CentraleSupélec

Rapport Technique Sujet 1

Malo Chauvel/Rany Fouad/Alexis Chapelant/Carlos Martín de Argila Lorente/Matthias Orbach

Séquence Thématique - Enseignement d'intégration
Robotique médicale
Sujet 1 - Etude des efforts d'interaction au contact du patient ou du chirurgien

23/01/2023 - 27/01/2023

Résumé

Dans ce rapport technique, nous allons décrire la méthodologie que nous avons appliquée lors de la création de cette simulation. Pour ce faire, nous avons, après une montée en puissance de nos compétences lors de la première journée, implémenté le modèle de la simulation sur Simulink lors de la deuxième journée. Cela, nous a amené à exploiter les résultats et résoudre les erreurs dans le code de la simulation au cours de la troisième journée.

Table des matières

1 Contexte **4**

2 Démarche adoptée pour résoudre le problème posé **5**

3 Hypothèses (de modélisation, . . .) **6**

4 Description du modèle **7**

5 Description de simulations **9**

6 Analyse de résultats **12**

7 Conclusion (lien éventuel avec autres équipes) **14**

8 Annexe **15**

1 Contexte

La robotique médicale est née de la fusion de domaines qui aux premiers abords peuvent sembler bien éloigné. En effet, la médecine est née de la volonté de soigner l'homme. Longtemps impuissante face aux maladies comme les épidémies, la médecine se développe au XIXe siècle avec les débuts de l'épidémiologie. Des avancées médicamenteuses, technologiques et biologiques permettent une forte diminution des maladies et une augmentation globale du niveau de vie. Toutefois, la médecine garde de nombreuses voies de développement, et pour cela cherche des solutions dans d'autres sciences.

La robotique quant à elle, voit le jour avec l'apparition des premiers automates au XIXe siècle. Elle grandit avec l'apparition de plus en plus importante de capteurs de mesure, ce qui permet aux automates d'interagir avec le monde. Une autre science vient booster l'essor de la robotique : l'informatique. En effet, celle-ci permet une plus grande liberté d'action aux robots. Aujourd'hui, la robotique est partout et c'est ainsi qu'on la retrouve également dans le monde médical. La robotique médicale touche de nombreux domaines, de la chirurgie mini-invasive à la téléopération, en passant par les exosquelettes. On peut faire une distinction en trois types de robots médicaux : les robots médicaux d'assistance aux docteurs, les robots médicaux d'assistance aux patients et les robots médicaux d'assistance aux praticiens pour la réhabilitation.

Ce sujet est inscrit dans le domaine de la robotique mini-invasive. On cherche à ajouter au modèle pré-existant de simulation sur Matlab et Simulink, un module permettant la modélisation des efforts d'interaction entre le robot et son environnement que se soit avec le patient mais également avec le personnel médicale environnant.

2 Démarche adoptée pour résoudre le problème posé

Pour répondre à ce sujet, nous avons du adopté une démarche par phase, tirée de projets précédents comme les *coding week*, l'utilisation d'une démarche MVP (*Minimum Viable Product*) qui permet une amélioration continuelle du modèle, avec des étapes intermédiaires fonctionnelles.
La première phase, qui nous a demandée une journée à été de comprendre et d'appréhender l'ensemble des documents. Cela, nous a permis d'avoir une montée en compétence que ce soit dans la démarche d'ingénieur en robotique médicale, mais également dans les notations utilisées. En effet, l'utilisation de la convention "Denavit-Hartenberg modifiée" permet non seulement une description de la position relative de deux solides avec seulement quatre paramètres (au lieu de six), mais également l'utilisation d'une méthode numérique par récurrence. On utilise principalement cette convention dans le cas de la modélisation d'une chaîne ouverte avec uniquement des liaisons pivots et glissières, ce qui est le cas dans notre modélisation de robot. La phase un, s'est soldée par l'écriture de la jacobienne utile pour les calculs de simulation. La phase deux, nous a permis d'établir une implémentation d'un code de simulation, prenant en compte l'interaction avec le patient. On utilise avec le modèle de Voigt (cf. section 4), qui nous permet de modéliser les tissus moux comme la peau, les muscles ou les organes. Cette modélisation, c'est s'appuyer sur un travail bibliographique pour déterminer le modèle mais également les coefficients en fonction du type de tissus. Le livre de référence trouvé est alors *Physics of Human Body* écrit par Irving P. Herman. Nous nous sommes principalement intéressés à la partie "Mechanical Properties of the Body".

La phase trois se résume à l'interprétation des résultats alors obtenus par la simulation, ainsi qu'à la correction des problèmes décelés. L'objectif de cette phase étant de constituer les divers livrables demandés, on valide la simulation pour l'intégrer aux livrables, et l'on rédige le rapport technique et le support visuel de la présentation.

3 Hypothèses (de modélisation, . . .)

La modélisation se sépare en deux. D'un coté, on doit modéliser le robot, de l'autre on doit modéliser le patient. Pour obtenir ces modélisations nous avons du faire des hypothèses de modélisation. Ainsi, nous allons ici les expliquer, et montrer en quoi elles sont justifiées.

