CEA Comité Español de Automática

XLV JORNADAS AUTOMÁTICA MÁLAGA 2024

UNIVERSIDADE DA CORUÑA

Jornadas de Automática

Integrator for Musculoskeletal Simulation in Python

Garcia-Mascaraque Herrera, Alicia*, Montesino, Ignacio, Victores, Juan G., Balaguer, Carlos, Jardón, Alberto

RoboticsLab, Dpto. de Ing. Sistemas y Automática, Universidad Carlos III de Madrid, Av. de la Universidad, nº 30, 28911 Leganés, Madrid, España.

To cite this article: Garcia-Mascaraque Herrera, Alicia, Montesino, Ignacio, Victores, Juan G., Balaguer, Carlos, Jardón, Alberto. 2024. Integrator for Musculoskeletal Simulation in Python.
Jornadas de Automática, 45. https://doi.org/10.17979/ja-cea.2024.45.10802

Resumen

La detección de la espasticidad es compleja debido a su origen en el sistema nervioso central y los síntomas musculares. Este proyecto desarrolla un método automático para detectar el grado de espasticidad según la escala Ashworth, usando movimientos pasivos, rápidos y suaves en el paciente y midiendo la excitación cerebral. Software como OpenSim y técnicas como Computed Muscle Control (CMC) han facilitado esta tarea ofreciendo el valor de la excitación cerebral dada una trayectoria de movimiento, aunque presentan errores cuando el tipo de movimiento es suave y rápido. Este artículo implementa el integrador Hybrid Computed Muscle Control (HCMC) con el método Runge-Kutta, mejorando la precisión con un error del 3% respecto a CMC. La implementación que se ha desarrollado en Python aumenta la accesibilidad y permitirá crear una base de datos de pacientes virtuales para aplicar técnicas de Machine Learning, avanzando en el desarrollo de nuevos métodos de diagnóstico.

Palabras clave: Tecnología de asistencia e ingeniería de rehabilitación, Bioinformática, Modelización, simulación y visualización de sistemas biomédicos, Dinámica y control, Identificación y validación, Control predictivo de modelos y basado en la optimización, Programación, coordinación, optimización .

Integrador para Simulación Músculo-esquelética en Python

Abstract

The detection of spasticity is complex due to its origin in the central nervous system and muscular symptoms. This project develops an automatic method to detect the degree of spasticity according to the Ashworth scale, using passive, fast and smooth movements in the patient and measuring brain excitation. Software such as OpenSim and techniques such as Computed Muscle Control (CMC) have facilitated this task by providing the value of brain excitation given a movement trajectory, although they present errors when the type of movement is smooth and fast. This paper implements the Hybrid Computed Muscle Control (HCMC) integrator with the Runge-Kutta method, improving the accuracy with an error of 3% with respect to CMC. The implementation that has been developed in Python increases accessibility and will allow the creation of a virtual patient database to apply Machine Learning techniques, advancing the development of new diagnostic methods.

Keywords: Assistive technology and rehabilitation engineering, Bioinformatics, Biomedical system modeling, simulation and visualization, Dynamics and control, Identification and validation, Model predictive and optimization-based control, Scheduling, coordination, optimization .

1. Introduction

Spasticity is generally defined as a "*disorder of sensory-motor control resulting from upper motor neuron injury, presenting as intermittent or sustained involuntary activation of muscles*" (Burridge et al., 2005). We use this definition to highlight the complexity of objectively measuring the level of involvement of the disorder in motion because of the variety of

*Autor para correspondencia: alicia.garciamascaraque@gmail.com, alicia.garcia-mascaraque@uc3m.es

symptoms and the nature of the disorder in the central nervous system.

The Ashworth scale is used to identify the level of spasticity, with values between zero and four. It represents the lack of increased muscle tone with the lowest values (0, 1, 1+), and severe limitations in muscle resistance to stretching with the highest (2, 3, 4). The diagnosis of a patient on this scale is made by observing slow and smooth voluntary movements of the involved limbs. Therefore, conventional medical methods are ineffective and slow Seth et al. (2011). As a consequence, there has been a growing need in recent years to develop instrumented or automated methods for the assessment of spasticity in neurorehabilitation Bengio et al. (1994).

