FARMS: Framework for Animal and Robot Modeling and Simulation

Jonathan Arreguit[1, 2†*], **Shravan Tata Ramalingasetty**[1, 3†*], **Auke Ijspeert**[1*]

[1]BioRob, School of Engineering, Institute of Bioengineering, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland; [2]Innobridge Services Sàrl, Lausanne, Switzerland; [3]Department of Neurobiology and Anatomy, College of Medicine, Drexel University, Philadelphia, USA

***For correspondence:**
jonathan.arreguitoneill@innobridge.com;
shravantr@gmail.com;
auke.ijspeert@epfl.ch

[†]These authors contributed equally to this work

Abstract The study of animal locomotion and neuromechanical control offers valuable insights for advancing research in neuroscience, biomechanics, and robotics. We have developed FARMS (Framework for Animal and Robot Modeling and Simulation), an open-source, interdisciplinary framework, designed to facilitate access to modeling, simulation, and analysis of animal locomotion and bio-inspired robotic systems. By providing an accessible and user-friendly platform, FARMS aims to lower the barriers for researchers to explore the complex interactions between the nervous system, musculoskeletal structures, and their environment. Integrating the MuJoCo physics engine in a modular manner, FARMS enables realistic simulations and fosters collaboration among neuroscientists, biologists, and roboticists. FARMS has already been extensively used to study locomotion in animals such as mice, drosophila, fish, salamanders, and centipedes, serving as a platform to investigate the role of central pattern generators and sensory feedback. This article provides an overview of the FARMS framework, discusses its interdisciplinary approach, showcases its versatility through specific case studies, and highlights its effectiveness in advancing our understanding of locomotion. Overall, the goal of FARMS is to contribute to a deeper understanding of animal locomotion, the development of innovative bio-inspired robotic systems, and promote accessibility in neuromechanical research.

Introduction

Animal locomotion results from the complex interactions between the body, the nervous system, the sensory modalities, and the environment. Each of these sub-systems have functional properties that emerge only when they interact with each other (***Dickinson, 2000***). Focusing on individual components provides only partial insights into the system's overall behavior, making the study of animal movement a challenging and interdisciplinary scientific endeavor.

Collecting relevant data from animal experiments is essential for testing scientific hypotheses. However, this process presents several technical and ethical challenges, which vary depending on the animal species considered. For instance, measuring ground reaction forces in small animals like *Drosophila Melanogaster* is currently infeasible with available technologies (***Lobato-Rios et al., 2022***). Ethical constraints further complicate data collection and, in some cases, make it impossible to conduct certain experiments. These limitations highlight the importance of careful planning and specification of required measurements to ensure the feasibility of a study and to minimize unnecessary harm to animals.

To handle these challenges, the aptitude to accurately predict the outcome of experiments and identify potential causes for failure can be crucial. Predictive models based on existing knowledge can help systematically explore how different subsystems interact, thereby advancing our under-

standing of animal movement without relying solely on live experiments. Researchers have developed various modeling approaches that complement experimental studies, categorized into three types: (i) theoretical or analytical models, (ii) physical models, i.e. robots that mimic the behavior of animals, and (iii) numerical models. While all approaches contribute to studying locomotion and motor control, each has its strengths and limitations. Theoretical models, such as inverted pendulum models, have provided valuable insights into the general principles of locomotion (***Holmes et al., 2006***). However, they are limited in their ability to capture the complex interactions of multiple subsystems. Meanwhile, physical models like robots offer promising avenues for understanding animal movement (***Ijspeert, 2014***), but current technologies still face challenges in replicating the full complexity of animal biomechanics, such as muscle and skeletal dynamics.

In light of these challenges, physics-based numerical simulations offer an appealing alternative. Among them, *neuromechanical simulations* are an *in silico* approach to replicating animal movement by integrating multiple components, including a physical body, neural systems, and muscle models (***Edwards, 2010***), allowing researchers to study locomotion under controlled, virtual conditions. By replicating or predicting the behaviors observed in the real world, neuromechanical simulations can provide valuable insights into the mechanics of movement, motor control, and sensorimotor integration.

However, such simulations can require significant computational resources and extensive modeling efforts. Fortunately, advancements in hardware and software tools over recent years have made these approaches more accessible. Neuromechanical simulations cannot fully replace biological experiments or robot experiments, but they can complement both by gathering additional information (***Ijspeert and Daley, 2023***), and help make predictions.

Neuromechanical simulations

The idea of using neuromechanical simulations to study animal movement is well established, and we will here review relevant works with a particular focus on vertebrate simulations. In the 1990s, Karl Sims showed that diverse animal morphologies and locomotory behaviors could be systematically evolved using genetic algorithms (***Sims, 1994a***,b). ***Ekeberg*** (***1993***); ***Grillner et al.*** (***1995***) integrated neural networks model of spinal cord with a mechanical model of the lamprey to study swimming and body orientation. Subsequently, ***Ijspeert*** (***2001***) explored how a salamander could achieve both walking and swimming patterns using a neuromechanical simulation in 2.5D physics. ***Yakovenko et al.*** (***2004***) used a hindlimb neuromechanical model of a cat to evaluate the contribution of sensory feedback during locomotion. ***Ekeberg and Pearson*** (***2005***) investigated how reflex rules could trigger a hindlimb walking patterns in the cat. ***Harischandra et al.*** (***2010***) extended the investigation of salamander locomotion with a 3D simulator, achieving walking with all four limbs. ***Geyer and Herr*** (***2010***) studied in 2010 how sensory information could be used to generate bipedal walking patterns in a simple 2D simulation. ***Daun-Gruhn*** (***2011***) investigated how sensory information could generate the motion of the limbs in a stick insect. ***Aoi and Funato*** (***2016***) explored the role of muscle synergies in sensory-motor control in humans and rats for adaptive locomotion using 2D neuromuscular simulation models. ***Hamlet et al.*** (***2018***) investigated how curvature-based control rule combined with sensory information could achieve optimal swimming of the lamprey. Finally, ***Nyakatura et al.*** (***2019***) investigated the locomotion of the extinct stem amniote Orobates pabsti by leveraging simulations as a central tool for reconstructing its plausible gaits. Their simulation-driven approach provided key insights into how Orobates may have moved, demonstrating that it likely exhibited more advanced terrestrial locomotion than previously assumed for early tetrapods.

Neuromechanical frameworks

Previous studies on neuromechanical simulations were typically developed separately by small research groups, and would have benefited from a shared infrastructure that could reduce the need for duplication and re-implementation of models and code from scratch. The absence of such an infrastructure not only wastes resources but also limits the potential for collaboration and knowl-

Table 1. Overview and comparison of features from different frameworks related to neuromechanical simulations. Features from other frameworks that have been enhanced or addressed in FARMS are highlighted in red.

	Webots	Gazebo	OpenSim	AnimatLab	Scone	Isaac	MyoSuite	FARMS
Release	1996	2004	2007	2010	2019	2021	2022	2023
Physics engine	ODE	Dart Bullet	Simbody	Bullet Vortex	Simbody HyFyDy	PhysX	MuJoCo	MuJoCo Bullet
Programing languages	C/C++ Python	C++	C++ Python	C++	Lua Python	C++ Python	Python	Python Cython
Muscles	No	No	Yes	Yes	Yes	No	Yes	Yes
Neural models	No	No	No	Yes	No	No	No	Yes
Data logging	No	Yes (ROS)	No	Yes	Yes	Yes	Yes	Yes
Fluids	Drag	Partial	No	No	No	No	No	Drag/SPH
GUI/Headless	GUI	Both	Both	GUI	GUI	Both	Both	Both
Playback	No	Rosbag	No	Yes	Yes	Yes	Yes	Blender
Optimization	No	No	No	No	Yes	Gym	Gym	Yes
Inv. dyn.	No	No	Yes	No	Yes	No	No	Yes
Modeling	Yes	Yes	Yes	Yes	No	Yes	No	Yes
Active	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
License	Apache2.0	Apache2.0	Apache2.0	BSD-3-Clause	Apache2.0	Propr.	Apache2.0	Apache2.0

edge sharing, while also impacting reproducibility and replication studies. To overcome these limitations and accelerate our understanding of animal motor control, it is crucial that we contribute to the development of a comprehensive and open-source framework, where a framework constitutes the software infrastructure that integrates the components required for neuromechanical simulations across various applications and research domains. More precisely, while numerous projects provide specialized interfaces for physics simulation (***Smith, 2006***; ***Coumans and Bai, 2016***; ***Todorov et al., 2012***; ***Lee et al., 2018***; ***Sherman et al., 2011***), muscle modeling (***Seth et al., 2011***, ***2018***), or neural network simulations (***Hines and Carnevale, 1997***; ***Goodman and Brette, 2009***; ***Stimberg et al., 2019***), a critical gap remains. Existing options remain largely confined to their respective domains, lacking a higher-level interface for seamless integration between these components. As a result, modeling and simulating neuromechanical systems continues to be cumbersome, highlighting the need for a framework that bridges these specialized tools to enable more comprehensive and accessible neuromechanical simulations.