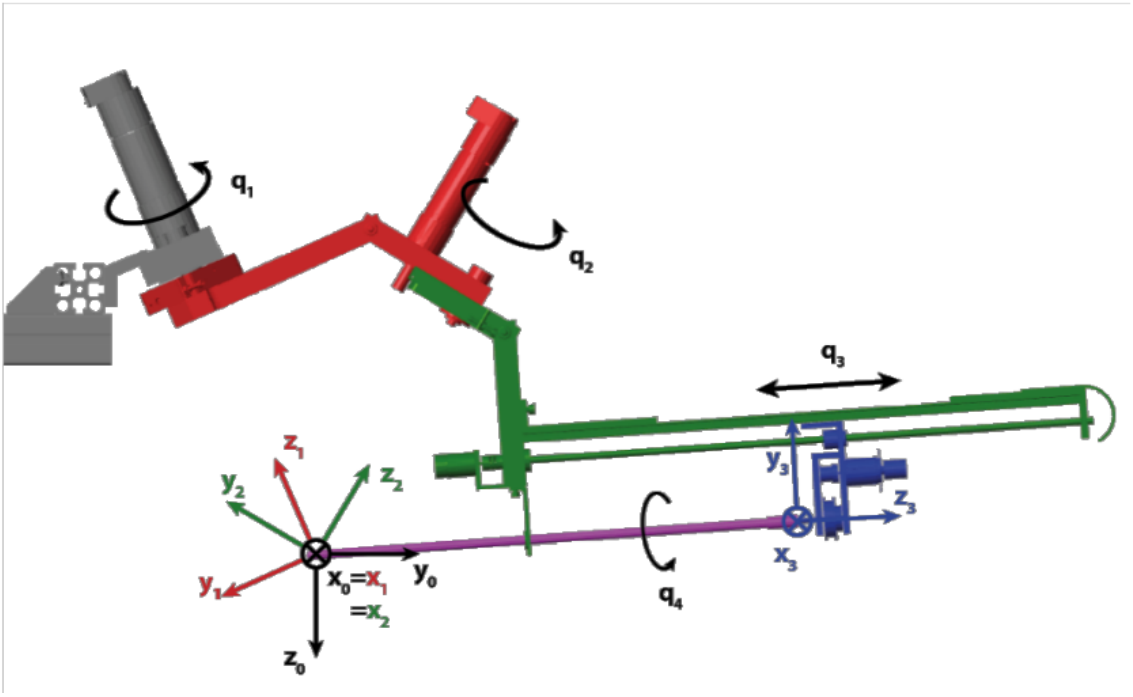

FIGURE 2 – Schéma de modélisation du robot

Pour la modélisation du robot, nous supposons que les moments ne sont pas transmis, en effet la prise en compte de moments localisés impliquerait une trop grande complexité de modélisation. Ainsi, l'étude s'intéresse uniquement à la prise en compte des contacts ponctuels avec le patient. Cela, nous induis la seconde hypothèse, lorsque l'on cherche à modéliser les contacts avec le patient. La rotation de l'axe 4 n'impliquant pas de forces de contact ponctuel mais uniquement des moments localisés, on peut supposer que cet axe est immobile (cf. Figure 2). La mise sous forme linéaire des efforts généralisés implique que tout les paramètres ne peuvent pas être identifiables indépendamment. On écrit alors les efforts généralisés sous la forme d'un problème des moindres carrés. Ainsi, on doit supposer connus les paramètres géométriques.

De l'autre part, le patient peut être modélisé par une couche de peau, suivie d'une couche de muscles et d'os. La peau et les muscles présentent des propriétés qui appartiennent à deux catégories de matériaux à la fois : la viscosité et l'élasticité. La viscosité traduit la capacité d'un matériau à dissiper l'énergie, à l'inverse de

l'élasticité, qui traduit la capacité d'un matériau à restituer de l'énergie, autrement dit l'élasticité traduit le retour à l'état initial d'un matériau après déformation. On retrouve ces deux aspects dans les tissus humains, elles reprennent bien la quasi totalité de leurs formes, mais subissent en même temps une légère déformation. Pour simplifier, la modélisation des chocs avec le corps médical prendra seulement en compte les chocs sur la partie 4 du robot, c'est-à-dire la tige du robot.

4 Description du modèle

Les hypothèses établit, nous pouvons maintenant écrire les équations de modélisation du mouvement. On obtient, en prenant en compte les hypothèses, les équations suivantes en appliquant la mécanique lagrangienne :
Soit le lagrangien L :

$$L = E - U \tag{1}$$

Avec E l'énergie cinétique et U l'énergie potentielle du système. Les équations du mouvement sont alors :

$$\Gamma_i = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} \tag{2}$$

Avec Γ_i la somme des forces généralisées non conservatives s'appliquant sur l'axe i.
Avec un robot réel, on peut écrire les efforts généralisés :

$$\vec{\Gamma} = \vec{\tau} - \vec{\tau_f} + \vec{\tau_{ext}} \tag{3}$$

Avec :

- $\vec{\tau}$ l'effort moteur au niveau articulaire
- $\vec{\tau_f}$ l'effort résistant dû aux frottements
- $\vec{\tau_{ext}}$ l'effort articulaire dû à un contact extérieur

Pour prendre en compte les efforts extérieurs, on établit l'équation suivante :

$$\vec{\tau_{ext}} = J(q)^T \vec{F_{ext}} \tag{4}$$

Avec $J(q)$ la matrice jacobienne du robot, et $\vec{F_{ext}}$ la force extérieur au point d'application. Après avoir écrit les matrices de transformation, on établit le Jacobien avec Matlab (cf. code 8)

Nous avons modélisé le patient comme une succession de couches de différentes natures dont nous pouvons faire varier l'épaisseur (on peut ainsi envisager une épaisseur de 0 cm pour l'os si nous souhaitons percer une zone sans os).

FIGURE 3 – Schéma de modélisation du robot

Il faut ensuite modéliser le comportement de chaque couche afin de connaître les forces appliquées par l'organe sur l'aiguille. Nous avons adopté le modèle de Kelvin-Voigt en considérant chaque couche comme l'association d'un ressort et d'un amortisseur. (Les constantes de ressort et coefficients d'amortissement des différentes couches sont spécifiés sur le schéma ci-dessus)

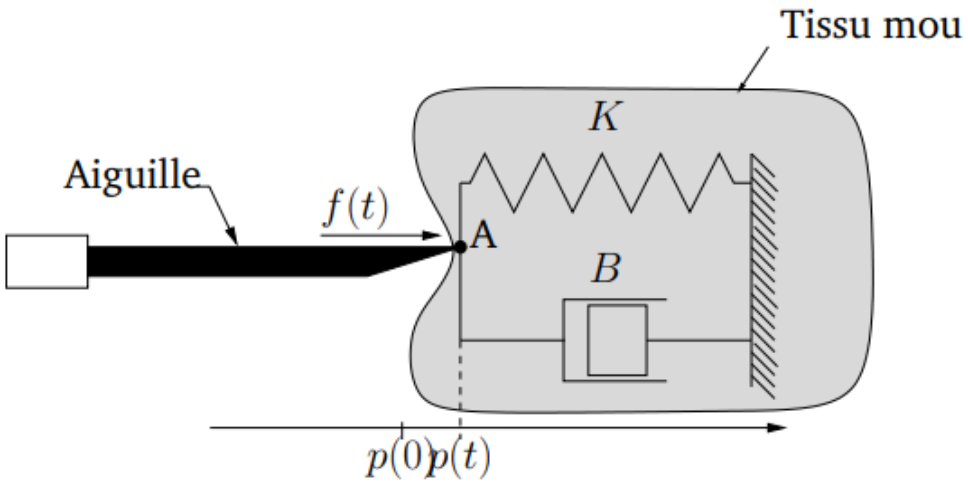

FIGURE 4 – Modele de Kelvin-Voigt

Sous ces hypothèses, on a une équation de la forme suivante :

$$\vec{F_{ext}} = -k\vec{x} - \alpha\vec{\dot{x}} \tag{5}$$

Le vecteur x est obtenu directement à partir de la dernière colonne de la Jacobienne (on prendra soin cependant de changer la coordonnée y afin d'avoir la profondeur de la couche que l'on traverse et non pas la distance à l'origine).

