Integrated bilateral system for simulation and haptic control for endovascolar catheter insertion procedure

Adriano Pacciarelli 1816493
Flavio Galasso 2034377

September 2022

Contents

Abstract

The aim of this project is to create a simulator using the API SOFA that allows the user to perform a realistic medical procedure. Specifically, given the exceptional capacity of the simulator to render elastic behaviour of the materials, we strictly focused on endovascular catheter insertion, which is nowadays one of the main topics in this area.
The final goal is to assemble an interconnected environment, comprehensive of a simulator (SOFA) and an haptic interface (Geomagic), able to allow the user to move the instrument and at the same time receive kinestetic feedback from the simulation. The resulting tool could be useful to physicians for both practising in the catheter insertion procedure and during the real procedure, in order to have a 3D navigation system and to be able to perceive the forces acting on the tip of the tool to have a better understating of what the tool is in contact with, without the use of a camera.

1 Introduction

1.1 Endovascolar catheters

Catheter insertion is a fundamental part of the medical procedure called ***catheterization***, consisting in the insertion of a thin flexible tube, usually made out of latex, polyurethane, or silicone, known as catheter into the patient organs, through specific body channels. The final aim of the procedure can be either to drain fluids from the organ (usually for diagnostic purpose) or to delivery drugs or other medical device in specific parts, without involving the patient in surgical operation. Regarding endovascolar catheters, there are many types of possible procedures, with different devices, channels (mostly vessels) and organs involved, especially for the treatment of arterial diseases. At this time, this seems to be kind of revolution, since those procedures allow to treat various aortic diseases by reducing morbidity and mortality associated with open surgical treatment of vascular diseases.

Figure 1: Endovascolar catheter

Catheters major limitations regard range of shape, flexibility and maneuverability. Especially in case of endovascolar catheters, the length and the very small section of the tool make it very difficult insert. In addition, image guidance in most endovascular suites remains the fluoroscopy, which is only two-dimensional, requires a nephrotoxic contrast and exposes patients and staff to radiations. Because of these limitations, conventional catheters are difficult to use, lack active tip maneuverability and stability during wire exchanges and require frequent catheter changes, exposing the patient to all sorts of trauma and medical complications.

Consequently, some solutions to limit endovascular surgical instruments were developed by the use of robotic endovascular catheters, such as Hansen Magellan (Figure 2), Sensei X (Figure 3) and Artisan Extend (Figure 4). Those systems provide a precise steering of guide wires and proprietary robotic catheters through complex anatomy, and provides a highly stable platform for the manual delivery of therapy.

Figure 2: Hansen Magellan robot

Figure 3: Sensei X robot

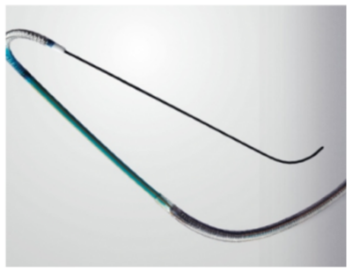


Figure 4: Artisan Extend Control Catheter.

In addition, new imaging platforms have been studied to improve endovascular navigation. The aim of those platforms is to give intuitive three-dimensional imaging, so that the procedure does not involve fluroscopy any more and remote guided devices can be used with precision within the arterial tree. Some important examples already in use are principally regarding master-slave systems for cardiac and urological surgery, such as the Da Vinci robot, allowing a greater precision of movement through a hand operated console. However, these technologies do not yet offer an adequate platform for endovascular procedures. For this motivation, physicians must perform catheter insertion procedures while seated at a remote physician console, away from radiation.

1.2 Project description

The aim of this project is to setup a catheter insertion simulation, specifically on that of a robotic endovascular catheter. This assignment can be identified in some smaller tasks, which are interconnected between each other:

1. **Simulation setup**: this part comprehends all the general settings of the scene, as well as the catheter model, phantom model and navigation model. In particular, the final result would be the one of an endovascolar catheter, that can be commanded form the base node (in rotation and translation) and pushed into some progressively more complicated vessel models.

2. **Forces extraction**: during the simulations, the interaction between the catheter and the vessel model is mapped as a constraining forces perceived by the catheter, which must be acquired and ordered.

3. **Haptic interface connection**: this part regard the connection between the simulator and the external device in both input and output. Specifically, the haptic interface has be be able to command the catheter navigating system as well as to map the constraining forces as kinestetic sensation to the user itself.

In conclusion, we used **SOFA framework** for the simulation with its **python plugin** to develop the scripts for the controller and forces extractor, and the **Geomagic** haptic interface.

1.3 SOFA framework

SOFA (*Simulation Open Framework Architecture*) is an open source framework for multi-physics simulation, available for **Linux**, **MacOS** and **Windows** environments.

SOFA engine is based on C++ and built using a flexible architecture. The base mechanisms of SOFA are implemented in a public core and, around this, several public modules and **plugins** are available, giving a high level of modularity, At the end SOFA appears to be an efficient tools for bench-marking and developing new training medical technologies.

Moreover, a simulation in SOFA is a Direct Acyclic Graph (DAG), therefore composed of nodes containing the components used for the computation (solvers, models, visualization, etc.).

Then SOFA architecture relies on a multi-model representation, which allows to have several representations of the of the same object:

- **Mechanical model**: it stores key mechanical components such as soft mechanics, degrees of freedom and all other state vectors (velocity, forces, etc.).
- **Collision model**: it describes how collisions between are handled, through collision detection and response methods.
- **Visual model**: it describes the physical appearance of the object (colour, shape, interaction with lights, etc.).

Those different representations are connected together through a mechanism called the "Mappings". With this features, it is also possible to have models of very different nature, such as **rigid models** (articulated bodies based on penalties or reduced coordinates), **deformable models** (mass spring and FEM with many different constitutive laws) and **fluid models**.
In addition, each representation has to follow an indicated **topology**, which can be either one already available in SOFA (points, edges, triangles, quads, tetrahedra, hexahedra) or directly loaded in various types of formats (obj, msh, vtk, etc.), trough the use of specific *Loaders*.

The advantages of using this framework are principally:

- **Modularity**: the availability of several public plugins is extremely helpful, since allows an easy access to useful libraries and models.
- **Python scripting**: even if the usual description of a SOFA simulation is performed using an XML syntax, the Python plugin allows to script using the Python language.
- **Complete interaction models**: nearly all types of physical interactions can be handled (with the appropriate models).

The drawbacks to face are principally due to the fact that it is a relatively new framework, meaning that is difficult to find information in the internet and the documentation is incomplete. This problem have been partially addressed with a quite active community answering in the forum of the SOFA itself.

2 Simulation construction

2.1 Simulation Setup

For the simulation setup the following nodes have been added with the following properties:

- **FreeMotionAnimationLoop**: The presence of this node enables the compututation of the projective constraints, the physics that solve the resulting free linear system. In a second pass, the correction step solves the constraints based on the Lagrange multipliers. The performance of this node has been increased by enabling multitasking with the following attributes: **parallelCollisionDetectionAndFreeMotion** and **parallelODESolving**
- **GenericConstraintSolver**: allows the use of Lagrange multipliers to handle complex constraints, such as contacts and joints between moving objects
- **DefaultPipeline**, **BruteForceBroadPhase**, **BVHNarrowPhase** to build a far and narrow collision pipeline for detecting contacts.
- **MinProximityIntersection** or **LocalMinDistance**: these nodes define the proximity method for intersection detection, they have two fundamental attributes:
  - **alarmDistance**: minimum distance between two object to start the detection of a possible contact
  - **contactDistance**: maximum distance where a contact is fully defined and detected

  We choose **MinProximityIntersection** at the cost of computational power because it is *more accurate* than **LocalMinDistance** that often lead to object penetration and unstable behaviour because of missing contacts.
- **CollisionResponse**: this defines the interaction between the contacts, it can be set to:
  - penalty method: efficient but subject to instability if not properly tuned
  - persistent method
  - constraints using Lagrange multipliers: processed by the solvers together with the other forces and constraints.