In recent years, numerous simulation frameworks have been developed, offering valuable tools for studying motor control and biomechanics. Among these, a curated selection is presented in ***Table 1***, which highlights key features relevant to the study of animal control. While not all of these frameworks are specifically designed for neuromechanical simulations or animal locomotion, they provide flexible capabilities that can often be adapted for such purposes. More precisely, they tend to be focused on simulating robotic systems, and tools for species-specific modeling, advanced muscle and neural models, modeling specific environments such as with fluids, and optimization or learning mechanisms for tuning controllers are missing in some frameworks or are difficult to incorporate.

Webots (***Michel, 2004***) was adopted in previous works (***Crespi and Ijspeert, 2006***; ***Nyakatura et al., 2019***; ***Thandiackal et al., 2021***) due to its ease of use. With its consolidated interface, it remains an attractive choice for introductory learning and the construction of robot models. However, its reliance on the ODE physics engine resulted in instability for complex models with many joints, as a consequence of its use of maximal coordinates for solving dynamics. Gazebo (***Koenig and Howard, 2004***) was an interesting option for the availability of multiple physics engines, but was not ideal due to its lack of direct support for a script-oriented programming language, and

limited cross-platform support. Focused on robot simulations, both Webots and Gazebo do not offer direct support for muscle models. On the other hand, OpenSim (***Seth et al., 2011***) was of particular interest for the availability of specialized tools for biomechanical modeling, including muscle models. It is a well-known tool for developing human musculoskeletal models and facilitates reverse-engineering of movement generation through motion capture. However, OpenSim is more limited for forward dynamics simulations in terms of performance and ease of use, and there remains a lack of usage related to simulating robots.

AnimatLab (***Cofer et al., 2010***) is to the best of our knowledge the only prior framework to offer both muscle and neural models for a multitude of animal models. However, its lack of active development since 2016 and limitations in extensibility highlight the need for more modern alternatives. It also lacks support for fluid dynamics, essential for swimming simulations, and is designed primarily as an interactive tool with the GUI, limiting its utility for batch simulations and parameter optimization, especially in headless mode to run on computer clusters.

Built upon OpenSim, Scone (***Geijtenbeek, 2019***) offers more efficient and stable biomechanical simulations, which is accomplished by using the closed-source HyFyDy physics engine (***Geijtenbeek, 2021***). Aiming towards also solving OpenSim's performance limitations for forward dynamics simulation, MyoSuite (***Caggiano et al., 2022***) replaces Simbody with the now open-source MuJoCo engine and provides interfaces to OpenAI's gym API (***Brockman et al., 2016***). However, both Scone's and MyoSuite's applications beyond biomechanical simulations remain secondary to their primary focus areas of simulating advanced biomechanics, and extending them to support robots and fluid simulations is not within their scope.

Isaac Sim (***Liang et al., 2018***) stands out for its reinforcement learning capabilities with robots, as it supports running multiple robot simulations in parallel thanks to its computation on the Graphics Processing Unit (GPU). However, it currently lacks integration of muscle and neural models as well as fluid environments. While efficient GPU computation support is largely beneficial, its proprietary license makes it limiting for open-source contributions.

For a comprehensive review of simulation frameworks from a robotics perspective, refer to ***Collins et al.*** (***2021***).

FARMS

To bridge the existing gaps in available tools (***Table 1***), we propose a Framework for Animal and Robot Modeling and Simulation (FARMS), specifically tailored to the needs of neuromechanical simulation and animal locomotion research. FARMS emphasizes modularity, open access and extensibility, aiming to unify current efforts and accelerate progress in understanding animal motor control. It is primarily written in the Python programming language and features a range of tools to help design models with realistic dynamics, to run controllers based on neural models, and for comparative studies with animals and robots. It also aims to support optimization methods for parameter tuning and 3D visualization for debugging and rendering high-quality videos for publication. In contrast to the previously listed frameworks, FARMS focuses on including the necessary components for simulating locomotion with musculoskeletal models, biological neural network models, and the appropriate sensors for the studied animals or robots in different environments, and promoting the development of neuromechanical models.

At an implementation level, FARMS leverages and integrates available open-source software into a set of implemented packages developed to facilitate interoperability between the different components and manage dependencies. For this purpose, it has been split into a series of packages to enable the users to pick the specific components required for their experiments, and to facilitate the development and handle contributions. Considerable effort was also put into integrating wrapper functions such that users could create and incorporate their own models and controllers and run physics-based simulations, with support for multiple physics engines, similarly to Gazebo (***Koenig and Howard, 2004***). For example, the farms_muscle package provides Hill-type

muscle models, enabling the simulation of biomechanical systems within a simulator. Another example is the farms_network package, used in ***Lobato-Rios et al.*** (***2022***) for simulating sparse neural networks, which can function independently or be seamlessly integrated with FARMS. In essence, FARMS can be viewed as a basis upon which wrapper code is written to glue existing software together.

Within FARMS, developing a neuromechanical simulation can be broken down into four main components (***Figure 1***): (i) modeling, (ii) simulation, (iii) analysis, and (iv) data, the latter serving as a bridge connecting the different components:

- **Modeling:** Two necessities for any neuromechanical simulation are the description of an animal or robot body and its controller for generating motions. The modeling block covers the tools and aspects of generating these sub-blocks.
- **Simulation:** Once both the body and controller are described, the next step consists in running a physics simulation with the controller in the loop. Simulations in FARMS involve integrating (i) a physics engine to simulate articulated rigid bodies and their interactions with the environment (e.g. land or water) and (ii) the control laws to compute the actions to generate movements.
- **Analysis:** Finally, during and at the end of a simulation, the progress of an experiment can be monitored through the collected data. The analysis component deals with performing post-processing, visualizing and replaying simulation data, and producing representations for scientific analysis and sharing.
- **Data:** At every step of the framework, the information flow is recorded and tracked. This is performed with the data block. Each module can read and write to it. This enables FARMS' modularity as data can be shared between blocks and can be populated by any independent tool.

Optimization and learning algorithms constitute an additional layer for manipulating each of these components to produce the desired animal or robot behavior.

Data

In FARMS, it is of fundamental importance to be able to reproduce a simulation experiment. To achieve this, at each stage (modeling, simulation, and analysis) appropriate data structures can be saved to disk. This decoupled approach makes FARMS a truly modular framework, allowing each component to function independently while remaining integrable. The main package that glues the components of FARMS together is farms_core, which provides consistent data handling across the modeling, simulation, and analysis stages. This unifying approach enhances the modularity of the framework, making it easier to share and reproduce experiments within the research community.

The data block within FARMS consists of three primary types: configuration files, run-time simulation data, and processed simulation data. These data types collectively contain all the necessary information to fully reproduce an experiment (***Figure 1***).

- **Configuration Files:** These files provide a comprehensive description of the simulation setup required to run experiments accurately. Typically saved in human-readable formats such as Extensible Markup Language (XML) and Yet Another Markup Language (YAML), configuration files detail the animal or robot's morphology, controller, and initial conditions, as well as environmental parameters such as friction and drag coefficients. They also specify simulation settings like timestep, simulation units, and logging preferences. This ensures that experiments can be replicated precisely, facilitating reproducibility.
- **Run-time Simulation Data:** During the simulation, the data structure provides access to the complete simulation state agnostic to the underlying physics engine, and logging information as specified by the user in the configuration files. This includes detailed records of links (positions, velocities, forces), joints (control commands, forces, limits), contacts (reaction forces), external forces, and muscles (excitations, lengths, forces).