Le vecteur vitesse est quand à lui obtenu à partir de la Jacobienne et de la vitesse articulaire selon la formule suivante :

$$\vec{\dot{x}} = J(q)\vec{\dot{q}} \tag{6}$$

Une fois cette phase théorique passée, nous pouvons implémenter ce modèle sur Simulink.

5 Description de simulations

Pour réaliser la simulation nous avons utilisé la plateforme de modélisation de système Simulink. Pour cela, on a utilisé le fichier "ulb-simu-lagrange-BF-eleves-fr.slx" comme base et on a ajouté des blocs supplémentaires. Dans un premier temps nous avons codé un LiveScript pour trouver les matrices O_0O_n (qui est la position en coordonnées cartésiennes) et la *Jacobienne* en fonction de $q1, q2, q3$ et $q4$. Pour cela, on a réalisé le code décrit dans [8].

On va maintenant expliquer notre modélisation. Pour faciliter la compréhension, on a simplifié le modèle Simulink en le divisant en blocs distincts (et en enlevant quelques-uns).

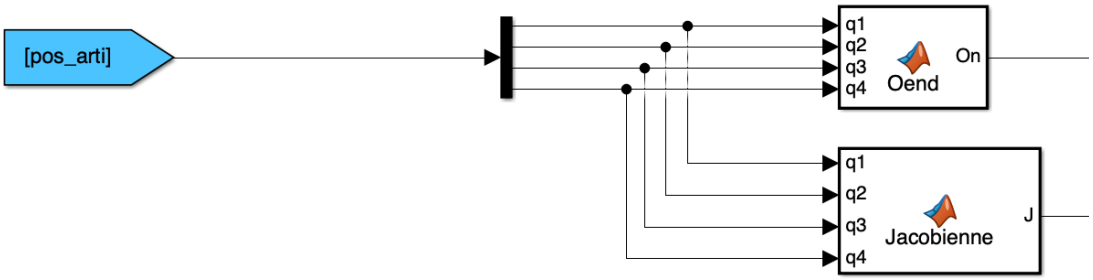

FIGURE 5 – Schéma bloc d'obtention de On et de la Jacobienne

Nous avons recopié les matrices O_0O_n [8] et de la *Jacobienne* [8] obtenues avec le LiveScript pour réaliser nos calculs. L'entrée de ces fonctions est le vecteur q et la sortie sont les matrices.

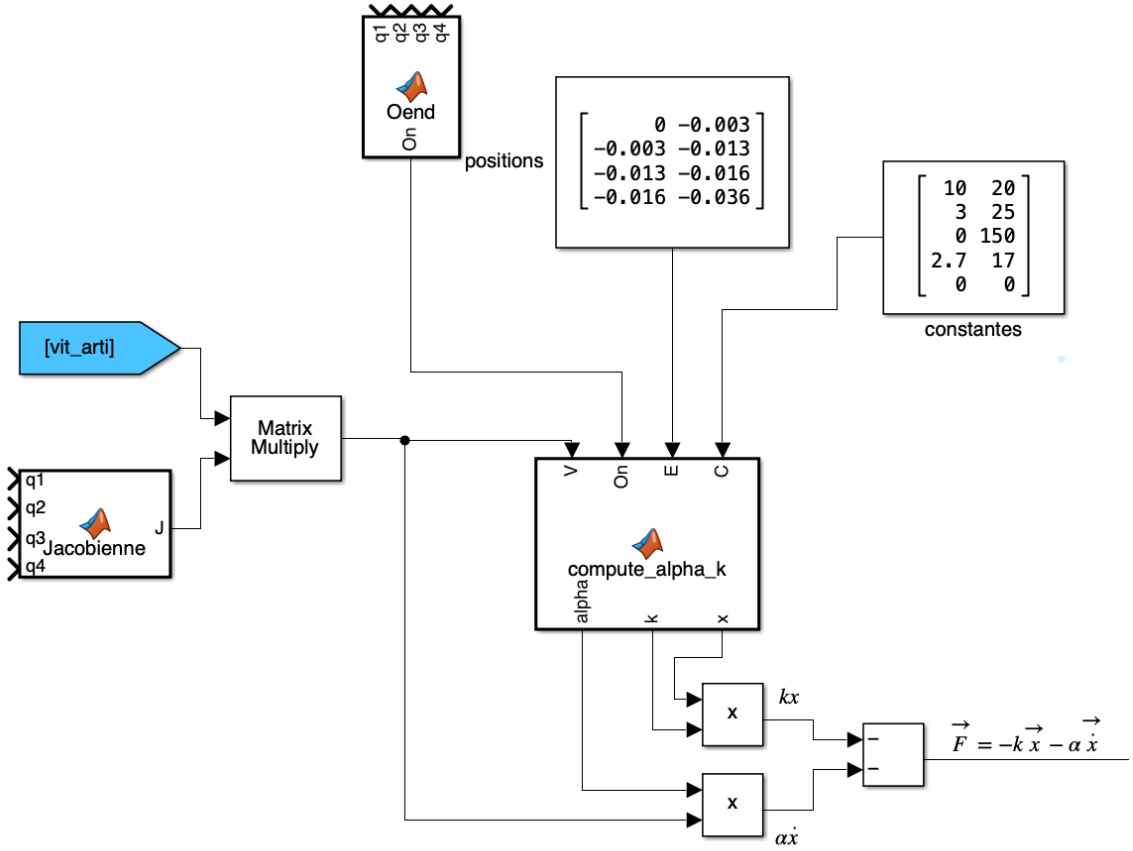

FIGURE 6 – Schéma bloc d'obtention des efforts extérieurs en cartésien

La deuxième partie de notre simulation consiste à calculer la force F en utilisant la formule suivante :

$$\vec{F} = -k\vec{x} - \alpha\dot{\vec{x}} \tag{7}$$

Dans cette partie, on utilise la matrice *Jacobienne* et O_0O_n, ainsi que les matrices *position* et *constantes*. La matrice *position* indique les positions occupées par chaque partie du corps du patient (dans l'axe y), tandis que la matrice *constantes* contient les coefficients k et *alpha* de chaque couche. Pour mieux comprendre l'utilisation de ces deux matrices, on a représenté les valeurs utilisées dans Simulink à l'aide du schéma [4].