  We choose the **FrictionContactConstraint** because it was the most accurate and offered a **mu** factor to change the friction behaviour

2.2 General parameters

The large number of nodes involved in various aspects of the simulation led to the increase of variables scattered in the code, so we decided to enclose them in a dictionary called **catheter_vars**, among them we can find:

- "length": length in mm of the catheter;
- "controlPoints":control points that compose the catheter deformable and collision model, the higher the better, at the cost of computational power;
- "radius": outer radius of the catheter model;
- "innerRadius": inner radius of the catheter model;
- "FEMradius": theoretical radius of the catheter model for FEM computations;
- "massDensity": density of the catheter material in kg/mm^3;
- "youngModulus": modulus in MPa of the catheter material;
- "poissonRatio": Poisson ratio of the catheter material.

There are also other dictionaries, such as **object_pose** and **controller_vars**, useful for setting up the simulation delta time, the position of the objects and the increments done by the roto-traslation controller.

2.3 Catheter model

Unlike other projects made by students in the past, we decided to make the catheter model completely modular, i.e. automatically adapting via our custom code to the **length, radius and control points** indicated by the user in the aforementioned data structures. The object that manages constraints, deformation and movement is composed of a series of points connected by segments to compose a **1D straight line** (Figure 5). After our computation, each point gets assigned to a computed pose [x,y,z,qx,qy,qz,qw] and a connecting graph to its neighbour point, to form a **MechanicalObject** and a **MeshTopology**.

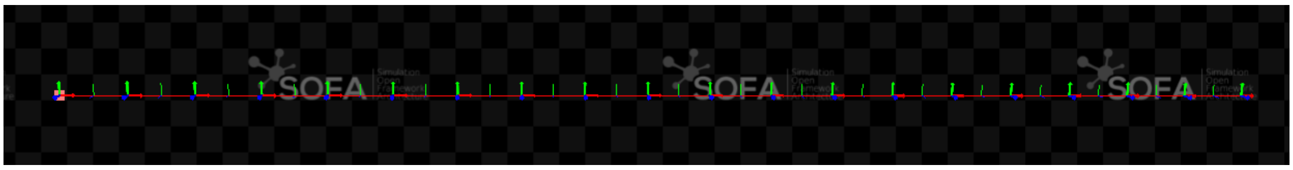


Figure 5: Auto-Generated catheter model

The physics-related cylindrical structure (never shown in the scene but used only in the calculations) and the **deformability** is computed by the **BeamFEMForceField** element with the relative elastic variables mentioned above (more details are decribed in the next subsection).

The collision-related cylindrical structure is built in a subnode of the catheter object using a **Cylinder-GridTopology** and a **BeamLinearMapping** to physically project any deformation and transformation of the main object. Since this structure has a variable [nx,ny,nz] that define how many points should be used in all directions to build the cylinder mesh and topology, we made sure that it will always have the same exact nz points to guarantee a 1:1 mapping with the real mechanical object defined before. This model (Figure 6)is used to check for collisions with other objects, in fact increasing the [nx,ny,nz] values will lead to an higher resolution and precision, but a mismatch in terms of total shape and deformation points.

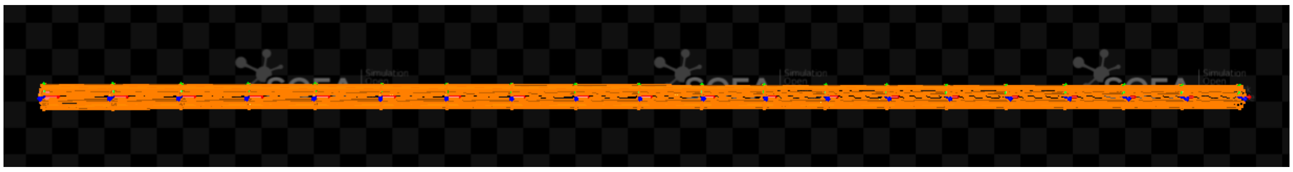


Figure 6: Collision Catheter model

The visual model (Figure 7)is just a sub-node of the collision sub-node with a 1:1 mapping of the collision model and a solid material definition.

Figure 7: Visual Catheter model

2.4 Catheter deformability with FEM force field

In this section we will describe how SOFA handles forces inside the simulation, through out the use of force fields. Since we will focus on the catheter deformability, those will be **internal forces** resulting by the effect of the soft body mechanics (elasticity and plasticity).

To begin with, both internal and external forces are the result of the **conservation of the linear momentum**. In order to build up the formula, you start with the Newton's second law:

$$\frac{d\boldsymbol{p}}{dt} = f$$

$$\frac{d\boldsymbol{p}}{dt} = f_{\text{vol}} + f_{\text{surf}}$$

where f_{surf} corresponds to the integration of traction forces and f_{vol} corresponds to the body forces. Then by following the Cauchy's law and Gauss's theorem, the conservation of linear momentum in the strong (or generalized) form can be written as:

$$\frac{D}{Dt}\int_\Omega \rho \boldsymbol{v} d\Omega = \int_\Omega \rho \boldsymbol{b} d\Omega \int_\Gamma \boldsymbol{t} d\Gamma$$

$$\int_\Omega \rho \frac{D\boldsymbol{v}}{Dt} d\Omega = \int_\Omega \rho \boldsymbol{b} d\Omega \int_\Gamma n \cdot \boldsymbol{\sigma} d\Gamma$$

$$\int_\Omega \rho \frac{D\boldsymbol{v}}{Dt} d\Omega = \int_\Omega \rho \boldsymbol{b} d\Omega \int_\Omega \frac{\partial \sigma_{ij}}{\partial xi} d\Omega$$

$$\rho \dot{v} = \rho \boldsymbol{b} + \nabla \cdot \sigma$$

where σ is the Cauchy's stress tensor, b is the density of external forces (force per mass unit), ρ is the mass density and $\dot{v}$ is the acceleration of the body and d is the acceleration of the body.

Continuing, in order to find a solution over the domain of the simulation, you need to integrate the momentum equation. Since no exact integration can be performed on a random domain Ω, the choice is to rely on the Finite Element Method (FEM), which acts on the subdivision of the domain into sub-domains (the finite elements), such as quads, triangles, tetrahedra, hexahedra, etc., so that the integration will be performed as the sum of the integrals over finite elements.

$$\int_\Omega \psi_j \nabla \cdot \boldsymbol{\sigma} d\Omega = \int_\Gamma \psi_j \cdot \boldsymbol{t} d\Gamma - \int_\Omega \nabla(\psi_j) : \boldsymbol{\sigma} d\Omega$$

At the end the FEM takes advantage of the simple shape of these finite elements to compute the integrals.

As previously mentioned, for our catheter we chose the **BeamFEMForceField**, which intuitively approximates the shape for the integration to a beam, in order to compute the integral of the momentum. This choice is very convenient because the catheter reflects a beam both in shape and behaviour. The parameters to be considered are:

- *FEM radius*: it is the base radius of the beam integrating surface.
- *Stiffness*
- *Young Modulus*: quantifies the relationship between tensile/compressive stress σ (force per unit area) and axial strain ε (proportional deformation) in the linear elastic region of a material and is determined using the formula:
$$E = \frac{\sigma}{\varepsilon}$$

All those parameters will concur on the computation of the internal forces and consequently the deformability.