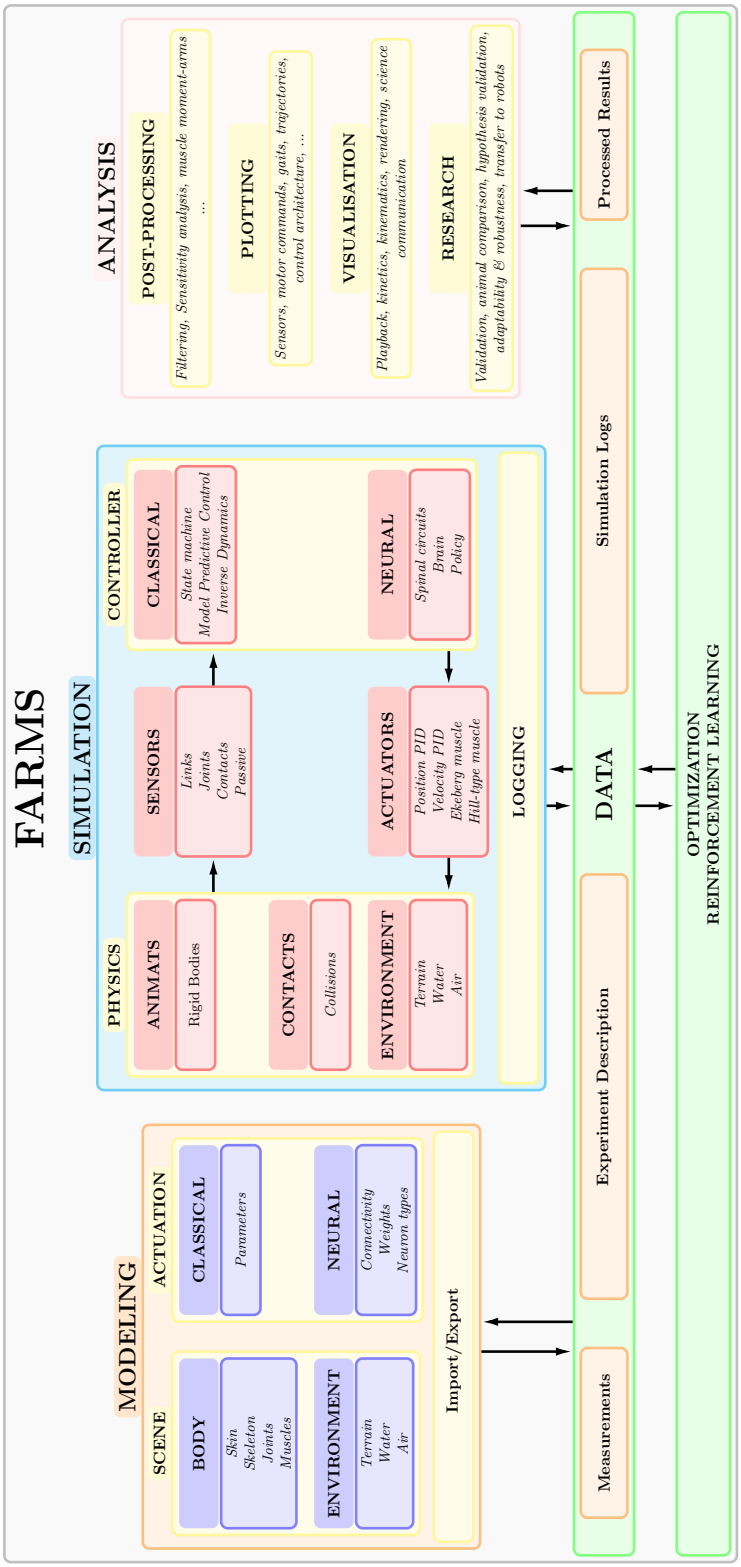


Figure 1. Overview of the modular FARMS architecture. Primary stages of a neuromechanical simulation: modeling, simulation, and analysis. FARMS establishes a standardized data structure that facilitates seamless interoperability across these stages. This structure supports the integration of external libraries, allowing them to interface directly with FARMS and access core neuromechanical simulation data.

- **Processed Simulation Data:** After the simulation runs, processed data is generated to facilitate analysis and interpretation. The Hierarchical Data Format (HDF5) format is used to store and organize this data, effectively reducing disk space usage (***The HDF Group, 2023***). The run-time data and simulation results saved to disk share the same structure, enabling researchers to verify exactly how the simulation behaved during post-processing using the same interface as for writing the controller. From this data, derived metrics and visualizations that summarize the experiment's outcomes can be generated, making it easier to compare results across different simulations and validate hypotheses.

The modularity of FARMS is further enhanced by the seamless interaction between these data blocks. Each module within FARMS can read from and write to the data block using the interface provided by farms_core, allowing different stages of the simulation to be revisited or modified without needing to redo the entire process. For example, a user can adjust parameters in the configuration file and rerun only the simulation stage, using the existing data from the modeling stage.

By systematically saving and organizing data, FARMS not only supports reproducibility but also encourages collaboration and knowledge sharing within the research community. The standardized data formats and comprehensive logging ensure that simulations can be easily shared and reproduced by others, contributing to a more collaborative and transparent research environment.

Modeling

A neuromechanical simulation can be described by the following minimum elements: (i) animal or robot bodies, where in this case the body represents both the geometric and dynamical properties, (ii) a controller, and (iii) the environment with which the animal or robot interacts. Modeling each of the three elements is crucial for successfully running neuromechanical simulations but is often taken for granted to be readily available. Moreover, creating models is an iterative process, considering that models are regularly modified when new measurements or experimental data are available. This implies an important requirement for rapidly reconfiguring the models when a simulation fails or does not match with experimental data.

Animat

An experiment can consist of one or more bodies, specifically animals, robots, or both. As models of robots and animals are treated the same way in FARMS, they will be referred to as *animats* hereafter. An animat is defined by several key elements that collectively describe the structural and dynamical properties of its physical body:

- **Rig:** A collection of links and joints that form the structural frame to support and define the entire body.
- **Link:** An individual element in the rig, characterized by its location (position and orientation). It may include an inertial element and some geometries.
- **Joint:** A connection between two links that defines how they interact kinematically and dynamically. Joints can be of different types, including fixed, ball-and-socket, slide, hinge, and a "free joint" for floating bodies. Joints can also be constrained to mimic physical limitations, such as ligaments in animals or mechanical stops in robots, to restrict the range of motion.
- **Inertial element:** An object that describes an individual link's inertial properties, including mass, centre-of-mass, and inertia.
- **Geometry:** A shape or structure that can have visual and colliding properties interacting with the environment. Each link may possess a set of geometrical elements describing rigid or soft bodies. The visual and colliding properties may differ. Colliding properties sometimes use simplified or primitive shapes that approximate the visual ones to facilitate the collision handling performed by the physics engine.

- **Actuator:** An element that applies forces on the body to generate motion. These can be motors (applying torques or forces at joints) or muscles (applying forces on different links).

These definitions provide the foundation for modeling animats in FARMS, allowing researchers to describe and simulate complex systems accurately.

Even though we represent an animal and a robot under the same description as an animat, the necessary information to define them is obtained differently. Robots are typically designed by humans in Computer Aided Design (CAD) software and usually have a well-defined fixed geometry. However, animal bodies are reconstructed either based on specimen scans [Computerized Tomography (CT) or Magnetic Resonance Imaging (MRI)] or anatomical reference data, and require careful consideration of the intra-animal variability that exists. The scans are then exported as meshes which can be reworked with specialized software, such as Blender (***Community, 2022***). The meshes are then referenced in the desired simulation description file. Making use of common standards for robot simulation, FARMS relies primarily on the Simulation Description Format (SDF), originally developed by the Open Source Robotics Foundation for the Gazebo robot simulator (***Koenig and Howard, 2004***). This format is widely used for robotics simulations and contains all the elements required to define the animat models in FARMS. Different views of the information contained within these files are illustrated in ***Figure 2***. While SDF is the main format used for all the models shown in this article, the modularity of FARMS allows integrating support for other formats as well. For example, preliminary support is already present for Unified Robotics Description Format (URDF) and MuJoCo Format (MJCF), two other commonly used formats in robotics simulation originally developed for Gazebo and MujoCo (***Todorov et al., 2012***) respectively. *OSIM* and *WBT* are some additional examples of frequently used file formats developed for OpenSim (***Seth et al., 2011***) and Webots (***Michel, 2004***) respectively. While these formats are not officially supported yet, it is possible to convert them to be compatible with FARMS (see farms_osim), and further integration remains a future possibility.

