Robotics Engineering course

University of Genoa

Things I did

Student Chiara Saporetti **Year:** 2020 - 2021

Contents

Figure 1: Geomagic touch haptic device

1 Materials and methods

1.1 Materials: Geomagic Touch

The Geomagic Touch is a mid-range, professional haptic device commonly used in medical and surgical simulation research 1.

It applies force feedback on the users' hand, allowing them to feel virtual objects and producing true-to-life touch sensations as users manipulates on-screen 3D objects. It allows 6 degrees of freedom and gives a kinesthetic active force feedback on the x,y,z axes from the measurement. To define the force feedback it measures the instantaneous position and velocity from joint proprioceptive sensors (optical encoders).

EQUATIONS (WRITE FULLY LATER) Forward kinematic equations:

$$x = f(q) \tag{1}$$

eccetera.

Inverse kinematic equations:

$$\theta = f^{-1}(x) \tag{2}$$

Kineshetic force feedback:

$$\tau = J^T F \tag{3}$$

ecc

In particular, the human operator applies a force Fh to the pencil by changing its joint coordinates, ad these signals are recorded by the

optical encoders. With the use of forward kinematic models of position and velocity it is possible to obtain the operational coordinates that define the position and velocity of the human operator in the Virtual Environment. When the haptic interaction point comes into contact with a virtual object a reaction force is computed. This force allows to compute the joints torque based on the equation 3 to recreate the kinesthetic stimulus on the Geomagic pencil.

1.2 Methods: SOFA Framework

SOFA (Simulation Open-Framework Architecture) is an open source software targeted at real-time simulation, with an emphasis on medical simulation. It is written in C++ and, since February 2021, bindings to allow coding in Python 3 were created.

The installation procedure of this software is not immediate since it must be compiled from sources. For this reason I wrote a tutorial [2] on how to install sofa and all the main plugins, in the hope that it will help students of Simav and sofa users in general.

In the following I will describe the main concepts of sofa.

1.2.1 The three models and the concept of mappings

In SOFA, any object can be simulated using three different models: deformation/behaviour, collision and visual. To maintain a continuity between the models, they are then linked by mappings.

The *behavior model* describes the internal mechanical behaviour and is typically represented with a tetrahedra based geometry. This comes from the idea of *discretization of space*: the object is modelled with a dynamic or quasi-static system of particles (simulation nodes) whose coordinates are the independent DOFs of the object and are calculated by integrating $F = ma$ at each time interval dt (*discretization of time*).

The *collision model* takes care of the interaction between objects via different collision detection approaches.

The *visual model* implements the realistic rendering. This is particularly important in the context of surgical simulation for training to have a suspension of disbelief i.e. when the user forgets that he or she is dealing with a simulator.

The *mappings* maintain the coherency between the models. The main idea is that one of the models, typically the behavior one, acts as the master, and imposes its displacements to slaves using mappings.

The positions and the velocities are propagated *top-down* in the hierarchy. Conversely, the forces are propagated *bottom-up* to the independent DOFs, where Newton law f = Ma is applied. Given forces fs applied to a slave model, the mapping computes and accumulates the equivalent forces fm applied to its master. The equations that model these mappings are shown below.
Master-slave mappings equations:

$$x_s = f(x_m) \tag{4}$$

$$v_s = Jv_m \tag{5}$$

$$a_s = Ja_m + \frac{\partial J}{\partial x_m} v_m \tag{6}$$

$$f_m = J^T f_s \tag{7}$$

1.2.2 Topologies

All models are based on meshes, and in particular they all have a *mesh geometry* that describes the location of mesh vertices in space, and a *mesh topology* that indicates how vertices are connected to each other by edges, triangles or any type of mesh element.

Shapes and meshes can be imported from external files in different formats (obj, msh, vtk, ...) and one of the suggested softwares that allows to generate meshes is Gmsh [1].

Topology objects are composed of four functional members (Container, Modifier, Geometry and Algorithms) to create, modify the topologies arrays, or give access to geometrical information.