La première colonne de la matrice *positions* indique la coordonnée y où débute chaque couche et la deuxième colonne indique la coordonnée où se termine la couche. On a fait ce choix plutôt que d'utiliser un vecteur avec les coordonnées de début car il peut y avoir des parties du corps qui n'ont pas une de couche (pas d'os par exemple). Dans ce cas, on pourrait mettre la position de début et de fin avec la même valeur et le modèle fonctionnerait quand même. Pour la matrice *constantes*, la première colonne indique la constante α et la deuxième colonne indique la constante k. À la

sortie de la fonction *compute_α_k* (voir annexe [8] pour voir le code), on avons k, α et x. La sortie x peut sembler triviale, mais nous a pris en compte ce qui est représenté dans le schéma. La valeur x ne prend en compte que la distance dans la couche, pas la distance totale depuis le repère. Finalement, on multiplie et soustrait pour obtenir $\vec{F}$.

Avec $\vec{F}$ on peut calculer $\tau_{ext} = J(q)^T F_{ext}$ avec les bloques suivants :

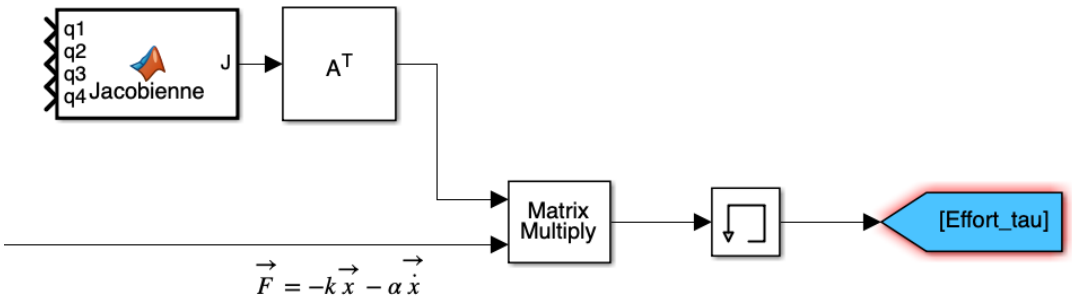

FIGURE 7 – Schéma de modélisation du robot

Pour modéliser un impact du robot avec l'effecteur, on a choisi d'utiliser une impulsion rectangulaire de 30N (voir annexe [8] qui représente l'impact. Ensuite, on multiplie cette impulsion par une version modifiée de la jacobienne (voir annexe [8]) en remplaçant $q3 + l4$ par $q3 + l4 - yop$, où yop représente la distance de l'impact par rapport à l'effecteur. Cela nous permet d'obtenir l'effort extérieur τ_{ext_op}.

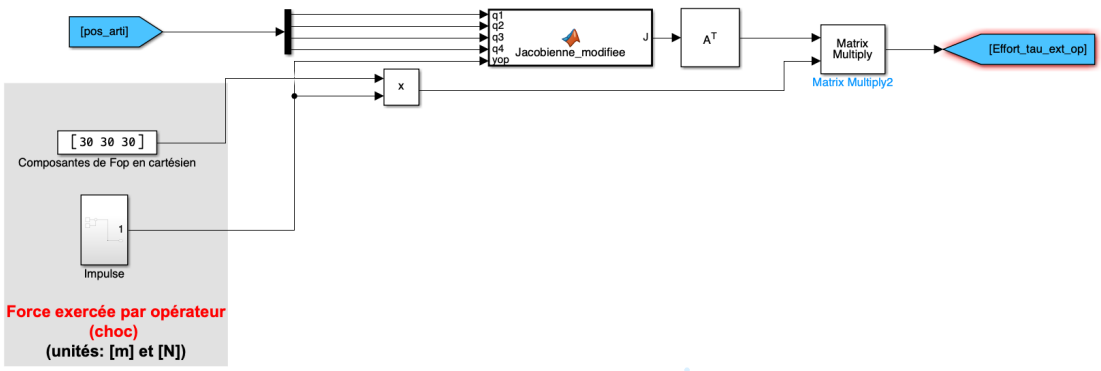

FIGURE 8 – Schéma bloc d'obtention des efforts τ

6 Analyse de résultats

Nous avons utilisé différents visualiseurs (scopes) pour observer les résultats sous forme de graphes. Dans un premier temps, la simulation sans bouclage des efforts extérieurs est cohérente. En effet les efforts augmentent puis décrochent avant de réaugmenter de la même manière trois fois de suite, avec des amplitudes et vitesses différentes. Cela correspond à la perforation des différents organes, à commencer par la peau, élastique et peu épaisse qui donne donc une augmentation relativement légère. Vient ensuite le muscle, au comportement similaire mais plus épais, puis l'os. Très résistant, l'os est modélisé par une très grande constante de raideur mais sur une faible durée : une fois cassé il ne résiste plus, ce qui est modélisé par une fine couche d'os. Vient enfin l'organe, mou mais plus épais. Le dernier décrochage est quand à lui lié à la consigne : en effet cette simulation est réalisée pour une consigne sinusoïdale en position qui explique l'augmentation plus progressive de la contrainte vers la fin de la perforation et l'annulation de celle-ci, correspondant au moment à partir duquel l'aiguille commence a reculer. On notera que les efforts sont toujours nuls lorsque la vitesse est positive dans le référentiel articulaire : lorsque la pointe repart en sens inverse, le robot ne subit pas d'effort extérieur particulier supplémentaire.