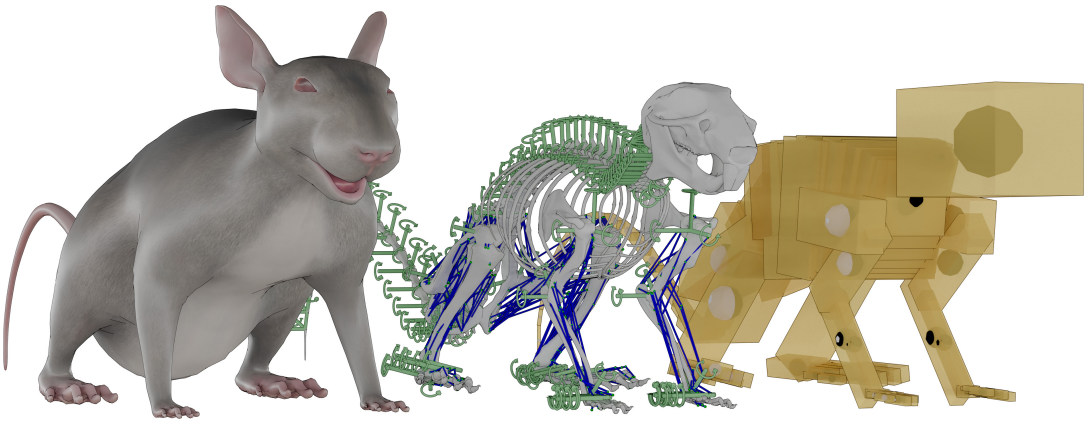


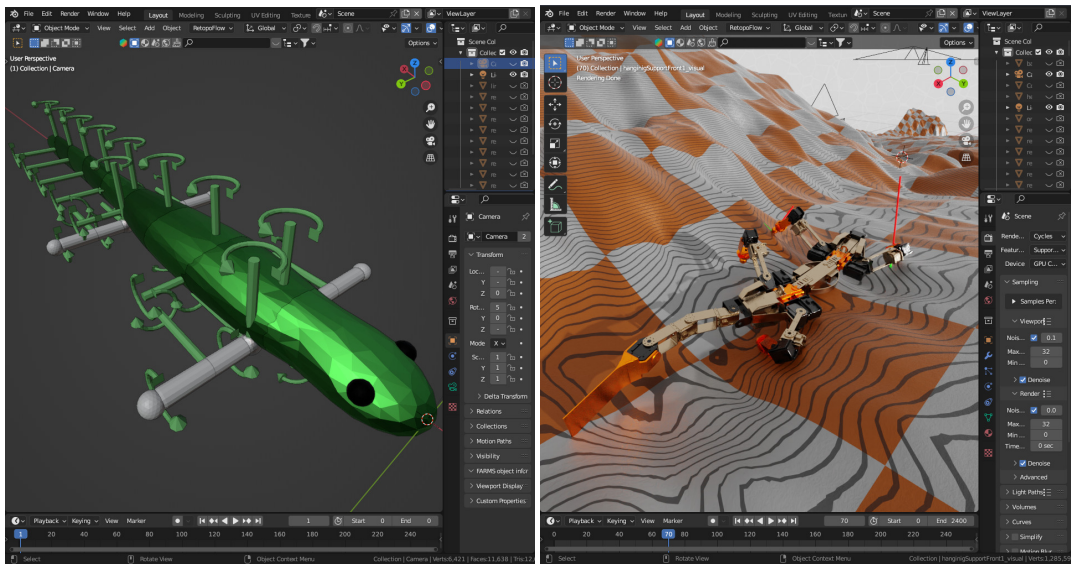
Figure 2. Animat model in FARMS From left to right visualizes the following elements of a mouse model (i) skin, (ii) skeleton, joints and muscles, (iii) inertials (inertia tensor, mass and center-of-mass). Segments of skin, skeleton, or simple primitive shapes can be chosen as collision elements to model interactions with the environment.

Blender add-on

Blender (***Community, 2022***) is a full-featured software for 3D modeling, animation, and rendering. It provides a powerful toolkit for creating complex visuals and animations. The farms_blender package is a Blender add-on, providing an interface to users to generate the models and visualize completed simulations, as shown in ***Figure 3***. SDF, URDF and MJCF are based on XML and are typically written in a text editor. The interface provides the ability to set all the links' poses and

geometries, the joints' poses, as well as the inertials through an interface to facilitate this step. Visual guides can help with validation, as shown in ***Figure 3a***.

The add-on was also developed to fully replay the results of the simulation in Blender, as shown in ***Figure 3b***. This provides a useful interface for reviewing the results of a simulation, showing and overlaying additional information such as interaction forces with the environment, and rendering the simulation results to video for high-quality publications.



(a) Modeling **(b)** Simulation visualization and rendering

Figure 3. Blender for modeling and rendering using the farms_blender add-on. The modeling example shows the salamander with a display of the joints' locations and their rotations. The rendering on the right shows a view of a simulation of Krock, a crocodile-like robot, in an uneven terrain after running the simulation and logging data to allow playback within Blender.

Controller

A controller defines the elements that send the commands to the actuators that will ultimately determine how the physical elements move over time. To study animal and robotics movement, several different control strategies are possible. The possible actuation methods currently are (i) model based control and (ii) neural control. Model based control methods are mainly derived from control theory developed for robotics. These include methods like inverse dynamics, state-machines and Model Predictive Control (MPC). Our expertise relies mainly on neural control, which is why controllers based on neural circuits with Central Pattern Generators (CPGs) and sensory feedback loops are mainly described and presented in the following sections. The use of neural controllers based on deep neural networks using Reinforcement Learning (RL) is also common in recent works. Naturally, the framework grants the ability to implement the other types of controllers as well.

Neural Networks

Neural control typically refers to employing neural network models, whether based on biological principles or machine learning techniques, to generate control signals for movement. This includes biologically inspired CPGs, spiking and non-spiking neural networks, and artificial neural networks. Modeling these systems involves specifying the neuron types, their biophysical or algorithmic properties, and their connectivity, all of which require proper parametrization to generate appropriate behaviors related to complex motor tasks.

Currently, there already exist well-established libraries to build artificial neural networks, such

as Pytorch (***Paszke et al., 2019***) and Tensorflow (***Abadi et al., 2016***). Biological networks can be broadly classified into spiking and non-spiking neural networks. For spiking networks, there are open-source libraries such as NEST (***Gewaltig and Diesmann, 2007***), Brian (***Stimberg et al., 2019***) or Neuron (***Hines and Carnevale, 1997***). For sparse, non-spiking networks, there is no library to the best of our knowledge that supports modeling and simulation of locomotion networks. As part of FARMS, we developed a platform and simulator agnostic sparse neural network software package called farms_network, which provides commonly used neural models for locomotion circuits. More precisely, it implements standard neural models at different levels of abstraction which have been commonly used in past research, such as phase oscillators (***Kuramoto, 1975***; ***Ijspeert et al., 2007***), Hopf oscillator (***Righetti et al., 2006***), Matsuoka model (***Matsuoka, 1985***), FitzHugh–Nagumo model (***FitzHugh, 1961***), Leaky-Integrator model (***Ijspeert, 2001***), Hodgkin-Huxley model (***Hodgkin and Huxley, 1952***) and non-spiking activity-based model (***Ermentrout, 1994***). ***Figure 4*** shows an example of sparse locomotor network based on ***Danner et al.*** (***2017***), modeled and visualized using farms_network.

While the models implemented in farms_network have no external dependencies, it is straightforward to use the implementations from the well-established libraries for artificial and spiking neural networks, as done in ***Pazzaglia et al.*** (***2025***).

Actuation

In FARMS, the controller also includes the actuation system, which constitutes the motor dynamics for a robot or the muscle for an animal. In the case of animal locomotion, the biomechanics can play a crucial role in generating the proper gait patterns. Muscles are not a feature often included as part of physics engines as most of them are in practice built for the gaming and robotics communities.

To address this gap, we developed the farms_muscle package, to provide users with different muscle models. Two particular types of muscles conventionally used in literature have been implemented, variations of Hill-type muscle models with options to choose between rigid and compliant tendons in a computationally efficient manner (***De Groote et al., 2016***; ***Geyer and Herr, 2010***; ***Millard et al., 2013***), and the Ekeberg antagonistic muscle pair model (***Ekeberg, 1993***).

