851.

[43] Eftychios Sifakis and Jernej Barbič. *Finite Element Method Simulation of 3D Deformable solids*. Synthesis digital library of engineering and computer science. 2016. ISBN: 1-62705-443-X.

[44] Jérémie Allard et al. “SOFA - an Open Source Framework for Medical Simulation”. In: *MMVR 15 - Medicine Meets Virtual Reality*. Vol. 125. Studies in Health Technology and Informatics. IOP Press, 2007, pp. 13–18.

[45] François Faure et al. “SOFA: A Multi-Model Framework for Interactive Physical Simulation”. In: *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*. Vol. 11. Studies in Mechanobiology, Tissue Engineering and Biomaterials. Springer, 2012, pp. 283–321. ISBN: 9783642290138.

[46] Matthias Müller et al. “Position Based Dynamics”. In: *Journal of Visual Communication and Image Representation* 18.2 (2007), pp. 109–118. ISSN: 1047-3203.

[47] Matthias Müller et al. “Meshless Deformations Based on Shape Matching”. In: *ACM Transactions on Graphics (TOG)* 24.3 (2005), pp. 471–478. ISSN: 1557-7368. DOI: 10.1145/1073204.1073216.

[48] Miles Macklin et al. “Unified Particle Physics for Real-time Applications”. In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), pp. 1–12. ISSN: 1557-7368. DOI: 10.1145/2601097.2601152.

[49] Christian Duriez et al. “Interactive Simulation of Flexible Needle Insertions Based on Constraint Models”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2009*. Ed. by Guang-Zhong Yang et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 291–299. ISBN: 978-3-642-04271-3.

[50] Pedro Moreira et al. “Modelling Prostate Deformation: SOFA versus Experiments”. In: *Mechanical Engineering Research* 3 (Aug. 2013), pp. 64–73. DOI: 10.5539/mer.v3n2p64.

[51] Bernd Simeon. *Computational Flexible Multibody Dynamics: A Differential-Algebraic Approach*. Differential-Algebraic Equations Forum. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. ISBN: 978-3-642-35157-0.

[52] Wikimedia Commons. *File:Least action principle.svg — Wikimedia Commons, the free media repository*. [Online; accessed 11-February-2019]. 2016. URL: https://commons.wikimedia.org/w/index.php?title=File:Least_action_principle.svg&oldid=217680619.

[53] Carl De Boor. *A Practical Guide to Splines*. Revised Edition. Vol. 27. Applied Mathematical Sciences. New York: Springer, 2001. ISBN: 0-387-95366-3.

[54] Julian Valentin. "B-Splines for Sparse Grids: Algorithms and Application to Higher-Dimensional Optimization". PhD thesis. Universität Stuttgart, Institut für Parallele und Verteilte Systeme, Fachbereich Informatik, Germany, 2019. URL: http://dx.doi.org/10.18419/opus-10504.

[55] Edda Eich-Soellner and Claus Führer. *Numerical Methods in Multibody Dynamics*. European Consortium for Mathematics in Industry. 1998. ISBN: 3-663-09828-1.

[56] Pavel S. Petrenko. "Robust Controllability of Linear Differential-Algebraic Equations with Unstructured Uncertainty". In: *Journal of Applied and Industrial Mathematics* 12.3 (2018), pp. 519–530. ISSN: 1990-4789. DOI: 10.1134/S1990478918030122.

[57] Volker Mehrmann. "Index Concepts for Differential-Algebraic Equations". In: *Encyclopedia of Applied and Computational Mathematics* (2015), pp. 676–681.

[58] Peter Kunkel and Volker Mehrmann. *Differential-Algebraic Equations: Analysis and Numerical Solution*. EMS Textbooks in Mathematics. Zürich: European Mathematical Society, 2006. ISBN: 3-03719-017-5.

[59] Ernst Hairer, Michel Roche, and Christian Lubich. *The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods*. Vol. 1409. Lecture Notes in Mathematics. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989. ISBN: 978-3-540-51860-0.

[60] Michael B. Cline and Dinesh K. Pai. "Post-stabilization for rigid body simulation with contact and constraints". In: *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*. Vol. 3. IEEE, Sept. 2003, 3744–3751 vol.3. ISBN: 0780377362. DOI: 10.1109/ROBOT.2003.1242171.

[61] Stephen C. Billups and Katta G. Murty. "Complementarity Problems". In: *Journal of Computational and Applied Mathematics* 124.1 (2000), pp. 303–318. ISSN: 0377-0427.

[62] James M. Ortega and Werner C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Saint Louis: Elsevier Science & Technology, 1970. ISBN: 9780125285506.

[63] Ioannis K Argyros. *Convergence and Applications of Newton-type Iterations*. 2008. ISBN: 1-281-49172-1.

[64] Thibaut Buchert. *Rapport de stage: Stage de fin d'études du 23 avril au 18 octobre 2018 à SenseGraphics*. English. Tech. rep. ENSIIE, Université de Strasbourg, 2018.

TRITA -SCI-GRU 2019:359