2.5 Other models

To conduct more experiments in different settings, multiple meshes representing different types of vessels have been taken in to consideration. Specifically, we focused on some models from previous years projects, presenting progressively more articulated tracks for the catheter: one straight (Figure 8), one with a curvilinear shape (Figure 9) and one with a shrinkage (Figure 10).

In addition, we found also more realistic meshes representing the carotids and the mid cephalic vein (Figure: (11, 12), but unfortunately due to the complexity of those meshes and of the computation of the constraining forces, the simulation looses a lot of resolution (sometimes crashing after few seconds).

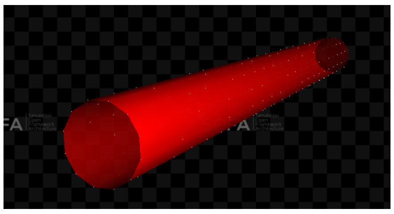


Figure 8: Vessel model 1.

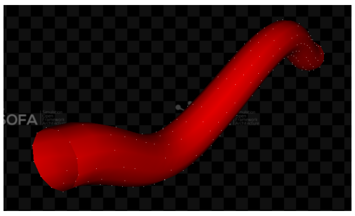


Figure 9: Vessel model 2.

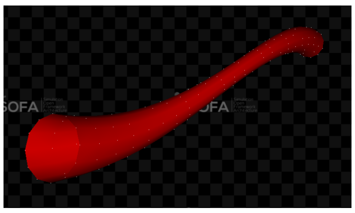


Figure 10: Vessel model 3.

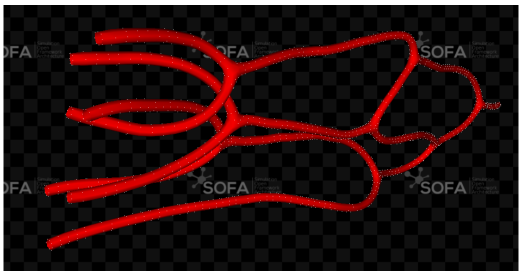


Figure 11: Mid cephalic veins model.

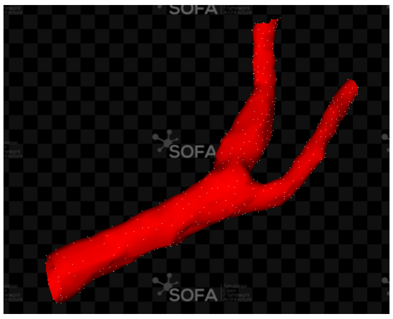


Figure 12: Carotids veins model.

In conclusion, we uploaded the models with MeshOBJLoader component and then we set the components used for collision detection and reaction, namely TriangleCollisionModel, LineCollisionModel and PointCollisionModel with contactStiffness set to 5. This value was chosen to obtain a realistic collision with the catheter while still maintaining stability (the models must not intersect each other). Finally, as for the model of the catheter, we applied the EulerImplicitSolver with the same parameters set at 0.1 and the CGLinearSolver with 25 iterations and 10^{-9} for tolerance and threshold.

2.6 Geomagic

A haptic device is a motorised device that applies force feedback on the user's hand, allowing them to feel virtual objects and produce true-to-life touch sensations as the user manipulates on-screen 3D objects.
Specifically, we adopted the Geomagic Touch device (Figure 13), which is a mid-range professional haptic device offering:

- 6-degree-of-freedom positional control in the workspace, by the use of the stylus
- 6-degree-of-freedom positional sensing
- Compact force feedback workspace with 3-degree-of-freedom force feedback
- Supports OpenHaptics toolkit

Figure 13: Geomagic Touch

Among the many features, the OpenHaptics toolkit allows the easy manipulation of many parameters from the Geomagic, including the extraction of both position and velocity with the functions *HD_CURRENT_POSITION* and *HD_CURRENT_VELOCITY* in the Cartesian space and with the functions *HD_CURRENT_GIMBAL_ANGLES*, *HD_CURRENT_JOINT_ANGLES* and *HD_CURRENT_ANGULAR_VELOCITY* in the Joint space (both for the firts three actuated joins and the last three passive ones), as well as the imposition of forces aimed to map kinestetic sensations to the user. In addition, the stilus presents two buttons giving 4 possible combinations which can be extracted with the functions *HD_CURRENT_BUTTONS*.

3 Technical aspects

3.1 Controller and catheter motion

In order to move the catheter by acting only on the base, a new mechanical object has been created, called **Gripper**, which is not subject to constraint forces or gravity and its movement is **one-way projected** to the catheter through an **AttachConstraint** node. This allows us to **accurately simulate an end-effector** of a robot and above all to locate its frame position in reference to any point of the catheter. We have adopted a Python plugin named *SPLIB3* which implements all mathematical objects required for the **roto-translation** of the Gripper node.

Regarding the teleoperation, the python program receives information of absolute positions from the haptic interface, among which we use: **Cartesian position of the end-effector** for the translation and the angles of the **joints J4, J5, J6** (Figure: 15) for the rotation.

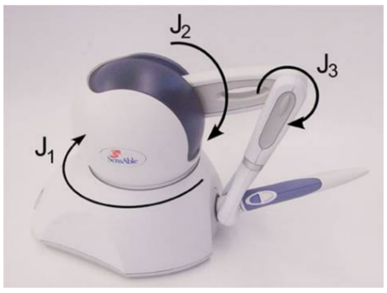


Figure 14: Actuated joins.

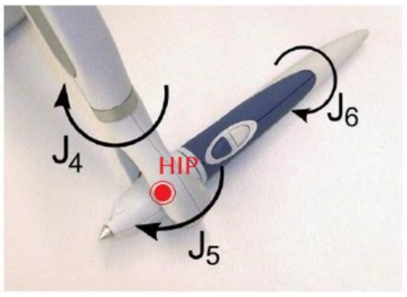


Figure 15: Passive joins.

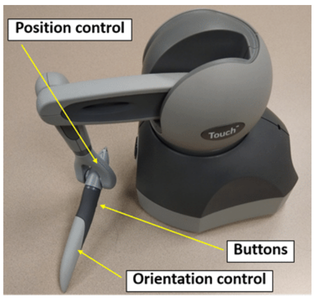


Figure 16: Commands.

Then, to transfer movement, each frame computes the **delta** as the difference in position and angles applied to the base of the catheter. This type of transfer ensures a **decoupling** from the absolute position of the Geomagic end-effector, which has a limited workspace, and allows to implement a virtual clutch system to be able to reposition it freely without affecting the position of the catheter. In detail, during the simulation, to enable motion transmission it is **necessary to press and hold button 1** of the Geomagic Touch Stylus (Figure: 16). The decoupling also made possible a and **decoupled scaling** of each component of each force/position/angle that can be easily tweaked by changing the `offset_positions`,`rotations_scale`,`translations_scale` 3D arrays and `force_scale` scalar to improve the end-user sensing and moving capabilities without introducing artifacts. Speaking of reference systems, we noticed a mismatch between that of the haptic device and the catheter so we implemented the following axis-angle mappings:

- -Z Axis of E-E `->` +X axis of Sofa (direction of catheter insertion)
- +Y Axis of E-E `->` +Y axis of Sofa
- +X Axis of E-E `->` +Z axis of Sofa
- J6 angle `->` Rotation on X axis of Sofa (direction of catheter insertion)
- J4 angle `->` Rotation on Y axis of Sofa
- J5 angle `->` Rotation on Z axis of Sofa

Going into the details, we observe that thanks to the physics solved by Sofa Framework, the catheter is able to follow the movement of the Gripper and to make every point react in a realistic way to variations in position and rotation of it.