As FARMS continues to evolve, the modularity of the framework allows for the future integration of additional muscle models and further refinements to the actuation systems.

Environment

In FARMS, the environment encompasses both the arena, which defines the terrain geometry, and the interaction properties between the bodies and the surrounding media. The terrain can vary in complexity, ranging from flat surfaces to more rugged landscapes, and can also include obstacles.

Interaction with the terrain is often rigid, as is typical for land-based locomotion. Most physics engines designed for robot simulations primarily support rigid-body interactions, making them well-suited for simulating this type of environment. Depending on the chosen physics engine, FARMS enables flexible interaction models for different terrains. For instance, Bullet (***Coumans and Bai, 2016***) provides robust support for rigid-body dynamics, ensuring accurate simulations of hard surface interactions. Conversely, MuJoCo (***Todorov et al., 2012***) offers advanced features for soft-body interactions, allowing for more detailed modeling of soft contact dynamics, such as deformations and friction. FARMS provides an interface that allows users to take advantage of these engine-specific capabilities, facilitating accurate neuromechanical simulations across varied terrains and environments.

The framework is also intended to accommodate various complex environmental media, such as water, sand, and air, with each medium characterized by its unique interaction dynamics. This flexibility allows for the simulation of diverse locomotion and interaction scenarios, enabling researchers to study behaviors across a range of physical environments.

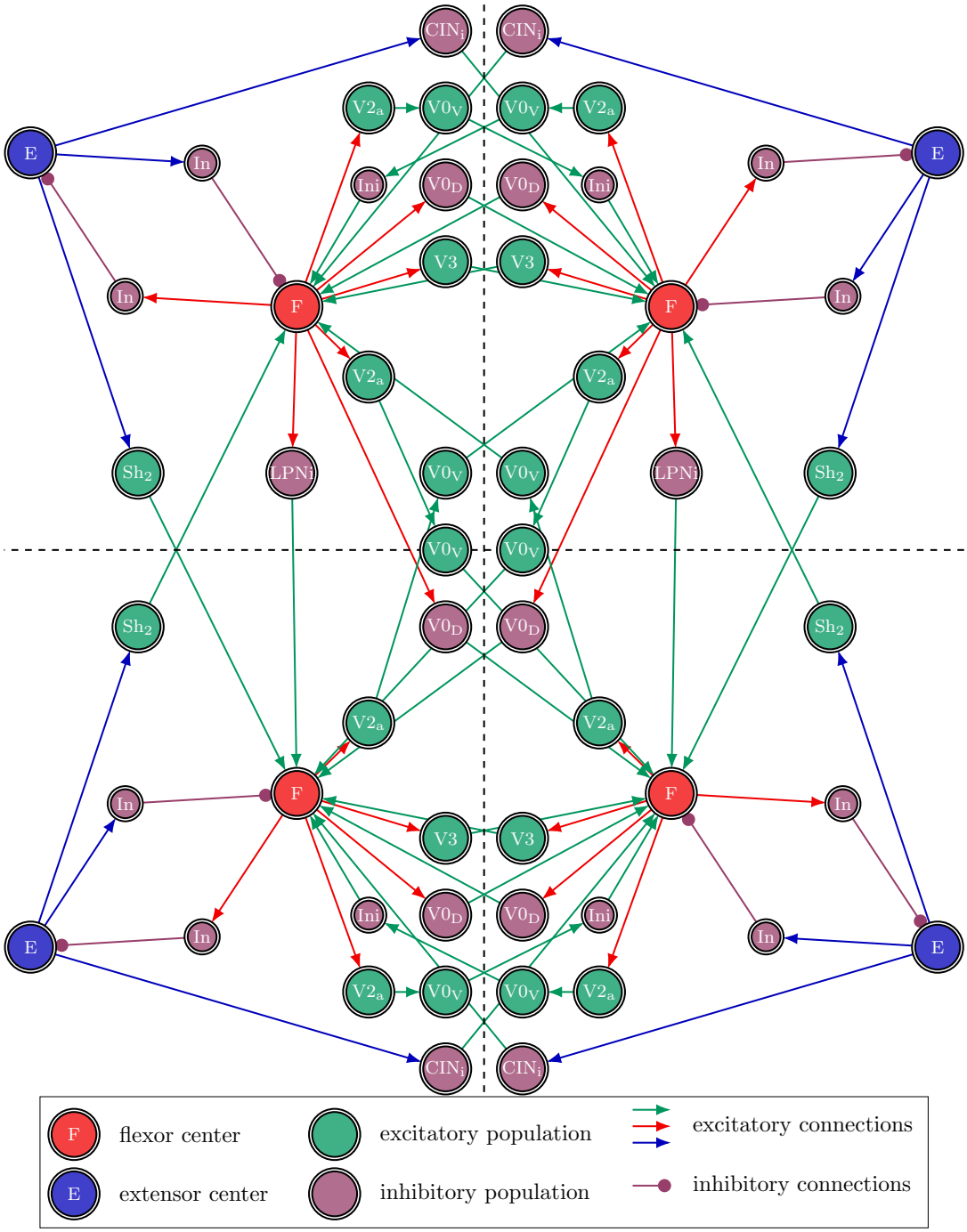


Figure 4. Example of neural network generation and visualization in FARMS. The network is a computational model of central spinal circuits for quadruped limb coordination proposed by ***Danner et al.*** (***2017***).

Simulation

Once the animat, controller, and environment are fully defined, the next step is to perform numerical simulations. For neuromechanical simulations, this primarily involves executing iteratively the controller commands of the animat and performing a forward pass of the physics simulation step of the entities within the simulated world. During the physics step, the physics constraints must be solved, and the dynamics of the system integrated over time. Leveraging existing physics engines

and available frameworks, FARMS provides wrappers that help bridge modalities of the simulation block, illustrated in ***Figure 1***.

Physics engine

The physics engine is the core of a neuromechanical simulation. The common steps performed by a physics engine are described by ***Geijtenbeek*** (***2013***) as follows:

- **Collision detection** checks if the body intersects with itself or other physical objects in the scene.
- **Forward dynamics simulations** compute the linear and angular accelerations of each simulated body.
- **Numerical integration** updates positions, orientations, and velocities of bodies, based on the previously computed accelerations.

The steps outlined above simulate the articulated rigid bodies and their interactions with rigid terrain. There are several physics engines available, each employing different strategies to perform these three steps. Choosing the appropriate physics engine depends on the features required by the experiment, including the accuracy and stability of the simulation. It is also important to consider that the differences in their implementations can affect the accuracy, stability and computation time of a simulation.

A valuable feature of FARMS is its potential to provide a common interface to multiple physics engines. FARMS currently supports Bullet (***Coumans and Bai, 2016***) with farms_bullet and MuJoCo (***Todorov et al., 2012***) with farms_mujoco as its primary physics engines. The Bullet and MuJoCo physics engines were mainly selected as primary physics engines in FARMS for their ease of use, extensibility, ability to install on all major desktop platforms, capability to simulate rigid multi-bodies in reduced coordinates for improved stability with many-joint models, and their computational speed when solving simulations of animat locomotion.

Bullet (***Coumans and Bai, 2016***) stood out for its ease of use and the simplicity of creating new models, similarly to Gazebo, as it supports standard description formats. Originally, FARMS only supported the Bullet engine, which stood out compared to other engines such as ODE (***Smith, 2006***) for its ease of use through the Python interface and for handling hard collisions reliably without requiring fine-tuning of the contact parameters. While this engine lacked essential features for neuromechanical simulations, such as muscle and fluid models, they could be implemented within FARMS in the farms_bullet package. However, a limitation of Bullet which could not be mitigated is that it implements joint damping explicitly, rather than using an implicit formulation. This led to numerical instabilities for high values of damping, which is an essential feature needed for simulating biomechanical models.

MuJoCo (***Todorov et al., 2012***) was released as open-source in 2022 and features an implicit implementation of joints damping, facilitating tuning and simulation of muscle-controlled animats. For this reason, the farms_mujoco package was implemented to support MuJoCo and seamlessly migrate certain experiments that required this feature.

MuJoCo also provides some additional beneficial features, such as simple muscle and fluid models. On the other hand, contacts are modeled as soft constraints. While this is a drawback for simulating rigid robot models with hard contacts, the advanced soft-contact model can be desirable to account for the deformation of the skin in animal models.