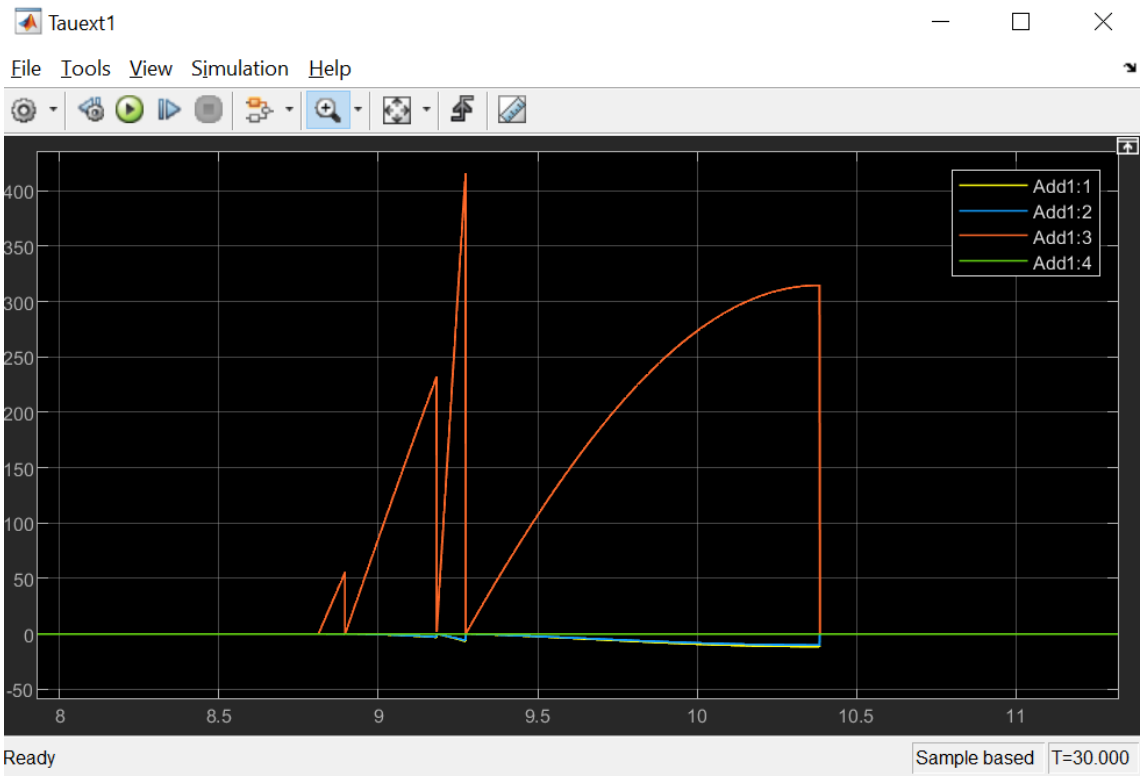

FIGURE 9 – Graphe des efforts extérieurs au parcours du corps

Dans un second temps, le bouclage est effectué. Plusieurs observations sont à analyser. On remarque que la machine sature et que le modèle est incohérent. Cela s'explique par notre analyse plutôt réaliste quant à la physique de la perforation du corps qui peut nécessiter des efforts assez importants, notamment pour percer les os très durs. Deux possibilités s'offrent à nous :
- Considérer que les efforts nécessaires sont plus importants que ceux annoncés et augmenter la limitation des couples moteurs, ce qui nécessite de revoir le modèle pour tous les autres groupes
- Diminuer artificiellement l'influence des efforts extérieurs avec un gain faible ($*0.01$) pour ne pas contraindre les moteurs
Nous avons choisi cette dernière option pour ne pas contraindre d'autres groupes à changer leur modèle et privilégier le modèle fourni, même si selon nous celui-ci pourrait être considéré par des aspects physiques. Après cette correction, le résultat donné par le modèle répond tout à fait à la consigne, avec une bonne réactivité, notamment concernant les efforts extérieurs dus à un choc avec l'opérateur par exemple.

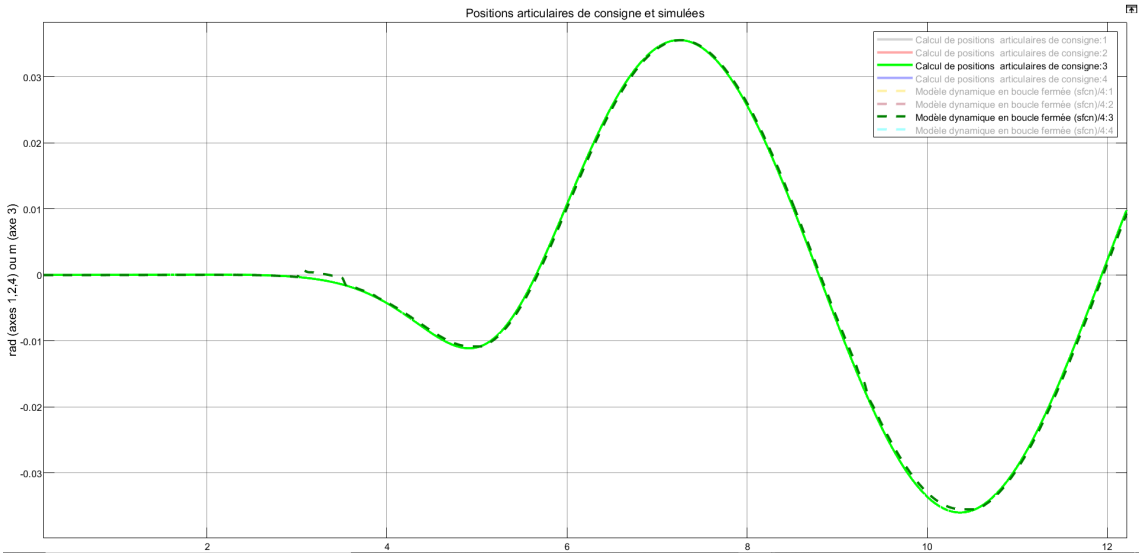

FIGURE 10 – Graphe de la position articulaire de réponse (pointillé) à la consigne (continue) avec réponse à un choc opérateur (entre 3 et 3,5s)

Ci-dessous un exemple de simulation, avec une collision opérateur entre 9,5 et 10 secondes d'une intensité de 30N sur chaque composante. On observe que le bouclage effectué permet de compenser les efforts de collision.

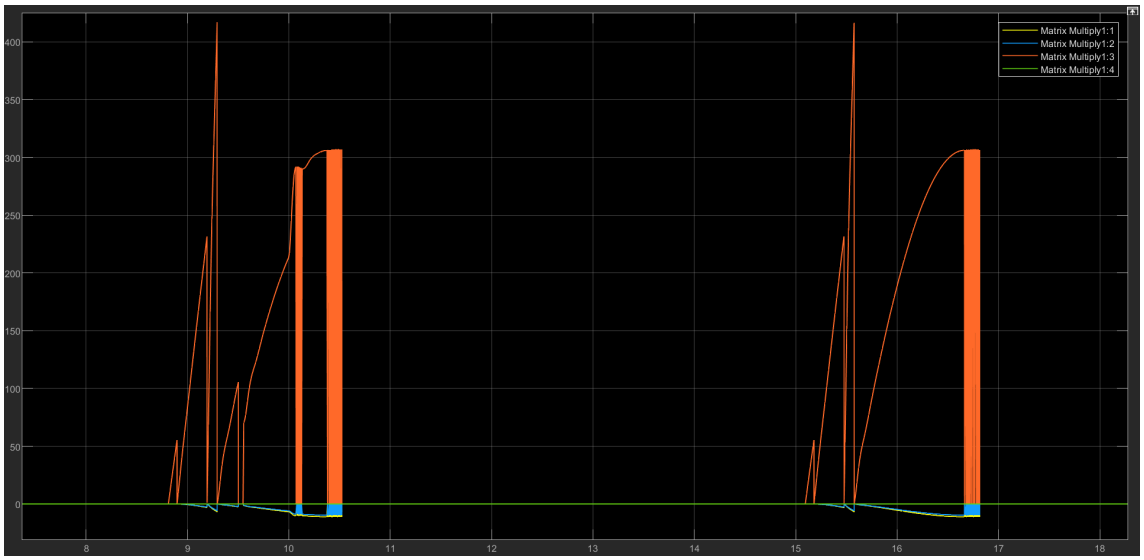

FIGURE 11 – Graphe des efforts extérieurs au parcours du corps en bouclage fermé, avec collision opérateur entre 9,5 et 10s