Initially, before the teleoperation implementation, through Python the keypress events were captured and the following operations were made available:

- CTRL + 9: forward movement
- CTRL + 7: backward movement
- CTRL + 6: clockwise rotation
- CTRL + 4: counterclockwise rotation

Finally, there is a variable inside the python code called `AUTOMATIC_MOVEMENT` that can be set to True to enable an automatic forward X-axis movement of the catheter for debugging and automation purposes.

3.2 Forces extraction

The extraction of the forces was one of the most challenging obstacles we had to face, since Sofa Framework does not provide directly the computed forces and there are many data fields to be checked, presenting confusing names (eg: *"Forces"* does not imply the forces binding received by the object).
In order to enable the computation of the constraint forces, the object *"GenericConstraintSolver"* was added to the scene. This node implements a Lagrange Multipliers approach and a single linearization by time step for computing the constraint forces of the mechanical objects in the scene. From the physical system to solve, the constraint problem can be expressed with the linear system as:

$$\left(\mathbf{M} + dt\frac{\partial f}{\partial \dot{x}} + dt^2\frac{\partial f}{\partial x}\right)\Delta v = -dt(f + dt\frac{\partial f}{\partial x}v - \mathbf{H}^T\lambda)$$

This can be simplified to:

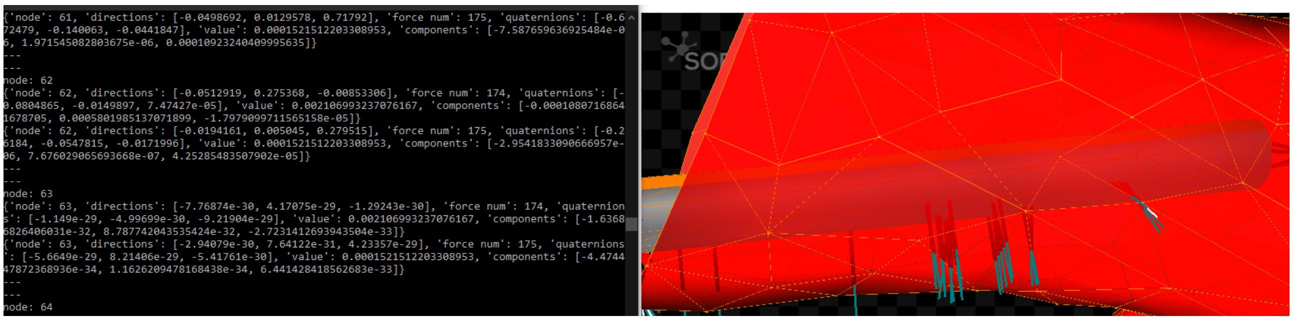
$$\mathbf{A}\Delta v = b + dt\mathbf{H}^T\lambda$$

where $\mathbf{H}^T\lambda$ is the vector of constraint forces contribution with $\mathbf{H}$ the matrix containing the constraint directions and λ are the so-called Lagrange multipliers. While the system is solving this problem, the H matrix is made available in the **"Constraints"** field of each mechanical object, however the data format needs a lot of rework because each line has the following structure:

```
EachLineOfConstraintsData {
    int forceID; //id of the force
    int numOfNodes; //how many nodes have this exact force applied
    int node1ID; //id of the first node that has this force
    float directionX; //direction in the global X axis
    float directionY; //direction in the global Y axis
    float directionZ; //direction in the global Z axis
    float quatX; //X quaternion of the force
    float quatY; //Y quaternion of the force
    float quatZ; //Z quaternion of the force
    int node2ID; //id of the second node that has this force
    float directionX;
    float directionY;
    //and so on
} ;
```

The constraint directions are also computed in the collision sub-node, but in that case the **quaternion fields are missing**. Then, in order to obtain the λ Lagrange coefficients or more simply the actual values of the forces, it is necessary to enable a particular option of the *GenericConstraintSolver* called **computeConstraintForces** and access the data field called **constraintForces**, which has an array structure, where it is possible to extract the value of the force by accessing the position corresponding to its id found earlier in Constraint data structure in the mechanical object.

This process is conducted by our code automatically each frame. The output of our algorithm is a ordered list where each position contains all forces applied in a particular node, and every force entry is characterized by its id, value, direction and quaternions. By altering the variable `USE_COLLISION_MODEL_FOR_FORCE_EVAL` our system is able to switch between printing out the constraint forces applied on the Mechanical object of the Physical model or the forces applied on the Collision model.



```
{'node': 61, 'directions': [-0.0498692, 0.0129578, 0.71792], 'force num': 175, 'quaternions': [-0.6
72479, -0.140063, -0.0441847], 'value': 0.000152151203308953, 'components': [-7.587659636925484e-0
6, 1.971545082803675e-06, 0.00010923240409995635]}
---
---
node: 62
{'node': 62, 'directions': [-0.0512919, 0.275368, -0.00853306], 'force num': 174, 'quaternions': [-
0.0804865, -0.0149897, 7.47427e-05], 'value': 0.002106993237076167, 'components': [-0.0001080716864
1678705, 0.0005801985137071899, -1.7979099711565158e-05]}
{'node': 62, 'directions': [-0.0194161, 0.005045, 0.279515], 'force num': 175, 'quaternions': [-0.2
6184, -0.0547815, -0.0171996], 'value': 0.000152151203308953, 'components': [-2.9541833090666957e-
06, 7.676029065693668e-07, 4.252854835079O2e-05]}
---
---
node: 63
{'node': 63, 'directions': [-7.76874e-30, 4.17075e-29, -1.29243e-30], 'force num': 174, 'quaternion
s': [-1.149e-29, -4.99699e-30, -9.21904e-29], 'value': 0.002106993237076167, 'components': [-1.6368
6826406031e-32, 8.787742043535424e-32, -2.7231412693943504e-33]}
{'node': 63, 'directions': [-2.94079e-30, 7.64122e-31, 4.23357e-29], 'force num': 175, 'quaternions
': [-5.6649e-29, 8.21406e-29, -5.41761e-30], 'value': 0.000152151203308953, 'components': [-4.4744
47872368936e-34, 1.1626209478168438e-34, 6.441428418562683e-33]}
---
---
node: 64
```

Figure 17: Constraint forces acting on the catheter.

3.3 Force Units

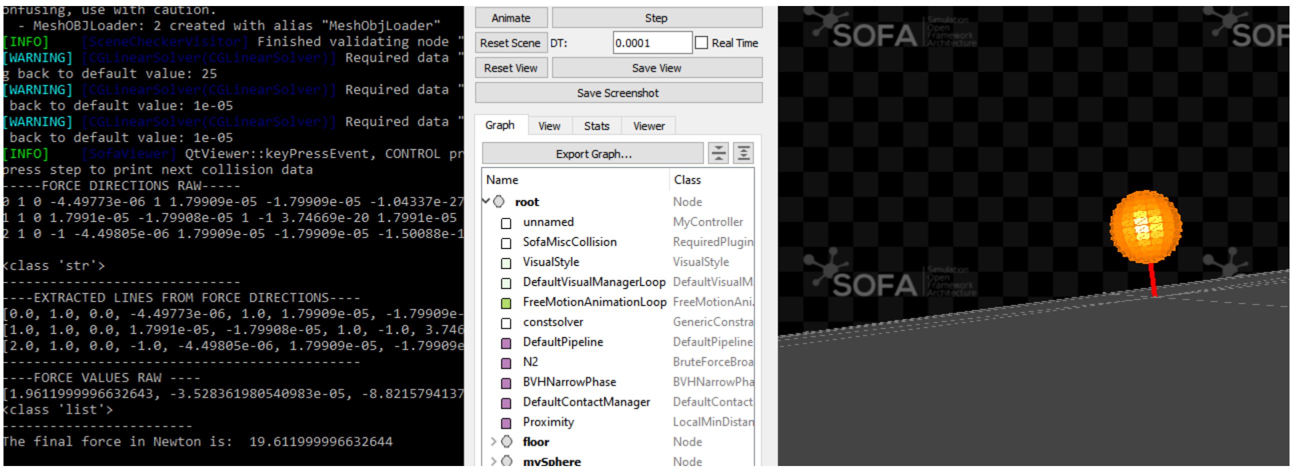
As mentioned before, no unit can be specified inside sofa, so every single configuration value must be carefully chosen by the User. In our case, we found out that using mm, kg, seconds, our forces had these units:

$$\frac{kg * mm}{s^2}$$

Also, we found out with our *sphere falling force test* the acceleration applied on any instant on our objects was dependent on the **Time Step**, resulting in this equation to be hold true for any value:

$$F = m * a * dt$$

With that being said, our force values need to be **multiplied by 0.001** to match with the Newton scale, and then **divided by the dt** as in Figure 18:



```
onfusing, use with caution.
  - MeshOBJLoader: 2 created with alias "MeshObjLoader"
[INFO]    [SceneCheckerVisitor] Finished validating node "
[WARNING] [CGLinearSolver(CGLinearSolver)] Required data "
g back to default value: 25
[WARNING] [CGLinearSolver(CGLinearSolver)] Required data "
 back to default value: 1e-05
[WARNING] [CGLinearSolver(CGLinearSolver)] Required data "
 back to default value: 1e-05
[INFO]    [SofaViewer] QtViewer::keyPressEvent, CONTROL pr
press step to print next collision data
-----FORCE DIRECTIONS RAW-----
0 1 0 -4.49773e-06 1 1.79909e-05 -1.79909e-05 -1.04337e-27
1 1 0 1.7991e-05 -1.79908e-05 1 -1 3.74669e-20 1.7991e-05
2 1 0 -1 -4.49805e-06 1.79909e-05 -1.79909e-05 -1.50088e-1

<class 'str'>
------------------------------
----EXTRACTED LINES FROM FORCE DIRECTIONS----
[0.0, 1.0, 0.0, -4.49773e-06, 1.0, 1.79909e-05, -1.79909e-
[1.0, 1.0, 0.0, 1.7991e-05, -1.79908e-05, 1.0, -1.0, 3.746
[2.0, 1.0, 0.0, -1.0, -4.49805e-06, 1.79909e-05, -1.79909e
---------------------------------------------
----FORCE VALUES RAW ----
[1.9611999996632643, -3.528361980540983e-05, -8.8215794137
<class 'list'>
------------------------
The final force in Newton is:  19.611999996632644
```

DT: 0.0001

Name	Class
root	Node
unnamed	MyController
SofaMiscCollision	RequiredPlugin
VisualStyle	VisualStyle
DefaultVisualManagerLoop	DefaultVisualM
FreeMotionAnimationLoop	FreeMotionAni
constsolver	GenericConstra
DefaultPipeline	DefaultPipeline
N2	BruteForceBroa
BVHNarrowPhase	BVHNarrowPha
DefaultContactManager	DefaultContact
Proximity	LocalMinDistan
floor	Node
mySphere	Node

Figure 18: Experiment with $mass = 2kg$ and $a = 9806mm/s^2$, giving an expected force of 19.62 N

3.4 Handling of the forces

As the forces have been extracted from the collision model and converted into Newton, it is necessary to handle them in order to reduce them into one single vector (to map into the Geomagic). This process is managed with the costumed function ***handle_forces***, which extract the constraining forces associated to each node, subdivide them into clusters of neighbouring forces, approximate each cluster to one single vector and finally returning a final force (depending on the chosen mode):

$$handle\_forces(force\_dict, force\_threshold, grouping\_model, mode, verbosity)$$

where:

- $force\_dict$ = dictionary of all forces extracted from the $GenericConstraintSolver$.
- $force\_threshold$ = threshold imposed to the module of each force.
- $grouping\_model \in [0, 1]$ = model of approximation of the clusters of neighbouring forces into one vector:
  - if put to 0, the function computes the average force weighted with a Gaussian centre in the **centre of gravity**,
  - if put to 1, the function computes the average force weighted with a Gaussian is centered in the position of the component with the **highest absolute value.**
- $mode \in [0 : 3]$ = mapping mode of the final force as kinestetic perception to the Geomagic, as a tuple with application node and components:
  - if put to **0**, the function returns **all approximated forces**,
  - if put to **1**, the function returns the **force action on the last node** (tip of the tool),
  - if put to **2**, the function returns the approximated force with the **highest absolute value**.
  - if put to **3**, the function returns the **sum of all approximated forces**.

- *verbosity* [*boolean*] = the function prints all approximated forces in the terminal.

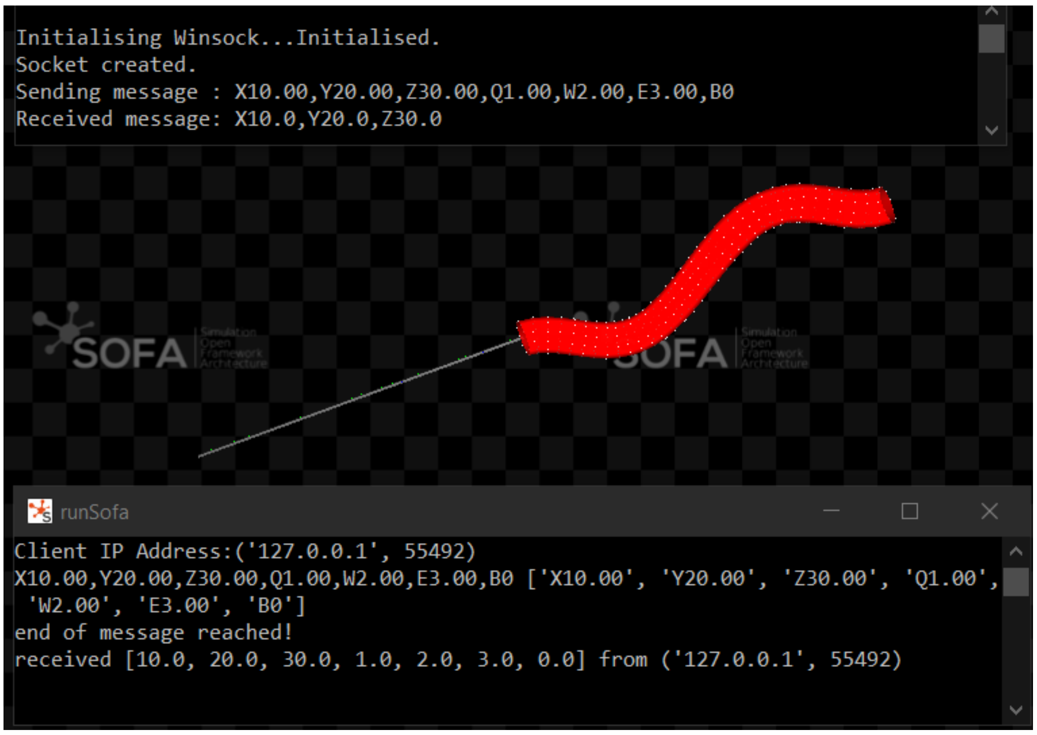
In addition, the clusterization process have been managed subdividing the ordered forces acting on each node depending on the number of nodes without any forces acting. This quantity is specified by the parameter *clusterization_factor*.

In conclusion, forces are sent to the Geomagic, which plots them into the stylus as kinestetic sensations. Naturally, it is possible to switch both force model and mode during the simulation, respectively pressing the both buttons at the same time and the second button.