Making a *topological mapping* means defining a mesh topology from another mesh topology by using the same DOFs. Mappings can be used either to go from one topology to a lower one in the topological hierarchy (from tetrahedra to triangles), or to split elements (quads into triangles). As usual mappings, forces applied on the slave topology are propagated onto the master one.

1.2.3 Data organization

In sofa, everything is implemented with a set of *connected scenegraph nodes*. Each node has *components* that are responsible for a reduced set of tasks implemented using virtual functions in an object oriented approach.

The component parameters are stored in member objects using Data containers and the user can access them through *engine components*.

The whole data structure (scenegraph) is processed using *visitors*, which apply virtual functions to each node they traverse. Then each node will apply virtual functions to the components they contain.

The *AnimationLoop* is the component that components orders all the steps of the simulation and the system resolution. At each time step, the animation loop triggers each event (solving the matrix system, managing the constraints, detecting the collision, etc.) through the visitor mechanism.

1.2.4 Simulation algorithms

All dynamic simulations assume to discretize the temporal evolution of the system through small time steps. This time step is usually noted dt. At each dt, the system must solve the equation F=ma to find the velocities and positions of all the points in the simulation.

An integration scheme is the numerical methods that describe how to find the approximate solution to the equation and define how the linear matrix system F=ma must be built. In particular Sofa provides two kinds of schemes: implicit (which are slower but more stable) and explicit (which are faster but need a very small dt to be stable). Since I wanted a stabler simulation I used *implicit solvers*. They define the F=ma as

$$(\alpha M + \beta B + \gamma K)\delta v = b \tag{8}$$

Where: M is the mass matrix, K is the stiffness matrix ($K = \frac{\partial f}{\partial x}$) and B is the damping matrix ($B = \frac{\partial f}{\partial v}$).
Implicit integration has the advantage of being more stable for stiff forces or large time steps. However, the solution of these equation systems requires linear solvers and SOFA offers many options to choose from.

1.2.5 Collision detection and response

For each object, a bounding volume hierarchy is defined: all nodes are wrapped in bounding volumes. then the nodes are grouped in small sets enclosed within larger bounding volumes in a recursive way. This will finally result in a tree structure with a single bounding volume at the top of the tree. The collision detection will then be divided in phases, each implemented in a different component and scheduled

by a CollisionPipeline component. Each potentially colliding object is associated with a collision geometry based on or mapped from the independent DOFs.

The *broad phase* quickly removes object pairs that are not in collision by checking the bounding boxes. It then returns the pairs that may potentially collide. Based on the first phase, the *narrow phase* defined which elements are actually colliding. It returns pairs of geometric primitives, along with the associated contact points. This is passed to the contact manager, which creates contact interactions (the *collision response* of various types based on customizable rules.

1.2.6 Constraints (finish later)

AS previously stated, the animation loop takes care of solving constraint. Constraints in sofa can be of two types. They may be *projective constraints* and allow to project the velocity of a constrained point to a desired velocity. They can work either by attaching two objects or by attaching a single object and some fixed points in space.

Conversely, the *Lagrangian constraints* handle complex constraints that can not be implemented using projection matrices only. The resolution of the constraint problem can be done using the Gauss-Seidel algorithm, which has different implementations in sofa.

To solve these constraints, the free motion animation loop needs a Constraint correction component:

- UncoupledConstraintCorrection: makes the approximation that the compliance matrix Compliance matrix is diagonal. Strong assumption: constraints are independent.
- LinearSolverConstraintCorrection: computes the compliance matrix, by using $A^{-}1$ from a direct solver. This approach can therefore be very computationally-demanding if you have many constraints. Note that this ConstraintCorrection proposes an optimization for wire-like structures (boolean option). Dani's one.
- recomputedConstraintCorrection: instead of computing Inverse of A at each time step, this constraint correction precomputes once the inverse of A at the initialization of the simulation and stores this matrix. Simulation is faster but lacks accuracy if A changes during simulation.
- GenericConstraintCorrection

Constraint laws:

- bilateral interaction: defines an holonomic constraint law between a pair of simulated body. Such a constraint is suited for attachment cases or sliding joints (paired objects).
- unilateral interaction: defines an non-holonomic constraint law between a pair of simulated body. defines an non-holonomic constraint law between a pair of simulated body. Must also use a ConstraintCorrection so that the corrective motion can be applied. Unlike other constraints, the UnilateralInteractionConstraint is mostly used in SOFA for contact modeling (contact and collision cases)
- Sliding interaction: like a bilateral but only active for some vectors of the physics space (for instance only the x-direction)

1.2.7 Additional plugins: Geomagic plugin

The *Geomagic plugin* allows the interfacing between SOFA and the Geomagic Touch device.

It should be noted that an important issue in haptic rendering is the need for at least 1000Hz in computing forces, haptic feedback could artificially add some energy inside the simulation that creates instabilities, if the control is not passive.

In SOFA this is solved via the *virtual coupling*. When working with the Geomagic device, the position of the end effector is provided by the API and this API asks for force values from the application. This could however generate instabilities, and so a virtual mechanical coupling is set. It corresponds to the use of a damped stiffness between the position measured on the device and the simulated position in the virtual environment. If very stiff constraints are being simulated, then the stiffness perceived by the user will not be infinite but will correspond to the stiffness of this virtual coupling. Hence, a compromise between stability and performance must be found by tuning the stiffness value of the coupling.

The *main advantage* of the virtual coupling technique is that it can be easily employed with every simulation of SOFA. *The main drawback* is that the haptic rendering is not transparent (i.e. the haptic interaction does not feel the same as the real interaction it is reproducing).

1.2.8 Additional plugins: SofaCarving

The *SofaCarving plugin* provides one component, the CarvingManager, that, when added to a scene, automatically allows to remove tetrahedra in run-time as soon as a collision is detected between the Carving Tool and the Carving Surface. These two must be defined by giving the right components the tags CarvingTool and CarvingSurface.

1.2.9 Additional plugins: SofaPython3

The *SofaPython3 plugin* brings python bindings for the SOFA simulation framework. The first version, then continuously updated, was released in February 2021. sWith this plugin it is possibile to define both the scenegraph and the actions to perform during the simulation directly into a python script. Most of sofa's components and features are available and, if needed, the user can also create bindings for them by adding to the existing ones.

This project is composed of two modules: the SOFA plugin (that embeds a python interpreter and a SOFA scene loader) and several python modules (C++ bindings of SOFA's APIs for python 3.7).

1.3 Methods: Gmsh

Gmsh is an open source 3D finite element mesh generator with a built-in CAD engine and post-processor. Its design goal is to provide a fast, light and user-friendly meshing tool with parametric input and advanced visualization capabilities. I used it to model the skin and the thread as boxes with small tetrahedral meshes.

2 Skin and instrument models

A part from the general parameters, all of the object models are defined into a separate code so that they can be called when needed.

2.1 General parameters

In Sofa it is necessary to specify some root simulation parameters prior to building the node hierarchy. These parameters are defined inside the main of each task simulation. In the following I give a brief description of the ones that I defined.

2.1.1 Animation Loop

I chose the animation loop that must be used when working with the Geomagic Plugin: the *FreeMotionAnimationLoop*. In particular, the FreeAnimationLoop divides each simulation step into two successive resolution steps: the free motion (unconstrained system Ax=b, which may also include projective constraints and collision detection responses; the solution is called x_free) and a corrective motion (the animation loop will look for an existing ConstraintSolver in the scene graph: it will handle the entire constraint process: allocation of memory needed, computation of the constraint system, resolution and application of the corrective motion ensuring valid constraints). This type of animation loop requires a *ConstraintSolver* and a *ConstraintCorrection*, which will be defined and analyzed later.

2.1.2 Collision pipeline

The collision pipeline, as stated in the previous Chapter, follows three steps: a reset of the collision, a collision detection (broad and narrow phases), a collision response. The pipeline itself is defined by the depth at which sofa should go when looking for collisions inside the Bounding Volume Hierarchy (I set 6 as suggested on the Geomagic Plugin examples).