To provide a solution in this area, we are looking for an algorithm that can calculate muscle excitation patterns based on a particular body movement trajectory. One crucial criterion that must be met is code structure and accessibility in order to serve as a foundation for future applications with Machine Learning modules in Python. Furthermore, this algorithm will allow us to build a synthetic patient database, which will be used to identify an adaptive strategy that works for each individual with spasticity. Actual methods are primarily produced using software such as OpenSim or Mujoco, and it is difficult to work on them for the time required to extract the necessary information: excitations of the muscles implied in the studied movement de-la Torre et al. (2024).

Prior to building the artificial patient database, the main goal is to create an algorithm that can calculate muscle excitation patterns based on a given body movement trajectory. To do so, it is essential to comprehend that, during voluntary movement, the brain releases an excitement impulse that goes via nerves to the muscles and produces force. These forces acting on the bones cause the joints to move in various directions. This suggests that a biological model for muscle dynamics must be constructed in order to extract the excitation from the equation of motion Delp and Loan (2000).

The analysis of the situation's modelling, including the selection of the muscle model and the dynamics for the equations of motion, is presented in Section 2. In Section 3, alternative integration methods, and accessible software for solving the problem, are compared to the actual approach utilised in this article. Furthermore, in Section 4, the method is tested with simulated data to illustrate the improvements.

2. Biomedical Modelling of the Musculoskeletal Dynamics

Let $T \in \mathbb{R}$ be the final time of the movement of the patient. Then, $v(t),\ t \in [0, T]$ is the muscle fibre velocities and $V \in \mathbb{R}$ is the limit velocity where spasticity starts to show symptoms in movement behaviour. If we consider $u(t),\ t \in [0, T]$ as the brain excitation, and $G \in \{0, 1, 1+, 2, 3, 4\}$ the spasticity level with the Ashworth scale.

$$\dot{u}(t) = \begin{cases} G \cdot v(t) & \text{if } v(t) > V \\ 0 & \text{if } v(t) \leq V \end{cases} . \tag{1}$$

The degree of spasticity may be easily determined if one knows the brain excitation: $0 \leq u(t) \leq 1,\ t \in [0, T]$, with this approach. In order to calculate it, one must solve an inverse kinetics problem using the joint space trajectories that represent the angular motions of joints (Wakeling et al., 2023). In this article, the focus will be on modelling the gait's outcomes as it is the most complex movement to simulate and it can be generalised to other limbs (Inai et al., 2020). To do this, we can specify the joint space trajectory as

$$q(t) = \begin{pmatrix} q_1(t) \\ q_2(t) \\ q_3(t) \end{pmatrix} \in \mathbb{R}^3,\ t \in [0, T]\,,$$

where each vector component represents the angles at different legs' joints: hip joint, $q_1(t) \in \mathbb{R}$, knee joint, $q_2(t) \in \mathbb{R}$, and ankle joint, $q_3(t) \in \mathbb{R}$ for each time step $t \in [0, T]$ as it can be specified in Figure 1.

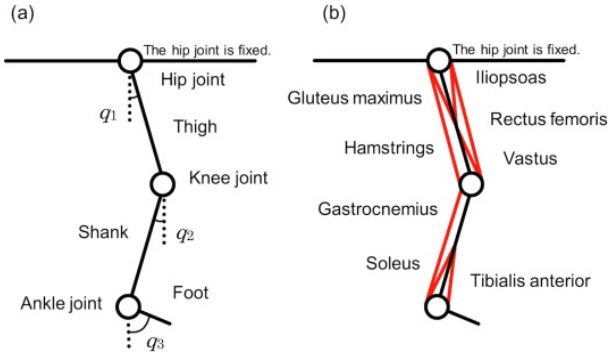

Figure 1: Leg scheme for implementing the musculoskeletal simulation using generalised coordinates. In (a), we can observe the variables used for indicating trajectories: angles between the vertical line and the thigh, shank, and foot (q_1, q_2, q_3). In (b), it is represented the musculoskeletal model with the implied muscles in producing the gait (Inai et al., 2020).