3.5 Geomagic connection

Many obstacles have been encountered to enable bilateral communication between Sofa and the Geomagic Touch. The low-level OpenHaptics library, which takes care of sending and retrieving information from the haptic device, works in the C programming language, while our Sofa script works in Python, so it was necessary to write a C program capable of communicating with Python through some kind of inter-process-communication. After several iterations, it was established that the simplest and most effective technique for exchanging information between two processes is a **local UDP server**, and a message sending / receiving protocol was developed and structured as such:

1. Python script is the UDP **Server**, C program is an UDP **Client**
2. C program pools the Geomagic Touch and fetches the necessary information, and sends it to the server in the form of an ASCII packet: `X{EE_X},Y{EE_Y},Z{EE_Z},Q{J4},W{J5},E{J6},B{BUTTON_STATE}`
3. Python server checks the available received packets in a non-blocking way, and if it receives it, it processes its data and sends a packet of the type: `X{FORCE_X},Y{FORCE_Y},Z{FORCE_Z}` containing the components of the found constraining force
4. C program waits for the reception of a packet in a blocking way to ensure time synchronization, and sends the received force values to the Geomagic Touch via the OpenHaptics HDAPI calls.
5. C program loops back to 2. infinitely until closed or interrupted
6. Both client and server have a check on the integrity on the packets to ensure data reliability.



```
Initialising Winsock...Initialised.
Socket created.
Sending message : X10.00,Y20.00,Z30.00,Q1.00,W2.00,E3.00,B0
Received message: X10.0,Y20.0,Z30.0
```

SOFA
Simulation Open Framework Architecture
SOFA
Simulation Open Framework Architecture
runSofa

```
Client IP Address:('127.0.0.1', 55492)
X10.00,Y20.00,Z30.00,Q1.00,W2.00,E3.00,B0 ['X10.00', 'Y20.00', 'Z30.00', 'Q1.00',
 'W2.00', 'E3.00', 'B0']
end of message reached!
received [10.0, 20.0, 30.0, 1.0, 2.0, 3.0, 0.0] from ('127.0.0.1', 55492)
```

Figure 19: Exchange of packages between the C program and SOFA.

4 Experiments & Conclusions

4.1 Experiments settings

We have conducted a set of experiments using the same catheter with different cavity models. The aim is to use the framework simulating different scenarios to test its behaviour in presents of multiple interaction forces and the effectiveness of the bilateral connection with the Geomagic Touch. In addition, for each simulation we tried all possible combination of grouping model and mode, in order to have a feedback on which is the more effective one.

Considering the parameters of the simulations, we built the catheter model with the following characteristics:

- $YoungModulus = 10^4 MPa$ and $Density = 1550 kg/m^3$ to be consistent with **silicone** material. Of course, since the catheter is actually a tube while the model is in fact a full cylinder, the final mass is a combination of silicone and air materials.
- $Length = 200mm$, $ExternalDiameter = 2mm$ and $internalDiameter = 1.6mm$, as the classic **MID-LINE model**
- $PoissonRatio = 0.49$ to minimize the compression along the x-axis.

Other important parameters are:

- $ForceScaling = 10$. We tuned empirically this parameter to make the interaction forces clear to the user, without affecting the precision of the user.
- $RotationScaling = [1/6, 1, 1]$, respectively around the rotation axis x, y and z.
- $TralationScaling = [1/20, 0, 0]$ respectively around the rotation axis x, y and z. The aim was to cancel both motion components along the y and z axis and heavily scale down the motion along the x axis, in order to be able to make more precise movement by the user along the only motion direction.

For each example, all commands have been tested and the catheter (comprehensively of translation and rotation of the grip, as well as the button commands) with all the different combinations of grouping models and modes.

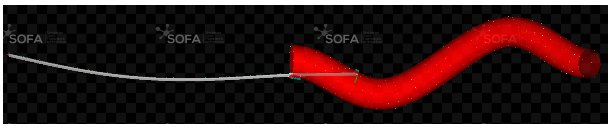


Figure 20: Mode 1 in with vessel model 2.

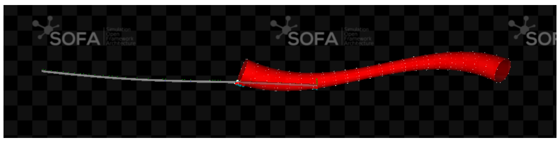


Figure 21: Mode 1 in with vessel model 3.

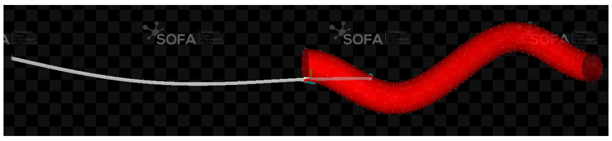


Figure 22: Mode 2 in with vessel model 2.

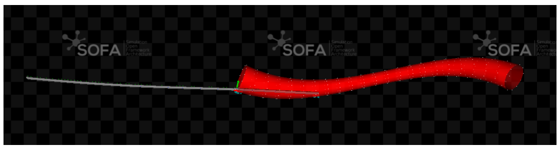


Figure 23: Mode 2 in with vessel model 3.

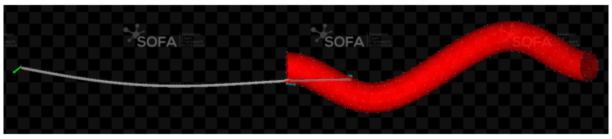


Figure 24: Mode 3 in with vessel model 2.

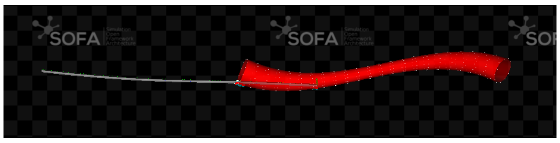


Figure 25: Mode 3 in with vessel model 3.

4.2 Force plots

During the simulations we also plot some of the registered forces that have been fed to the Geomagic touch as kinestetic sensations. Those vectors were ordered by node, as the script for force handling follows how SOFA has computed them. Consequently, we are going to show only the forces registered with mode 1 (interaction with the tip only) and mode 3 (sum of all approximated forces). The simulation considered is the one with the *artificialModel_2* and it will be conducted with constant speed along the x direction, no rotation and a $\delta t = 0.001$, along the whole body of the cavity model.

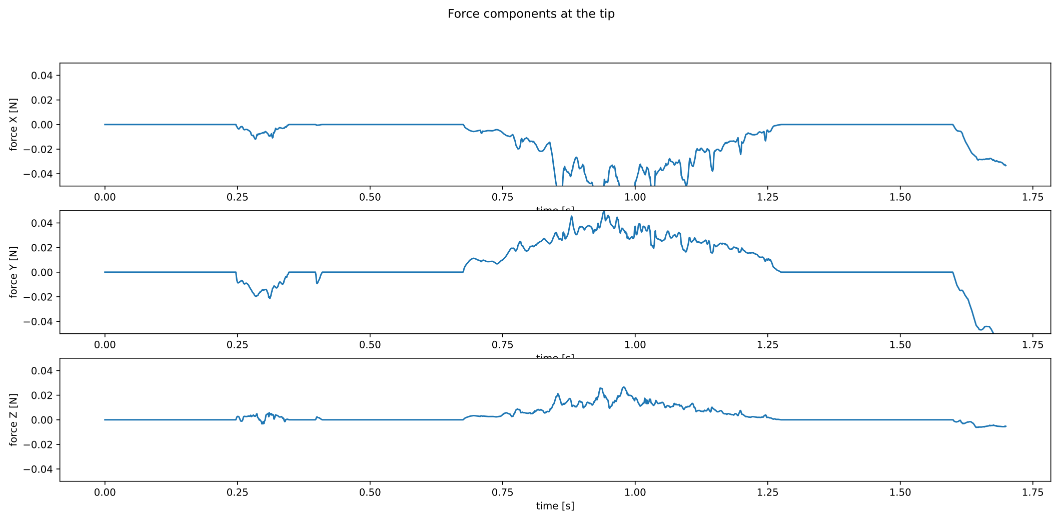


Figure 26: Forces plot with mode 1.