7 Conclusion (lien éventuel avec autres équipes)

Ce travail sur la chirurgie mini-invasive nous a permis d'approfondir nos connaissances dans le domaine. De plus, la décomposition en différent sujet permet au mieux de comprendre le monde de l'entreprise, en simulant le travail d'un service R&D. Notre démarche a tout d'abord été de modéliser les efforts d'interaction du robot avec les différentes parties du corps. Pour cela nous avons réalisé une modélisation dynamique du corps en introduisant différents paramètres physiques. Nous avons aussi modélisé les collisions avec l'environnement à partir d'une force extérieure donnée. Le module de simulation permet d'étudier l'influence des paramètres sur les efforts d'interaction visibles au niveau articulaire. Ce travail permet le développement du robot en relation avec les autres études réalisées par les autres groupes. En effet la modélisation de la collision donne l'idée de générer un discriminant entre les efforts extérieurs fournis et les efforts extérieurs théoriques obtenus par simulation. La différence engendrée laisse mettre en place des mesures de protection en cas de choc, comme l'arrêt du robot, selon des critères définis en amont. On pourra par exemple mettre une valeur seuil de différence pour prendre en compte les incertitudes de calcul mais aussi la perforation d'un tissu au lieu d'un autre qui aurait donc des constantes physiques différentes. Une autre possibilité serait la détection de discontinuités indiquant l'apparition d'un tissu imprévu - un petit organe dans le parcours de la chair par exemple - ou d'une collision opérateur. Il est alors possible, grâce au module de capteur développé par le groupe S2, de totalement déterminer les interactions du robot avec son environnement. Le dimensionnement du robot, travail réalisé par le groupe S5, permet de répondre aux problématiques moteurs

de la mise en place du système. Enfin ce travail s'implémente dans le modèle dynamique du robot du groupe S3 en ajoutant des contraintes réalistes et nécessaires à l'utilisation du robot dans sa fonction d'agir sur le corps du patient.

8 Annexe

Code Live Script pour calculer la jacobienne et Oend :

```
syms q1 q2 q3 q4

q = [q1 q2 q3 q4];
l4=0.4357;
m = [156*pi/180 0 q1 0; 55*pi/180 0 q2 0; 55*pi/180 0 0
   q3+l4; 0 0 q4 -l4];

[Jv,Jw, Oend]=Jacobienne(4,m, q);

J = [Jv; Jw]
vpa(J)

Oend
vpa(Oend)


Jv = subs(Jv, [q1,q2,q3,q4],[0,0,0.4357,0]);
Jw = subs(Jw, [q1,q2,q3,q4],[0,0,0.4357,0]);


Jw = vpa(Jw);
Jv = vpa(Jv);


Jt = transpose(J)



function output = T(j,m)
    theta = m(j,3);
    alpha = m(j,1);
    d = m(j,2);
    r = m(j,4);
    output = [
```

```
        cos(theta) -sin(theta) 0 d;
        cos(alpha)*sin(theta) cos(alpha)*cos(theta) -sin(
           alpha) -r*sin(alpha);
        sin(alpha)*sin(theta) sin(alpha)*cos(theta) cos(
           alpha) r*cos(alpha);
        0 0 0 1];
    end


function [Oend,output, zn]= Tf(n,m)
    output = T(1,m);
    zn = [output(1:length(m)-1,length(m)-1)];
    for i = 2:n
        output = output * T(i,m);
        zn = [zn output(1:length(m)-1,length(m)-1)];
    end
    Oend = output(1:length(m)-1,length(m));
end

function [Jv, Jw, Oend] = Jacobienne(n,m, q)
    [Oend,resultTf, zn]=Tf(n,m);
    Jv = jacobian(Oend,q);
    Jw = zn;
    Jw = [Jw(:,1), Jw(:,2), 0*Jw(:,3), Jw(:,4)];
end
```

Code pour Simulink pour calculer Oend :

```
function On = Oend(q1,q2,q3,q4)
    On = [0.819152044288991789684488385916 84*(q3 +
       0.4357)*(cos(q1)*sin(q2) +
       0.573576436351046096108031912826 16*cos(q2)*sin(q1)
       ) - 0.356904545696713722765531589743 97*cos(q1)*sin
       (q2) - 0.204712037438210141466187706065 19*cos(q2)*
       sin(q1) - 0.204712037438210141466187706065 19*sin(
       q1) + 0.469846310392954192027054638662 37*sin(q1)*(
       q3 + 0.4357); 0.573576436351046096108031912826 16*(
       q3 + 0.4357)*(0.748332629178859081789068418108 14*
       cos(q1) - 0.233294554268804871707436481 71) -
       0.118913154116726856076111337569 87*cos(q2) -
       0.187013751926438931525622000404 37*cos(q1)*cos(q2)
        - 0.187013751926438931525622000404 37*cos(q1) +
       0.326048526533228901935497109769 72*sin(q1)*sin(q2)
```

```
        + 0.81915204428899178968448838591684*(q3 +
       0.4357)*(0.33317915266278373753003874609319*cos(q2
       ) + 0.52398814803932854993517923405158*cos(q1)*cos
       (q2) - 0.91354545764260089550212757198532*sin(q1)*
       sin(q2)) + 0.058302001271399294442300358421917;
       0.083263886904825127836070372288138*cos(q1) -
       0.26708331704710803614716623159 81*cos(q2) +
       0.083263886904825127836070372288138*cos(q1)*cos(q2
       ) - 0.14516615681517487444183788167 28*sin(q1)*sin(
       q2) + 0.81915204428899178968448838591684*(q3 +
       0.4357)*(0.74833262917885908178906841810814*cos(q2
       ) - 0.23329455426880487170743648171*cos(q1)*cos(q2
       ) + 0.40673664307580020775398599034 15*sin(q1)*sin(
       q2)) - 0.57357643635104609610803191282616*
       (0.33317915266278373753003874609319*cos(q1) +
       0.52398814803932854993517923405158)*(q3 + 0.4357)
       + 0.13094843884777317402311075151591];
end
```