To implement the *Broad phase* I used the *brute force* component. It is based on the comparison of the overall bounding volumes in the Bounding Volume Hierarchy to determine if they are in collision or not.

For the *Narrow phase* Sofa provides two possible component: the minProximityIntersection and the localMinDistance. Both of them detect a possible contact as soon as pair of collision elements are close

to each other (i.e. the distance is smaller than a predefined alarmDistance) and create contact when the distance is lower than contactDistance. However, in minProximityIntersection, the intersection might have a high number of contacts and, by using a response method based on Lagrange multipliers, there would be too many constraints generated i.e. too computationally demanding. For this reason I used the *localMinDistance* component with alarm distance 0.3 and contactDistance 0.1 (it is generally better to set a contact distance that stays above zero).

For the *Response* there is only one option available in SOFA, and it is the default contact manager. However, it has some parameters that allow for a greater variation. I chose, for example, to set a friction contact.

2.1.3 Integration scheme

As specified in the previous Chapter, I needed to chose the time step and the integration scheme. Regarding the time step: choosing a very small dt is more computationally demanding but also more accurate. I chose dt=0.1 which is a good trade-off.

As integration scheme I used the implicit Euler, which is a time integrator that uses backward Euler scheme for first and second order Ordinary Differential Equations. To define it, two parameters must be set: RayleighStiffness and RayleighMass. They both refer to the Rayleigh damping, which is a numerical damping with no physical meaning. It corresponds to a damping matrix that is proportional to the mass or/and stiffness matrices using coefficients, respectively Rayleigh stiffness factor or Rayleigh mass factor. This numerical damping is usually used to stabilize or ease convergence of the simulation. The values should be tuned.

2.1.4 Solvers

As previously stated, the *linear solvers* solve the Ax=b problem that was defined by the integrator. Among the possible *linear solvers*, I used the *Conjugate gradient linear solver*, that solves the equation without prior knowledge. It relies on conjugate gradient method: $r = b - Ax^k$, where r is the residual, which is used to compute mutually conjugate vectors to be used as a basis for the approximate solution.

To solve constraints a *constraint solver* is instead needed. In particular, the Geomagic plugin needs the LCPConstraintSolver component, which targets collision constraints, contacts with frictions which corresponds to unilateral constraints.

2.2 Skin parameters

My skin model is composed of three models mapped on each other. All the mechanical objects are of the type Vec3d, the variable type that describes 3D objects with meshes.

2.2.1 Behavior model

It is made of a tetrahedral mesh imported as a msh file with the MeshGmshLoader.

The choice of this type of mesh is due to the fact that it is the only one that can be carved via the Sofa Carving plugin. I modelled it via the Gmsh software [1] and set dimensions of 30x20x5 with meshes of 0.1. I had to decide this values by trial and error: I needed a surface that was big enough to allow the user to move confortably, but not too big or it would be too computationally demanding to check it all continuously for collisions. Also, I wanted it thick enough to be pierced for suture without touching the underneath layer (which is fixed). But I also wanted small meshes to give a nice looking effect. In the end, these values were was a good trade-off.

As for any topology I defined a topology container and geometry algorithms. Also, since I wanted to use the Carving plugin, I defined the topology modifier.

The core of the model is given by the mechanical object, which is of type Vec3d since the object is deformable.

Then, as previously stated, I took care of building the F=ma system.

For the mass I considered a diagonal mass with density 2 (?).

For the force I considered the TetrahedronFEMForceField. To define this force it is necessary to specify the youngModulus and poissonRatio of the skin. I took into consideration the values that I found in the literature but in the end I chose the ones that made the simulation look more realistic: young modulus 300, poisson ratio 0.49.

To solve the internal deformations with F=ma, I set the Euler implicit solver and the CG linear solver as for the root node.

To solve contraints I temporarely set the uncoupled constraint correction with compliance 0.001 as suggested on the sofa forum.