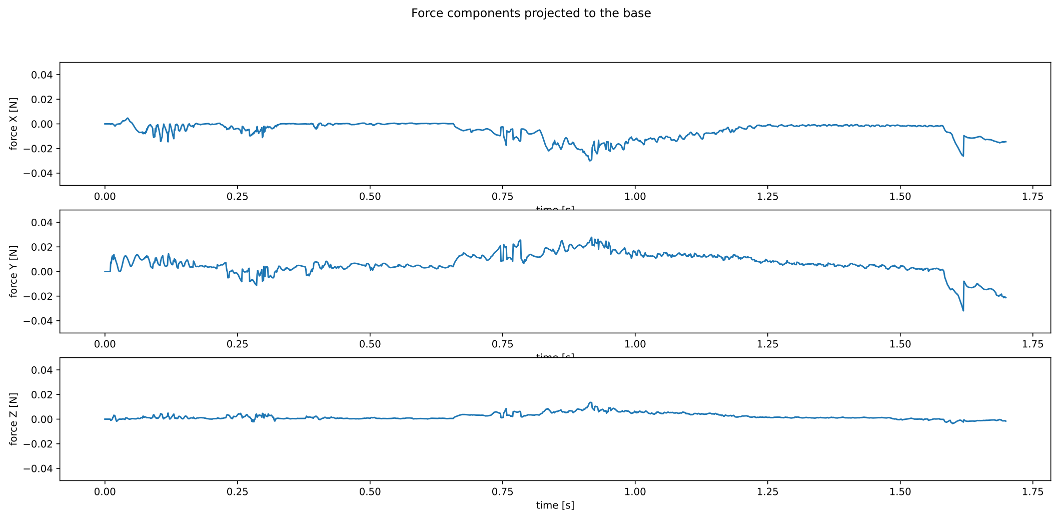


Figure 27: Forces plot with mode 3.

Considering the plot with *mode 1*, it is evident that forces are registered only in case the tip directly impacts the model walls. In particular, there are three considerable time intervals:

1. First interaction (Figure 28): the catheter hits the wall of the mesh, generating an interaction force of the order of 0.01 Newton, predominantly directed along the negative y-axis.

2. Second interaction (Figure 29): the catheter hits the wall of the mesh, generating an interaction force of the order of 0.01 Newton, predominantly directed along the positive y-axis.

3. Third interaction (Figure 30): the catheter hits the wall of the mesh, generating an interaction force of the order of $4 * 0.01$ Newton, predominantly directed along the negative y-axis and slightly inclined towards the negative x-axis. To be noticed, in this phase the catheter has already registered more that 230 interaction and the simulation is already running around 5.5 FPS.

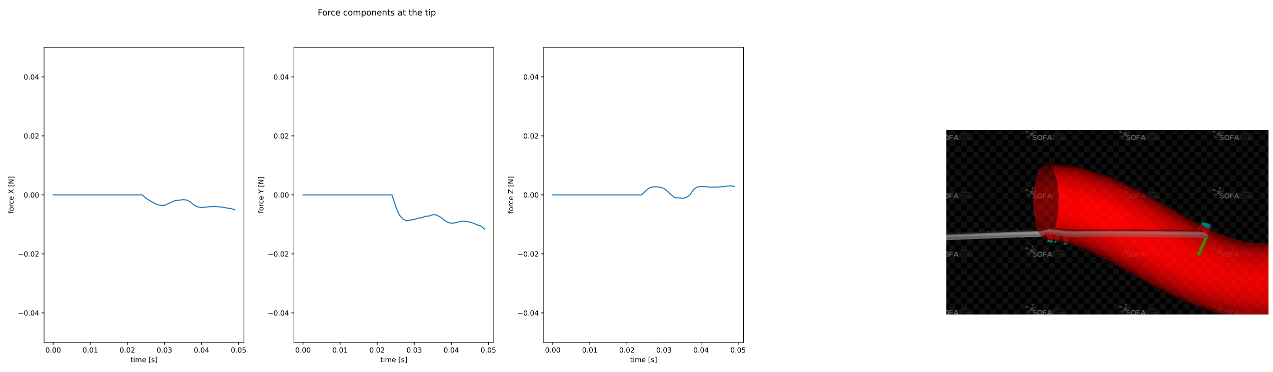


Figure 28: First impact of the catheter into the mesh with its tip.

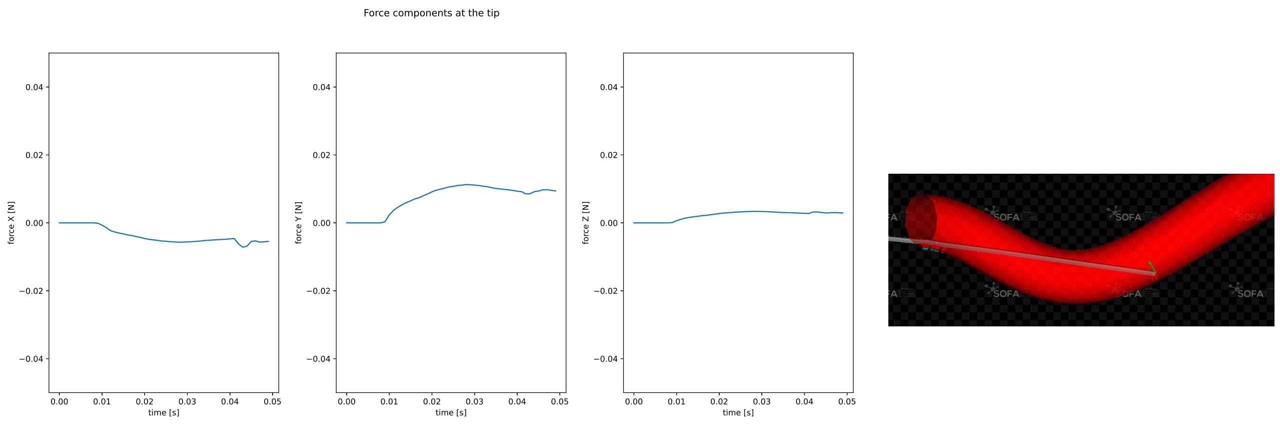


Figure 29: Second impact of the catheter into the mesh with its tip.

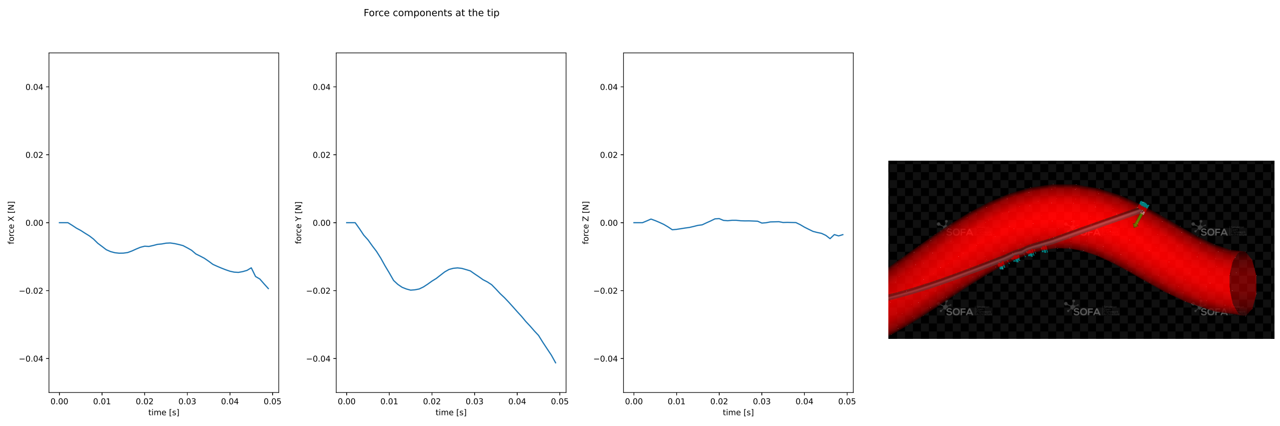


Figure 30: Third impact of the catheter into the mesh with its tip.