To create any muscle-driven simulation tool, initially it is important to choose a dynamic model of the musculoskeletal system and its interactions with the environment. Let $a(t) \in \mathbb{R}^n,\ t \in [0, T]$ where $n \in \mathbb{N}$ is the number of muscles suggested in the action; in this case, $n = 8$ as in Figure 1. The kinetics of muscle activation are expressed as follows:

$$\dot{a}(t) = f_a(a(t), u(t))\,, \tag{2}$$

where $f_a(a(t), u(t))$ is the activation function. The Hill-type muscle model, a 3-element model based on a Contractile Element (CE) in series with a Spring Element (SE), and an elastic Parallel Element (PE), is the most commonly used muscle model to analyse the dynamics of muscular contraction (Delp and Loan, 2000). It is represented by the scheme in Figure 2, and the variables utilised in the equations are:

- $\dot{l}^M \in \mathbb{R}^n(m/s)$ is the muscle fibre velocity.
- $l^M \in \mathbb{R}^n(m)$ is the muscle fibre length.
- $l_0^M \in \mathbb{R}^n(m)$ is the optimal muscle fibre length.
- $l^T \in \mathbb{R}^n(m)$ is the tendon length.
- $l^{MT} = l^M + l^T \in \mathbb{R}^n(m)$ is the musculo-tendon length.
- $l_S^T \in \mathbb{R}^n(m)$ the tendon slack length and $l^{MT}(m) \in \mathbb{R}^n$ is the musculo-tendon length.
- $F_0^M \in \mathbb{R}^n(N)$ is the muscle force at optimal fibre length.
- $f_T \in \mathbb{R}^n(N)$ is the tendon force at each muscle.

- α (°) $\in \mathbb{R}^n$ is the pennation angle, and α_0 (°) $\in \mathbb{R}^n$ is the pennation angle at optimal fibre length.

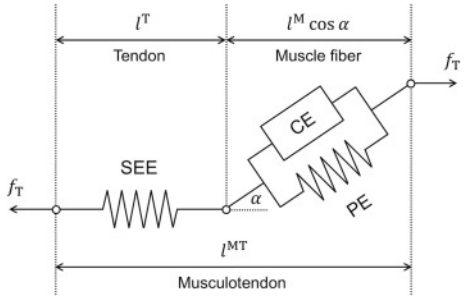

Figure 2: The Hill model consists of a Contractile Element (CE), a Passive Element (PE), and a Series Elastic Element (SEE) (Inai et al., 2020).

Taking this structure into account, in the algorithm it is necessary to implement an integrator for solving numerically the muscle contraction dynamic in Equation 3 and the equations of motion in Equation 4.

$$\dot{l}^M(t) = f_V^{-1}(l^M, l_0^M, l_S^T, l^{MT}, F_0^M, \alpha_0, a) \ , \tag{3}$$

$$\ddot{q}(t) = f_{\dot{q}}(q, \dot{q}, E, Q) = M(q)^{-1} \cdot (C(q, \dot{q}) + G(q) + Q + E) \tag{4}$$

where $M \in \mathbb{R}^{n \times n}$ is the mass matrix, $C \in \mathbb{R}^n$ the Coriolis and centrifugal force, $G \in \mathbb{R}^n$ is the gravitational force, $Q \in \mathbb{R}^n$ is the net joint moments and $E \in \mathbb{R}^n$ is the external forces.