Code pour Simulink pour calculer la Jacobienne :

```
function J = Jacobienne(q1,q2,q3,q4)
    J = [0.35690454569671372276553158974397*sin(q1)*sin(
       q2) - 0.20471203743821014146618770606519*cos(q1)*
       cos(q2) - 0.20471203743821014146618770606519*cos(
       q1) + 0.46984631039295419202705463866237*cos(q1)*(
       q3 + 0.4357) +
       0.81915204428899178968448838591684*(q3 + 0.4357)
       *(0.57357643635104609610803191282616*cos(q1)*cos(
       q2) - sin(q1)*sin(q2)),
       0.20471203743821014146618770606519*sin(q1)*sin(q2)
        - 0.35690454569671372276553158974397*cos(q1)*cos(
       q2) + 0.81915204428899178968448838591684*(q3 +
       0.4357)*(cos(q1)*cos(q2) -
       0.57357643635104609610803191282616*sin(q1)*sin(q2)
       ), 0.46984631039295419202705463866237*sin(q1) +
       0.81915204428899178968448838591684*cos(q1)*sin(q2)
        + 0.46984631039295419202705463866237*cos(q2)*sin(
       q1), 0; 0.18701375192643893152562200040437*sin(q1)
        + 0.32604852653322890193549710976972*cos(q1)*sin(
       q2) + 0.18701375192643893152562200040437*cos(q2)*
       sin(q1) - 0.42922596264961884674230433877524*sin(
       q1)*(q3 + 0.4357) -
```

```
0.81915204428899178968448838591684*(q3 + 0.4357)
*(0.91354545764260089550212757198532*cos(q1)*sin(
q2) + 0.52398814803932854993517923405158*cos(q2)*
sin(q1)), 0.11891315411672685607611133756987*sin(
q2) + 0.18701375192643893152562200040437*cos(q1)*
sin(q2) + 0.32604852653322890193549710976972*cos(
q2)*sin(q1) - 0.81915204428899178968448838591684*(
q3 + 0.4357)*(0.33317915266278373753003874609319*
sin(q2) + 0.52398814803932854993517923405158*cos(
q1)*sin(q2) + 0.91354545764260089550212757198532*
cos(q2)*sin(q1)),
0.42922596264961884674230433877524*cos(q1) +
0.27292438401819338094126999671763*cos(q2) +
0.42922596264961884674230433877524*cos(q1)*cos(q2)
 - 0.74833262917885908178906841810814*sin(q1)*sin(
q2) - 0.13381225905760682681271599362386, 0;
0.81915204428899178968448838591684*(q3 + 0.4357)
*(0.40673664307580020775398599034l5*cos(q1)*sin(q2
) + 0.23329455426880487170743648171*cos(q2)*sin(q1
)) - 0.08326388690482512783607037228813 8*sin(q1) -
 0.14516615681517487444183788167 28*cos(q1)*sin(q2)
 - 0.08326388690482512783607037228813 8*cos(q2)*sin
(q1) + 0.19110371105078064685809128365421*sin(q1)
*(q3 + 0.4357), 0.26708331704710803614716623159 81*
sin(q2) - 0.08326388690482512783607037228813 8*cos(
q1)*sin(q2) - 0.14516615681517487444183788167 28*
cos(q2)*sin(q1) +
0.81915204428899178968448838591684*(q3 + 0.4357)
*(0.23329455426880487170743648171*cos(q1)*sin(q2)
- 0.74833262917885908178906841810814*sin(q2) +
0.40673664307580020775398599034l5*cos(q2)*sin(q1))
, 0.61299820300001844422117565204979*cos(q2) -
0.19110371105078064685809128365421*cos(q1) -
0.19110371105078064685809128365421*cos(q1)*cos(q2)
 + 0.33317915266278373753003874609319*sin(q1)*sin(
q2) - 0.30054725464258245128095191993552, 0; 0,
0.81915204428899178968448838591684*sin(q1), 0,
0.46984631039295419202705463866237*sin(q1) +
0.81915204428899178968448838591684*cos(q1)*sin(q2)
 + 0.46984631039295419202705463866237*cos(q2)*sin(
q1); -0.40673664307580020775398599034l5,
0.74833262917885908178906841810814*cos(q1) -
```

```
        0.233294554268804871707436481 71, 0,
        0.429225962649618846742304338775 24*cos(q1) +
        0.272924384018193380941269996717 63*cos(q2) +
        0.429225962649618846742304338775 24*cos(q1)*cos(q2)
         - 0.748332629178859081789068418108 14*sin(q1)*sin(
        q2) - 0.133812259057606826812715993623 86;
        -0.913545457642600895502127571985 32, -
        0.333179152662783737530038746093 19*cos(q1) -
        0.523988148039328549935179234051 58, 0,
        0.612998203000018444221175652049 79*cos(q2) -
        0.191103711050780646858091283654 21*cos(q1) -
        0.191103711050780646858091283654 21*cos(q1)*cos(q2)
         + 0.333179152662783737530038746093 19*sin(q1)*sin(
        q2) - 0.300547254642582451280951919935 52];
end
```

Code pour Simulink pour calculer α et k et x :

```
function [alpha, k,x] = compute_alpha_k(V,On,E,C)
% E est une matrice de n*2 éléments : chaque ligne
   contient 2 élements,
% le début d'une frontière et sa fin dans l'espace des
   positions, avec n
% matières possibles à traverser
% C est un vecteur de dimension 2 x n qui contient les
   alpha et les k pour
% chacun des n matériaux
multiplier = 1000;
    if On > E(1,2) && On < E(1,1)
        alpha = C(1,1)/multiplier;
        k = C(1,2)*multiplier;
        begin = E(1,1);
    elseif On <= E(2,1) && On > E(2,2)
        alpha = C(2,1)/multiplier;
        k = C(2,2)*multiplier;
        begin = E(2,1);
    elseif On <= E(3,1) && On > E(3,2)
        alpha = C(3,1)/multiplier;
        k = C(3,2)*multiplier;
        begin = E(3,1);
    elseif On <= E(4,1) && On > E(4,2)
        alpha = C(4,1)/multiplier;
        k = C(4,2)*multiplier;
```

```
        begin = E(4,1);
    else
        alpha = C(5,1)/multiplier;
        k = C(5,2)*multiplier;
        begin = E(4,2);
    end
    x = On-begin;
    if V >= 0
        k = 0;
    end
end
```