FARMS supports both headless operation, enabling users to run large batches of simulations efficiently, and a graphical user interface (GUI) that is accessible to beginners as results are visually accessible during execution. Currently, the GUI directly exposes the one provided by either Bullet or MuJoCo, depending on which engine was selected by the user for the experiment. This dual capability ensures that FARMS is suitable for both advanced researchers and those new to the field, making it a versatile tool for diverse user groups.

As research requirements evolve and new engines emerge, interfaces to additional simulation platforms and physics engines could be developed in a similar way to the MuJoCo and Bullet integrations.

Simulation step

During each simulation step, FARMS follows a structured process consisting of five sequential stages: Sensors, External forces, Controller, Actuators, Logging, and the Physics engine step. This process repeats iteratively until the simulation reaches a predefined termination condition, e.g. the completion of the simulation duration.

- **Sensors:** At the start of each simulation step in FARMS, the sensor stage collects the state of the simulation as provided by the physics engine for the sensors defined during the modeling phase. This includes exteroceptive signals such as contact forces and ground reaction forces, as well as proprioceptive signals like muscle spindle feedback, joint angles, and velocities. These sensory data points are essential for informing the external forces and controller stages, providing real-time information on the system's state for subsequent computations. The collected data is stored within a dedicated data interface, allowing efficient access without re-querying the physics engine, thereby minimizing redundant computations and improving performance.
- **External forces:** In this stage, the external forces acting on the animat are defined. These can include user-defined perturbations, such as custom-applied forces or torques, as well as environmental forces derived from predefined properties or dynamic models, such as fluid dynamics or wind resistance. The computation of external forces can depend on sensor data collected in the previous stage, such as velocity, contact forces, or positional information, which inform the calculation of dynamic environmental forces. For instance, drag forces encountered during swimming are computed based on the velocity of each link in contact with the simulated fluid. Using the velocity data from the sensors, FARMS calculates the resulting drag forces, which are then applied as external forces to the animat. As with sensor data, the computed external forces are logged and made available for control and analysis.
- **Controller:** The controller governs the animat's behavior, determining its actions at each simulation step. Depending on the chosen control approach specified during the modeling phase, this could involve either a model-based control strategy or a neural control system. FARMS provides the controller access to all the sensors and external forces data collected at the previous stage, allowing it to compute the outputs as necessary for the actuators in the next stage. Any user is free to implement their own controller as needed for their application, although the farms_network package is available for facilitating the development of neural controllers. A neural controller usually uses its own solver to integrate the neural dynamics or perform other necessary computations to generate the previously stated outputs, and can operate with an internal timestep to handle stiff equations.
- **Actuators:** The outputs of the controller dictate the actions of the actuators. These outputs include (i) muscle activation signals, (ii) desired joint positions and velocities, and (iii) desired joint torques. FARMS applies the appropriate method for each actuator as defined during the modeling phase, ensuring proper actuation for both robotic and biomechanical systems.
- **Logging:** For each of the previous stages, FARMS provides data interfaces that automatically log the sensor, external forces, controller, and actuator data. This ensures traceability of the simulation state, providing each stage access to past data, which can be useful for adaptive control and feedback mechanisms. The logged data can also be stored to disk, facilitating post-processing and detailed analysis. This logging feature allows users to revisit and study any point in the simulation, facilitating the extraction of meaningful insights for further research and development.
- **Physics engine step:** Once the previous stages are complete, FARMS updates the physics engine with the final outputs from the actuators and external forces. The physics engine then

handles the integration step, typically in the order of 1-5 [ms], solving physical constraints and updating the dynamics over time. After completing the integration, the stages are repeated for the next iteration, continuing until the simulation reaches its predefined termination.

Custom physics

As part of FARMS, we provide access to both phenomenological and advanced fluid dynamics models. For simplified fluid dynamics, we offer a drag model implemented in both the farms_bullet and farms_mujoco packages, which approximates interactions with fluids through computationally efficient drag forces. For more sophisticated simulations, preliminary support for fluid dynamics based on Smoothed-Particle Hydrodynamics (SPH) methods is available in the farms_sph package. This package integrates the PySPH library (***Puri et al., 2013***; ***Ramachandran et al., 2021***), enabling two-way coupling between fluid dynamics and rigid-body simulations, similar to approaches described by Angelidis et al. (***Angelidis et al., 2022***).

The drag model applies forces and torques based on the following equations:

$$\vec{f}_i = \vec{c}_{f,i} \odot \vec{v}_i^2 \tag{1}$$

$$\vec{\tau}_i = \vec{c}_{\tau,i} \odot \vec{\omega}_i^2, \tag{2}$$

where f_i is the force and τ_i it the torque applied on link i, v_i and ω_i are respectively the linear and angular velocities of link i in local frame, and c_f and c_τ are drag coefficients, which are multiplied to the velocities using the element-wise Hadamard product.

Buoyancy forces are also supported and calculated based on the provided volume of each link, generating vertical forces as appropriate. In both the drag model and the SPH-based integration, fluid-induced forces are calculated independently from the rigid-body physics engine. These forces are applied as external forces during each simulation step, with the rigid-body physics engine responsible for integrating the equations of motion. This method ensures compatibility across engines by treating fluid interactions as additional external inputs. While support for sand and air simulations is not yet available in FARMS, a similar methodology could be used to implement these media, leveraging PySPH's existing capabilities for simulating sand and gas dynamics (***Ramachandran et al., 2021***). Development of the SPH integration is still in progress, with plans to extend support to additional SPH models, such as those provided by SPlisHSPlasH (***Bender, 2017***), in future updates.

Optimization

A key aspect of modeling is estimating model parameters. The advantage of neuromechanical simulations is that they allow for the investigation of the role of the parameters for various components of an experiment, such as the morphology and controller of an animat, or the role of the environment and its physics. In engineering and research applications, it is common to seek a specific behavior out of the simulation, but finding the parameters to obtain such behaviors can be challenging. Indeed, it can be difficult to measure certain neural or muscular parameters from the animals or impractical to directly test on robot hardware, while simulations can leverage evolutionary algorithms to tune these parameters. For this reason and inspired by (***Auerbach et al., 2014***), we implemented support for evolutionary algorithms in farms_evolution.

Evolutionary algorithms have been employed in a large number of works including studies of lampreys (***Ijspeert and Kodjabachian, 1999***), salamanders (***Ijspeert, 2001***), ants (***Guo et al., 2018***) and humans (***Dzeladini et al., 2014***). More precisely, ***Ijspeert and Kodjabachian*** (***1999***); ***Ijspeert*** (***2001***) use Genetic Algorithms, ***Guo et al.*** (***2018***) uses Covariance Matrix Adaptation Evolution Strategy and ***Dzeladini et al.*** (***2014***) uses Particle Swarm Optimization, all of which optimize a single cost function. The farms_evolution packages leverages the Pymoo library (***Blank and Deb, 2020***), which provides a range of algorithms for single and multi-objective optimization, with or without constraints, for optimizing the parameters of a simulation task (***Arreguit et al., 2021***).

Analysis

The analysis block plays a crucial role in extracting valuable insights from the simulation data. As shown in ***Figure 1***, the analysis block is seamlessly integrated into the overall architecture. The logging mechanism incorporated in farms_core ensures that all relevant information used to handle the state of the animat at any point of time within the simulation is recorded, facilitating further analysis and post-processing. With this comprehensive dataset, researchers can for example generate plots of sensor values, evaluate model performance, identify distinct gaits, cluster behaviors, and even replay the results of the simulation to gain deeper understanding without recomputing it.

The ability to replay the simulation within a 3D interface is provided with Blender (***Community, 2022***) through the farms_blender plugin. This enables replaying the simulation by overlaying additional data to provide deeper insights to the researcher, such as collision or external forces applied on the animat. High-quality visualization tools go a long way in helping researchers in understanding the high dimensional information gathered during simulation. By leveraging this capability, scientists can quickly explore various what-if scenarios, optimize their models accordingly and generate high quality renders for publication purposes.

Automatically having access to the logged data also facilitates the ability to perform comparative studies with the animals or hardware experiments. By analyzing the simulated behavior alongside real-world observations, researchers can refine their models and improve their predictive capabilities. By providing a comprehensive framework for data logging, post-processing, visualization, and comparison with animal data, FARMS empowers researchers to gain valuable insights from their simulations, ultimately driving advancements in neuromechanical modeling.