Due to the software's widespread use, the first intuition for solving this model of musculoskeletal dynamics is to use Computed Muscle Control (CMC) from OpenSim, an open-source platform for modelling, simulating, and analysing the neuro-musculoskeletal system, to create a set of muscle excitation based on experimental kinematics and external forces (Delp et al., 2007). However, there were certain limitations while using this technique to achieve our ultimate goal of creating a synthetic patient dataset :

1. Errors in the generalised coordinates between the simulated and experimental kinematics may rise quickly with high acceleration since it is determined by proportional derivative control (Thelen and Anderson, 2006), (Delp and Loan, 2000) . When it comes to diagnosing a patient with spasticity, as the motions for rehabilitation are typically smooth and rapid, an insufficient performance could result in an incorrect diagnosis.
2. The OpenSim software's performance and accessibility have not been improved enough for the task we would like to conduct. It is not a practical tool for building a database by collecting an extensive number of simulations. It also makes applying Machine Learning methods to them challenging Wakeling et al. (2023).

3. Integrator: an Algorithm for Computing Muscle Excitation

We propose the implementation of the new algorithm called Hybrid Computed Muscle Control (HCMC). The main motivation is to overcome the limitations presented in Section 2 of CMC algorithm in accurately simulating muscle excitation patterns that track kinematics during rapid movements. Furthermore, to facilitate the accessibility throughout Machine Learning libraries, instead of having the implementation in SciLab as in the cited article (Inai et al., 2020), we implement the algorithm in Python. The usage of several Python libraries will improve algorithm behavior, as shown in Table 2, in addition to improving accessibility and the foundation of the future patient syntehtic database.

The basic idea behind HCMC is to use a hybrid of numerical integration and optimisation to compute the net joint moments in each time step of simulation and to optimise muscle activation. In contrast to CMC, the algorithm's initial step involves determining the net joint moments Q from Equation 4 through a combination of optimisation and the 4th-order Runge–Kutta method. The information from the previous time step is derived from the simulation, and the desired net joint moments are calculated through numerical integration utilising feedback gains and external forces. This approach provides improved accuracy in tracking kinematics, especially during rapid movements (Inai et al., 2020). The static optimisation method is then used to optimise muscle activation following the computation of net joint moments. This method entails decreasing the sum of squared muscle activation while guaranteeing that the computed net joint moments match or exceed the specified ones.

Table 1 can be reviewed (Howell et al., 2023) to obtain a more thorough understanding of the different approaches for integrating the inferred equation based on the application, the numerical solver and integrator, gradients, and the language it has been developed in. This description helps us understand the significance of the Python implementation and the absence of Machine Learning adaptations in other muscle excitation computation tools.

3.1. Code Implementation in Python

To assess the HCMC algorithm in Python, we employ some improvements to the results by utilising well-known Python libraries for optimisation, such as Scipy. Scipy offers linear programming, constrained and nonlinear least-squares optimisation techniques with the use of the Levenberg-Marquardt algorithm. The `qpsolvers` library has been utilised for solving quadratic programming problems, as can be seen in Table 1.

When working on a small scale, the number of decimals taken into account has a significant impact on the final result since differences in computing derivatives must be considered. The issue faced when implementing this computation is comparable to the vanishing gradient problem (Bengio et al., 1994). It is essential to note that in SciLab, solvers and integrators are based on symbolic interpretation instead of numerical, thereby raising specific issues in the Python implementation. Numpy offers a variety of data types for representing numbers, including float64 and float128, capable of representing around 15 significant digits and 34 significant digits, respectively. However, the use of float128 can lead to slower computation time, greater memory consumption and rounding errors in both data types. As a result, the use of arbitrary-precision libraries such as `mpmath` can be beneficial when working with problems that require high precision. Any

Table 1: Comparison of engines used for robotics musculoskeletal simulations. Examining the various muscle excitation algorithms and software programs, filtered by year (Howell et al., 2023). The gradients of the algorithm, the implementation language, the solver and integrator, and the applications are all used for comparison. Note that: LCP: Linear Complementarity Problem. NCP-Nonlinear Complementarity Problem. CG-Conjugate Gradient. QP-Quadratic Programming. LM-Levenberg-Maequardt algorithm.