Code pour Simulink pour calculer la Jacobienne d'un impact avec l'effecteur du robot :

```
function J = Jacobienne_modifiee(q1,q2,q3,q4, yop)
    J = [0.35690454569671372276553158974397*sin(q1)*sin(
       q2) - 0.20471203743821014146618770606519*cos(q1)*
       cos(q2) - 0.20471203743821014146618770606519*cos(
       q1) + 0.46984631039295419202705463866237*cos(q1)*(
       q3 + 0.4357-yop) +
       0.81915204428899178968448838591684*(q3 + 0.4357-
       yop)*(0.57357643635104609610803191282616*cos(q1)*
       cos(q2) - sin(q1)*sin(q2)),
       0.20471203743821014146618770606519*sin(q1)*sin(q2)
        - 0.35690454569671372276553158974397*cos(q1)*cos(
       q2) + 0.81915204428899178968448838591684*(q3 +
       0.4357-yop)*(cos(q1)*cos(q2) -
       0.57357643635104609610803191282616*sin(q1)*sin(q2)
       ), 0.46984631039295419202705463866237*sin(q1) +
       0.81915204428899178968448838591684*cos(q1)*sin(q2)
        + 0.46984631039295419202705463866237*cos(q2)*sin(
       q1), 0; 0.18701375192643893152562200040437*sin(q1)
        + 0.32604852653322890193549710976972*cos(q1)*sin(
       q2) + 0.18701375192643893152562200040437*cos(q2)*
       sin(q1) - 0.42922596264961884674230433877524*sin(
       q1)*(q3 + 0.4357-yop) -
       0.81915204428899178968448838591684*(q3 + 0.4357-
       yop)*(0.91354545764260089550212757198532*cos(q1)*
       sin(q2) + 0.52398814803932854993517923405158*cos(
       q2)*sin(q1)), 0.11891315411672685607611133756987*
       sin(q2) + 0.18701375192643893152562200040437*cos(
       q1)*sin(q2) + 0.32604852653322890193549710976972*
```

```
cos(q2)*sin(q1) -
0.819152044288991789684488385916 84*(q3 + 0.4357-
yop)*(0.333179152662783737530038746093 19*sin(q2) +
 0.523988148039328549935179234051 58*cos(q1)*sin(q2
) + 0.913545457642600895502127571985 32*cos(q2)*sin
(q1)), 0.429225962649618846742304338775 24*cos(q1)
+ 0.272924384018193380941269996717 63*cos(q2) +
0.429225962649618846742304338775 24*cos(q1)*cos(q2)
 - 0.748332629178859081789068418108 14*sin(q1)*sin(
q2) - 0.133812259057606826812715993623 86, 0;
0.819152044288991789684488385916 84*(q3 + 0.4357-
yop)*(0.406736643075800207753985990341 5*cos(q1)*
sin(q2) + 0.233294554268804871707436481 71*cos(q2)*
sin(q1)) - 0.083263886904825127836070372288 138*sin
(q1) - 0.145166156815174874441837881672 8*cos(q1)*
sin(q2) - 0.083263886904825127836070372288 138*cos(
q2)*sin(q1) + 0.191103711050780646858091283654 21*
sin(q1)*(q3 + 0.4357-yop),
0.267083317047108036147166231598 1*sin(q2) -
0.083263886904825127836070372288 138*cos(q1)*sin(q2
) - 0.145166156815174874441837881672 8*cos(q2)*sin(
q1) + 0.819152044288991789684488385916 84*(q3 +
0.4357-yop)*(0.233294554268804871707436481 71*cos(
q1)*sin(q2) - 0.748332629178859081789068418108 14*
sin(q2) + 0.406736643075800207753985990341 5*cos(q2
)*sin(q1)), 0.612998203000018444221175652049 79*cos
(q2) - 0.191103711050780646858091283654 21*cos(q1)
- 0.191103711050780646858091283654 21*cos(q1)*cos(
q2) + 0.333179152662783737530038746093 19*sin(q1)*
sin(q2) - 0.300547254642582451280951919935 52, 0;
0, 0.819152044288991789684488385916 84*sin(q1), 0,
0.469846310392954192027054638662 37*sin(q1) +
0.819152044288991789684488385916 84*cos(q1)*sin(q2)
 + 0.469846310392954192027054638662 37*cos(q2)*sin(
q1); -0.406736643075800207753985990341 5,
0.748332629178859081789068418108 14*cos(q1) -
0.233294554268804871707436481 71, 0,
0.429225962649618846742304338775 24*cos(q1) +
0.272924384018193380941269996717 63*cos(q2) +
0.429225962649618846742304338775 24*cos(q1)*cos(q2)
 - 0.748332629178859081789068418108 14*sin(q1)*sin(
q2) - 0.133812259057606826812715993623 86;
```

```
        -0.91354545764260089550212757198532, -
        0.33317915266278373753003874609319*cos(q1) -
        0.52398814803932854993517923405158, 0,
        0.61299820300001844422117565204979*cos(q2) -
        0.19110371105078064685809128365421*cos(q1) -
        0.19110371105078064685809128365421*cos(q1)*cos(q2)
         + 0.33317915266278373753003874609319*sin(q1)*sin(
        q2) - 0.30054725464258245128095191993552];
end
```

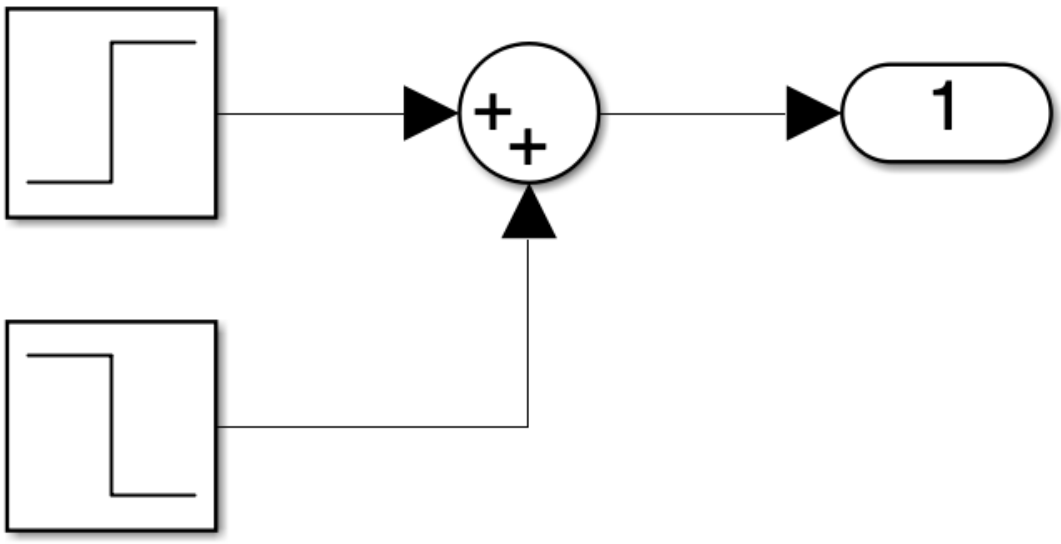

FIGURE 12 – Schéma bloc d'obtention des efforts extérieurs en cartésien : deux blocs step qui forment un créneau impulsion