Demonstrations

To highlight the features and capabilities of FARMS, we present a number of experiments with different animats, controllers and environments in the sections that follow.

Robots

We support a set of robots that are used in the Biorobotics laboratory at EPFL. An illustration showing the different robots that we work with is found in ***Figure 5***. These robots are primarily based on eel and salamander-like morphologies, and in the future we plan to support additional robots morphologies which are actively being designed and made available in the community.

We replicated the behaviors presented in ***Ijspeert et al.*** (***2007***), where a salamander-like robot, Salamandra Robotica II (***Crespi et al., 2013***), is capable of transitioning from swimming in water to walking on land and vice versa. To generate these different locomotor patterns, the controller uses a network of weakly-connected oscillators, based on central pattern generators, and replicated from the work of ***Ijspeert et al.*** (***2007***) which simulates the spinal cord dynamics.

The simulation results obtained for transitions from swimming to walking are shown in ***Figure 6***. This experiment was performed using the MuJoCo physics engine. An arena featuring a ramp and a body of water was created such that the simulated robot could go from one media to the other. To demonstrate how FARMS can include custom physics through MuJoCo's external forces, the forces from the water are applied using a simple drag model provided by ***Equation 1*** and ***Equation 2***, and a buoyancy force allows the robot to swim at the surface of the water given its mass density. The descending drive signals were modulated based on passing a threshold value of contact with the ground such that walking patterns would be generated when the robot is in contact with the ramp. Based on the original work of ***Ijspeert et al.*** (***2007***), we find in ***Figure 6*** that the simulated robot is successfully capable of swimming in the water and switching to a walking pattern once it reaches the ground and is then capable of walking itself out of the water. All the features demonstrated here are available through FARMS, including the robot model, the ramp arena, the custom fluid physics, and the oscillator controller.

Figure 5. Examples of robots in FARMS. These robots have also been featured in literature; Pleurobot in ***Karakasiliotis et al.*** (***2016***), Orobot in ***Nyakatura et al.*** (***2019***), K-rock in ***Melo et al.*** (***2017***, 2023), Salamandra Robotica II in ***Crespi et al.*** (***2013***), Agnathax in ***Thandiackal et al.*** (***2021***) and Amphibot in ***Crespi et al.*** (***2005***); ***Crespi and Ijspeert*** (***2006***). Pleurobot and Salamandra Robotica II have been used in previous works to study salamander locomotion (***Karakasiliotis et al., 2016***; ***Ijspeert et al., 2007***). The salamander model shown in the center allows for a size comparison between these robots and the studied animal.

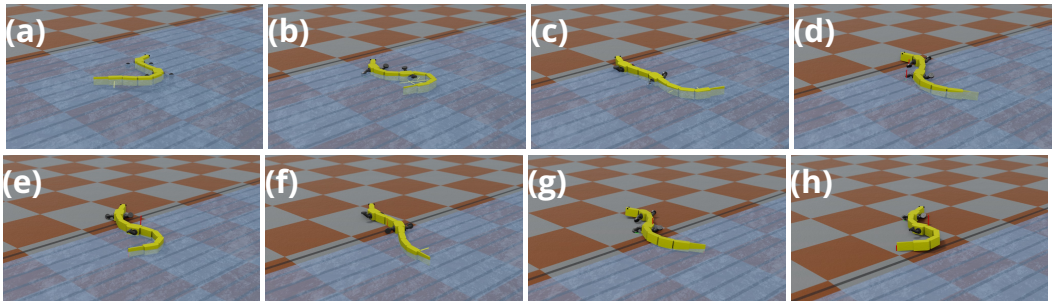


Figure 6. Salamandra Robotica II performing a swim-to-walk transition. Sequence of rendered snapshots shown at different times separated by 1.5 [s]. The robot is swimming from right to left in 6a and comes into contact with the ramp in 6b. In 6c the robot transitions from swimming to walking by changing the descending drive commands and the limbs activate. The trunk oscillators synchronize with the limb oscillators, inducing the body to switch from a travelling wave to a standing wave. From 6c to 6h, the robot has switched to a trotting gait and walks out of the water.

By adapting the proposed oscillator controller to the morphology of the other robots, we show in ***Figure 7*** that walking and swimming gaits can also be obtained with the K-rock (***Melo et al., 2017***) and Pleurobot (***Karakasiliotis et al., 2016***) robots models using an adapted network architecture. The particular difference between these robots and Salamandra Robotica II is that they have a higher number of Degrees of Freedoms (DoFs) in the limbs and different numbers of joints within the trunk as well. While still using a double chain of oscillators for the trunk joints, the network is adapted by also using pairs of oscillators for each joint of the limbs.

Salamander

Extending the salamander-like robots to animal simulations, a biological salamander model was designed based on the work of ***Harischandra et al.*** (***2010***). By leveraging the Blender 3D modeling software with the farms_blender package, we provide tools for generating a 3D shape, as illustrated in ***Figure 8***. This resulted in an approximate, but comparable shape to the animal. The mass was

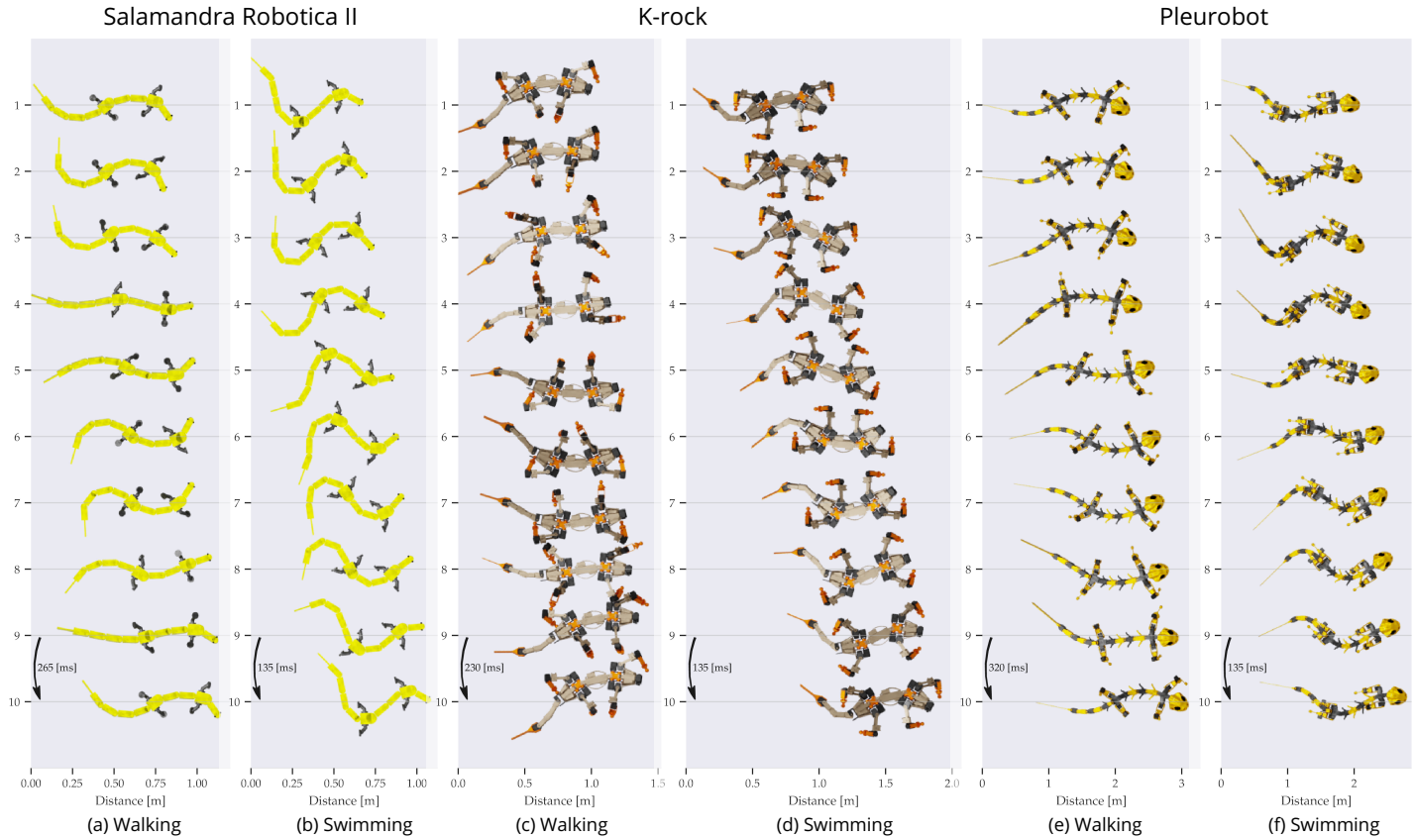
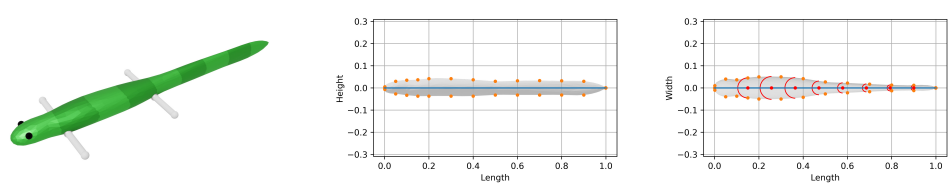