simulator	*year*	*application*	*integrator*	*solver*	*language*	*gradients*
Ours	**2024**	**Machine Learning**	**implicit Euler/RK4**	**Newton/RK4/QP/LM**	**Python**	**finite-difference**
Dojo	2022	robotics	variational	NCP	Julia	smooth gradient
Brax	2021	graphics	explicit Euler	N/A	Python	sub-gradient
RaiSim	2021	robotics	implicit Euler	bisection	C++	-
Drake	2019	robotics	implicit Euler / RADAU5	LCP / Newton	C++	gradient-bundle
MuJoCo	2015	robotics	implicit Euler / RK4	Newton / PSG / CG	C	finite-difference
DART	2012	robotics	implicit Euler	LCP	C++	sub-gradient
Bullet	2006	graphics	implicit Euler	LCP	C/C++	sub-gradient

calculation can be performed equally well at 10-digit or 1000-digit precision (mpmath development team, 2023).

4. Results

To be able to prove the algorithm's performance, we have some synthetic data validation. Cosine curves were used to create three repetitive movements, ranging from rest to a flexed position, with a flexion angle of 10° of the hip, 20° flexion angle of the knee, and 10° plantar flexion angle of the ankle joints. The most rapid range of movement ($T = 0.6\ s$) was used to illustrate the improvements. The musculoskeletal model defined in Figure 1 and the parameters established in the cited article were used (Inai et al., 2020). The motivation for using synthetic data to validate the algorithm's behavior is to simplify the actual gait in a human body. We reduced the movement to a three-joint scheme, a periodic movement, and a limitation for each joint to produce the actual activity, rather of utilizing six muscles and their joints to compute the equation of motion.

The computation time for obtaining the excitation values using HCMC in Python is 4.85 s, whereas it takes 7.7 s in SciLab. The computational time for the Python CMC algorithm is 4.6 s. These outcomes demonstrate how using Python optimisation libraries enhances algorithm execution performance.

From the results for each muscle excitation in Figure 3, we can also see that as in CMC, maximum errors increased as the total movement time of synthetic data decreases, in HCMC algorithm this is overcome. Understanding the role that each muscle plays in the gait is also made easier by Figure 3. The rectus femoris, gastrocnemius, and iliopsas are the muscles that require an excitation value $u(t) > \frac{1}{2}$ at some time $t \in [0, T]$ in order to execute the movement.

From the graphics of each muscle computation in Figure 3, the variability of the results depending on the use of CMC or HCMC is validated. Let $\Delta t = 0.01$, then the used time partition is $\mathcal{P} = \{t_0 = 0, t_1, \ldots, t_N = T\}$. Explicit errors are calculated using the simulated trajectory $q_{sim}(t_j) \in \mathbb{R}^3\ t_j \in \mathcal{P}$, and the synthetic trajectory $q(t_j) \in \mathbb{R}^3\ t_j \in \mathcal{P}$. The maximum error at each joint,

$$e^i_{max} = \max_{t_i \in \mathcal{P}} \left\{ \left| q^i(t_i) - q^i_{sim}(t_i) \right| \right\}\ i \in \{1, 2, 3\},$$

and RMSE error at each joint,

$$e^i_{RMSE} = \frac{1}{N+1} \cdot \left\| q^i - q^i_{sim} \right\|^2 = \frac{1}{N+1} \cdot \sum_{j=0}^{N} \left(q^i_{sim}(t_j) - q^i(t_j) \right)^2\ i \in \{1, 2, 3\},$$

are shown in Table 2. Taking into account all the computed errors over a mean of 10 rounds, the error using HCMC is a 3% with respect to CMC.