Then, inside this model I defined multiple *boxROIs*. A boxROI is a data engine used to find the primitives (vertex/edge/triangle/quad/tetrahedron/hexahedron) inside it. In particular I used one for the base of the skin, one for the border and some others for other purposes that will be explained later. The one at the base was useful to fix the model in the desired position (or it would fall subject to gravity), while the others are to save indices that will be used in the realization of the tasks. To fix the indices I used the component RestShapeSpringsForceField, that brings all the indices back to the original position after a deformation.

2.2.2 Collision model

It is composed of a triangular mesh generated by applying a topological transformation to the tetrahedral mesh. In fact, what I did was to define triangle topology container, modifier and geometry algorithms, and then I created a component *Tetra2TriangleTopologicalMapping* from the previous tetrahedra container to the new triangle container. Since topological mappings always refer to the same mechanical object, I did not need to define a new one for the collision model.

Finally, in order to use the Carving option, I defined a triangle collision model and set it to a CarvingSurface.

2.2.3 Visual model

It is defined as an identity mapping from the collision model, and therefore it has its own mechanical object.

It has a pink color to look more like the skin and the meshes are small enough to give a nice looking effect.

2.3 Suture needle and scalpel parameters

2.3.1 Behavior model

Its shape is imported as an obj file. It doesn't have meshes so it cannot be directly used as a collision model. However, I did not want to use the whole needle, I just need its ending points.

To solve the F=ma problem I set an Euler implicit and the CG linear solver.

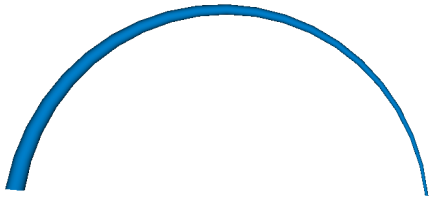

Figure 2: Needle

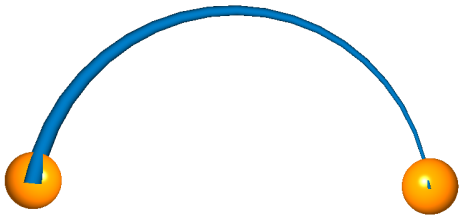

Figure 3: Needle

It has a Rigid3d Mechanical object, a mass of 3 (I received continuous warnings by setting it to lower values).

To solve constraints I temporarily set the uncoupled constraint correction with compliance 0.001 as suggested on the sofa forum.

2.3.2 Collision model

The suture needle model has 2 instances of *SphereCollisionModel* that correspond to two spheres, one for each extremity of the needle. The front sphere is used to carve the skin, while the back sphere is used to link the skin to the needle and simulate friction. I set the template of the spheres mechanical objects to Vec3d so that I could use them to attach springs, as will be explained later. Fig. 3.

In the case of the scalpel I only considered one sphere as a collision model, for simplicity.

The spheres are then mapped to the behavior with a rigid mapping so that they can move together.

2.3.3 Visual model

This is defined as a rigid mapping from the collision model. Note: I could not define an identity mapping since the needle melted (...ah boh.). The final result is shown in Fig. 2

2.4 Linking the instruments to the Geomagic Touch

I then proceeded to add the needle and scalpel to the Geomagic in the simulation. This should be done by adding a *RestShapeSpringsForceField* to link them to the Geomagic end effector (which creates the virtual coupling explained in the previous chapter) and by adding the *LCPForceFeedback* with a certain force coefficient, that i set to 0.1.

This last component finds the normal force generated in a collision.

The position of the instrument on the haptic device end-effector was chosen based on the comfort of the user when grasping the Geomagic end effector.

References

[1] *Gmsh software*. URL: https://gmsh.info/.

[2] *Installation procedure for Sofa, SofaPython3 and Geomagic plug-ins*. URL: https://github.com/ChiaraSapo/Haptic-surgery-simulation/blob/main/1_%5C%20Sofa%5C%20installation/SOFAv20.12.02%5C%2Bgeomagic%5C%2Bsofapython3%5C%20installation.pdf.