Regarding *mode 3*, the plot is much more noisy, since there is a constant component summing up as friction, due to the contact with the mouth of the vessel model. Nonetheless, in Figure 31 where both mode 3 (in blue) and mode 1 (in orange) plots are compared, it is quite clear that all three components follow a path similar to *mode 1*. This clearly suggests that the highest contribution to the final force is indeed from the tip of the tool, while the other approximated forces have a lower impact.

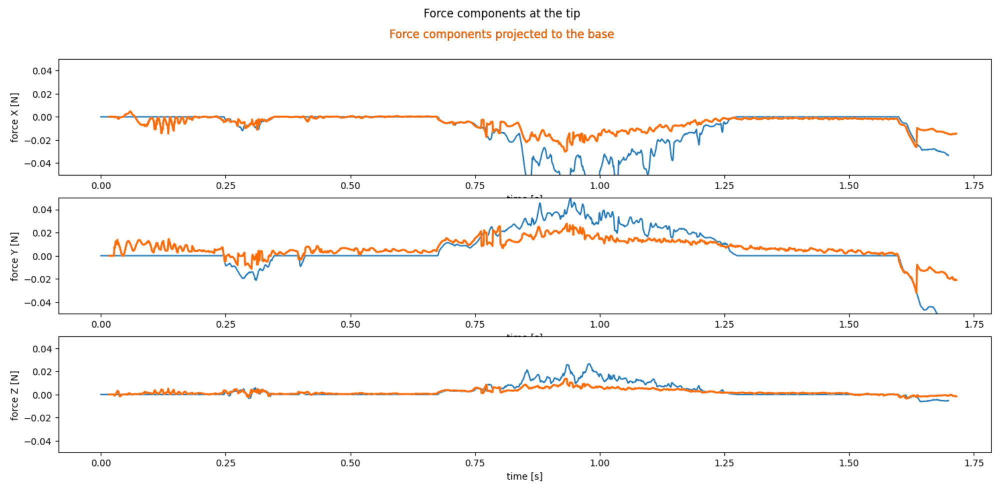


Figure 31: Comparison between the two plots.

4.3 Results

Overall, the **parameters of the catheter model** have given convincing results, in terms of appearance and flexibility. The length of the model is just enough to be effectively deformable with a mesh with 100 nodes (one node every 2 mm), while not heavily affecting the simulation resolution.

Regarding the three different **modes**, all simulations have been giving quite the same results in terms of user feedback. Specifically, the first one, that maps the forces from the tip, definitely appears effective and intuitive, even thought the force component originate in the tip and are mapped to the first node as kinestetic sensation, while the model is not rigid. Then, the second mode, which maps the highest force acting on the whole body of the catheter, remains quite confusing, even if an indicator (visible green vector) has been inserted into the simulation for this purpose. Finally, the third mode, which sums up all forces in the catheter, appears to be quite ineffective, since the resulting vector should map different reaction forces and the *openHaptics* library does not allow a decoupling of kinestetic sensations.

Considering the **grouping models**, no obvious differences emerged, except for those cases in which the tip of the catheter hit the cavity model by the tip with an angle. In those cases, the model generates a cluster of interaction forces around the tip, involving several nodes, with the highest located in the final node. Consequently, the first grouping mode considers the centre of the distribution as the last node of the model (the one node with the highest absolute value), while the other grouping model considers the centre of gravity. This locates the final force in completely different parts of the model and with slightly different absolute values.

4.4 Performances

Considering the SOFA simulator, the experiments have shown good results regarding the **catheter model**, which appears to be deformable under the action of external forces (gravity, interaction forces and friction) and controlled in position. In addition, due the structure of the simulation, both material and structure proprieties are easily editable, making the system adaptable to different types of catheters. However, the deformability of the model still lacks in terms of accuracy, due to the limitation of SOFA itself. Specifically, the number of nodes describing the topology has to deal with a trade-off between accuracy of the representation and definition of the simulation.

Furthermore, the **interactions between rigid objects and the deformable catheter** worked properly, even if we had to set in the *MinProximityIntersection* model high values of *alarm* and *contact distance* (respectively 1.5 mm and 1 mm) in order to give SOFA the time to make the collisions detectable. This is actually common in these kind of simulators, but still gives some margin of improvement, also considering that vessels are already small environments and by enlarging the contact distance you de-fact make artificially more difficult the catheter to slide inside.

Finally, the most critical aspect of the simulator regards the **computation of the interaction forces** by the *GenericConstraintSolver*, which is the only constraint solvers enabling the computation of contact forces at each collision. Consequently, it is also the far less optimized constraint solvers, making the simulation rapidly decreasing in performances when many collisions occurs. To make up for this problem and build a more fluid experience by the user, we had to decrease the dt to 0.001 seconds. Accordingly, the simulations decreased in FPS(frame-per-second) prior a complete crush of the simulation, just over 240 collisions detected and some unreal oscillatory movements of the catheter occurs.

Moreover, considering the **scripts for handling interaction forces**, results are difficult to evaluate in terms of accuracy. Nonetheless, the paradigm has been confirmed to be valid by the simulation *falling_sphere.py* in terms of magnitude, while the direction can be visually justified through the green indicator mapping in the simulation itself the final force.

In conclusion, considering the **connection between SOFA and OpenHaptics**, it has given great results in terms of latency and flexibility. Due to this results it can be considered a good solution, probably more effective than the *SofaGeomagic* plugin, which in this case was no use at all.

4.5 Final remarks & Future developments

One of the biggest obstacles that we encountered is represented by SOFA simulator itself, which is still in its early stages of development. Specifically, it results not optimized and the documentation is pretty much not existent, making hard for the developer to build more complex simulations and for the users to apply it in real case scenarios.

Regarding future implementations and improvements not every possibility has been exploited so far. For instance, with the *SofaCuda* plugin and a better hardware, it's possible to delegate most of the CPU heavy computations to the GPU, in this way the simulation would be able to run at an higher frame-rate, handle smaller tolerances for the collisions, improve the computation of the constraining forces and, last but not least, obtain a more truthful deformation of the FEM model.
From another point of view, the connection protocol between Sofa and C that we implemented from scratch is very easy to extend to additional haptic interfaces that could improve the interaction between the operator and the simulation. Lastly, a future opportunity could also be the connection between SOFA and CoppeliaSIM parallel simulations to better simulate the robot side and the behaviour of the "gripper node".

In conclusion, one of our most important letdowns has been the impossibility to extract the internal forces generated at every node (but in particular the base node). The problem has been that if in one hand SOFA can compute the contact forces and our script can efficiently extract them by fusing several data fields from the Constraint solver and the mechanical objects, on the other hand the simulator does not seem to explicitly make available the projection of the forces applied on each point of the mechanical object nor the forces due to deformation of the original structure.

5 References

1. SOFA Component: official website.
2. SOFA: official website.
3. https://github.com/SofaDefrost/STLIB