Table 2: Maximum and RMSE error for applying HCMC algorithm to synthetic data.

	q_1	q_2	q_3
$e_{max}(^\circ)$			
HCMC in Python	0.0537	0.0880	0.2305
CMC in Python	1.6756	2.8159	10.4
$e_{RMSE}(^\circ)$			
HCMC in Python	0.0104	0.01503	0.0375
CMC in Python	0.7147	1.2252	5.6407

Acknowledgement

This research has been financed by ROBOASSET, "Sistemas robóticos inteligentes de diagnóstico y rehabilitación de terapias de miembro superior", PID2020-113508RB-I00, financed by AEI/10.13039/501100011033; "RoboCity2030-DIH-CM, Madrid Robotics Digital Innovation Hub", S2018/NMT-4331, financed by "Programas de Actividades I+D en la Comunidad de Madrid"; "iREHAB: AI-powered Robotic Personalized Rehabilitation", ISCIII-AES-2022/003041 financed by ISCIII and EU; and EU structural funds.

References

Bengio, Y., Simard, P., Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks 5 (2), 157–166.
DOI: 10.1109/72.279181

Burridge, J. H., Wood, D. E., Hermens, H. J., Voerman, G. E., Johnson, G. R., van Wijck, F., Platz, T., Gregoric, M., Hitchcock, R., Pandyan, A. D., 1 2005. Theoretical and methodological considerations in the measurement of spasticity. Disability and Rehabilitation 27, 69–80.
DOI: 10.1080/09638280400014592

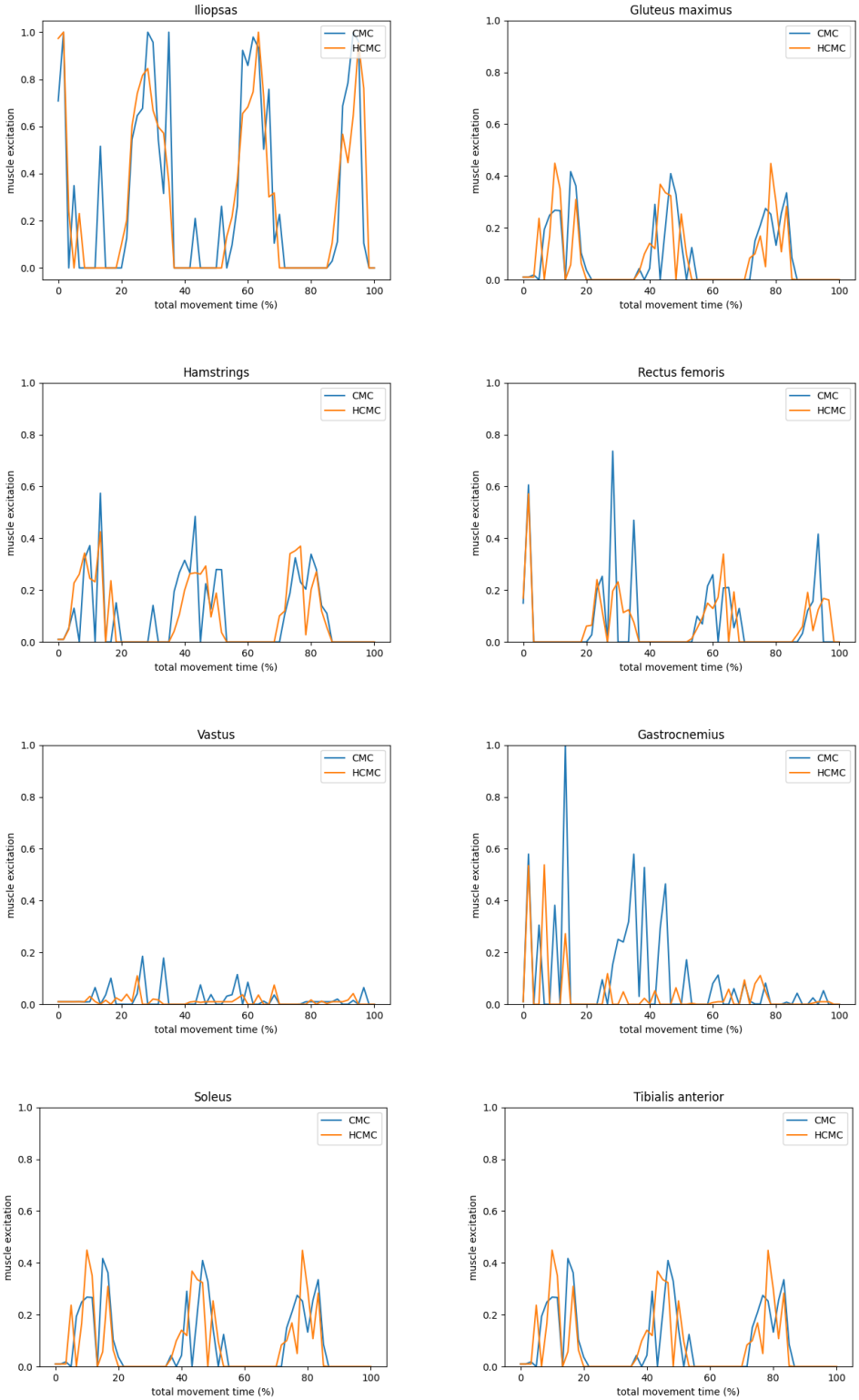

Figure 3: Muscle excitation, $0 < u(t_{\%}) < 1$ $t_{\%} \in [0, 100]$, time represented by percentage from total movement time ($t_{\%} = \frac{t}{T}$ $t \in [0, T]$), computation using HCMC and CMC for the fastest movement in gait, $T = 0.6$ s. Understanding the implicit representations of each muscle (such as the iliopsoas, gluteus maximus, soleus, and tibialis anteriorim) in the gait dynamics is essential to identifying the primary muscles utilized in the activity. The CMC algorithm has been employed to illustrate the muscle excitation at each time step in blue, while the HCMC method yields the same result in orange. To better understand the movement, it is repeated in three cycles, as indicated by the frequency of the curves.

de-la Torre, R., Oña, E. D., Victores, J. G., Jardón, A., 1 2024. Spasticsim: a synthetic data generation method for upper limb spasticity modelling in neurorehabilitation. Scientific Reports 2024 14:1 14, 1–16.
URL: https://www.nature.com/articles/s41598-024-51993-w
DOI: 10.1038/s41598-024-51993-w

Delp, S. L., Anderson, F. C., Arnold, A. S., Loan, P., Habib, A., John, C. T., Guendelman, E., Thelen, D. G., 2007. Opensim: Open-source software to create and analyze dynamic simulations of movement. IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING 54.
DOI: 10.1109/TBME.2007.901024

Delp, S. L., Loan, J. P., 2000. A computational framework for simulating and analyzing human and animal movement. Computing in Science and Engineering 2.
DOI: 10.1109/5992.877394

Howell, T. A., Cleac'h, S. L., Brüdigam, J., Kolter, J. Z., Schwager, M., Manchester, Z., 2023. Dojo: A differentiable physics engine for robotics.
DOI: 10.48550/arXiv.2203.00806

Inai, T., Takabayashi, T., Edama, M., Kubo, M., 2020. Algorithm to compute muscle excitation patterns that accurately track kinematics using a hybrid of numerical integration and optimization. Journal of Biomechanics 107.
DOI: 10.1016/j.jbiomech.2020.109836

mpmath development team, T., 2023. mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 1.3.0). http://mpmath.org/.

Seth, A., Sherman, M., Reinbolt, J. A., Delp, S. L., 2011. 2011 symposium on human body dynamics opensim: a musculoskeletal modeling and simulation framework for in silico investigations and exchange open access under cc by-nc-nd license. Procedia IUTAM 2, 212–232.
DOI: 10.1016/j.piutam.2011.04.021

Thelen, D. G., Anderson, F. C., 1 2006. Using computed muscle control to generate forward dynamic simulations of human walking from experimental data. Journal of Biomechanics 39, 1107–1115.
DOI: 10.1016/J.JBIOMECH.2005.02.010

Wakeling, J. M., Febrer-Nafría, M., Groote, F. D., 6 2023. A review of the human to develop muscle and musculoskeletal models for biomechanics in the last 50 years. Journal of Biomechanics 155, 111657.
DOI: 10.1016/J.JBIOMECH.2023.111657