Figure 7. Salamandra Robotica II, K-rock and Pleurobot walking and swimming gaits in simulation. The robots are shown from above with snapshots along time. The spines for each robot exhibit travelling waves which allow them to propel themselves forward in water.

computed assuming constant density close to water such that the models could float slightly below the surface. The resulting salamander model is 10 [cm] long and weighs 3 [g]. It contains 11 body joints allowing the body to bend laterally and 4 DoF for each limb, including 3 DoF at the interface with the body, and one DoF at the elbow/knee.



(a) Salamander model morphology **(b)** Side **(c)** Top

Figure 8. Salamander shape validation and a render of its final morphology. In the two views, side and top, the orange points represent the measurements obtained from the animal. The greyed area represents the final shape of the generated 3D Model. The red points represent the position of the joints along the body. Spherical cuts are applied to the body such that no mass overlap is present when the joints are rotated, which is a useful property for swimming applications as density remains constant.

Similarly to the robots demonstrations, a controller was implemented to showcase walking and swimming using an oscillator network, as shown in ***Figure 9***. However, contrary to the robots presented above in position control, each of the DoF for the salamander is modeled as a revolute joint

controlled by a flexion-extension muscle pair model proposed by ***Ekeberg*** (***1993***) and governed by:

$$\tau_j = \underbrace{\alpha_j(M_{F,j} - M_{E,j}) + \beta_j(M_{F,j} + M_{E,j})\Delta\varphi_j}_{\tau_{active,j}} + \underbrace{\beta_j\gamma_j\Delta\varphi_j + \delta_j\dot{\varphi}_j}_{\tau_{passive,j}}, \tag{3}$$

where, similarly to ***Ijspeert and Kodjabachian*** (***1999***); ***Ijspeert*** (***2001***), α_j represents the active gain, β_j the stiffness gain, γ_j the intrinsic stiffness, δ_j the damping coefficient, $M_{F,j}$ and $M_{E,j}$ the neural output activities for the flexor and extensor obtained from a pair of oscillators, $\Delta\varphi_j$ the position and $\dot{\varphi}_j$ the velocity for each joint j. In particular, we differentiate between the active torque $\tau_{active,j}$ and passive torque $\tau_{passive,j}$.

Just like for the previous demonstrations, this model is also capable of exhibiting both swimming and walking patterns based on a descending drive signal. The main difference is that all the robots in the previous experiments were position controlled, while the joints of the salamander are torque controlled using the muscle implementation from ***Equation 3***. In contrast to the Bullet engine, we found that the MuJoCo physics engine was beneficial for properly simulating the dynamics of the Ekeberg muscle model thanks to its implicit implementation of damping. Indeed, increasing to high values of damping δ_j would lead to stable expected behaviors in MuJoCo, while it would lead to jitters and numerical instabilities in the Bullet physics engine.

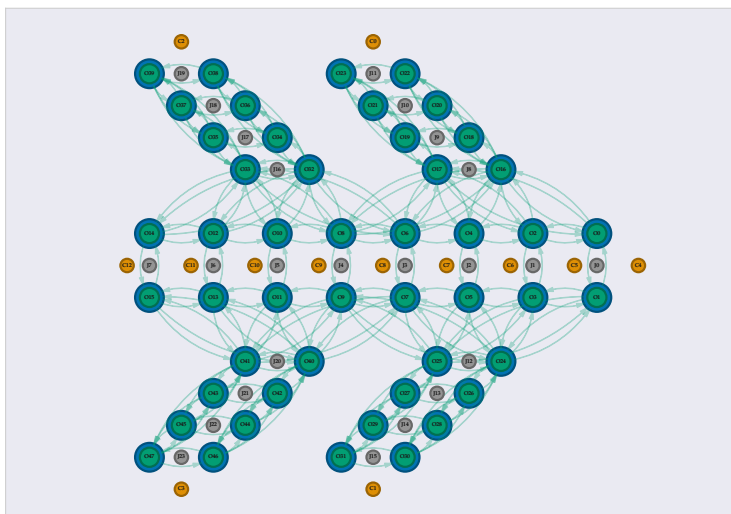


Figure 9. Salamander connectivity for an oscillator network. The network is inspired from the work of ***Ijspeert et al.*** (***2007***). Here, we explore a connectivity based on weak couplings where oscillators are only connected to neighbors. One of the key differences with ***Ijspeert et al.*** (***2007***) is that the limb oscillators have no direct couplings between them. Although it is known that long-range connections between the limb CPGs exist, here we found that this connectivity was sufficient for the salamander model to achieve walking and swimming patterns, as shown in the demonstrations below.

In ***Figure 10***, We demonstrate how the salamander can cross a river, and thus perform walking-to-swimming and swimming-to-walking transitions. In that experiment, a water current is also present and the appropriate descending signals are computed to drive the model to follow a path to the other side of the river while switching to the appropriate gait.

Combining the oscillator network with muscle models was found to yield stable walking gaits. We demonstrate in ***Figure 11*** that the salamander is capable of walking over a rough terrain. In this experiment, the passive properties of the muscles are useful for providing compliance to the limbs and permit the limbs to interact with the ground without predefined trajectories. This model is walking blindly without any knowledge of the surrounding terrain, but shows it can handle it in a robust manner.

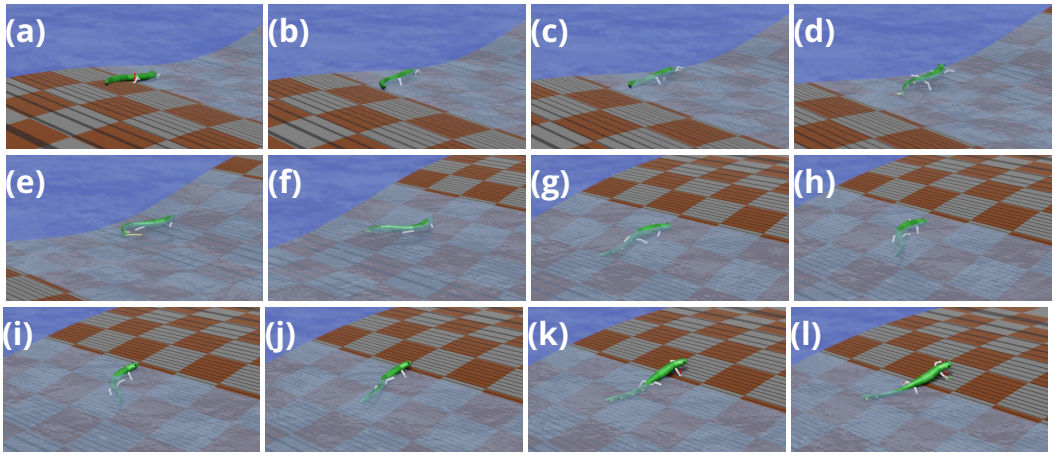


Figure 10. Simulated salamander crossing a river. Sequence of rendered snapshots separated by 1.0 [s]. A salamander is simulated in an arena featuring a river, where a ground created using a heightmap with two banks separated by a river is found. The water has a velocity of 3 [cm/s] flowing from left to right. The salamander is initialized on one side of the river and moves towards the other side. In 10a, the salamander comes into contact with the water, but does not switch to a swimming gait until 10d, simiarly to the experiment with Salamandra Robotica II in ***Figure 15***. From 10d to 10g it swims to the other side pushed by the water flow until it comes into contact with the ground and switches to a walking gait. From 10g to 10l it walks up and out of the water.

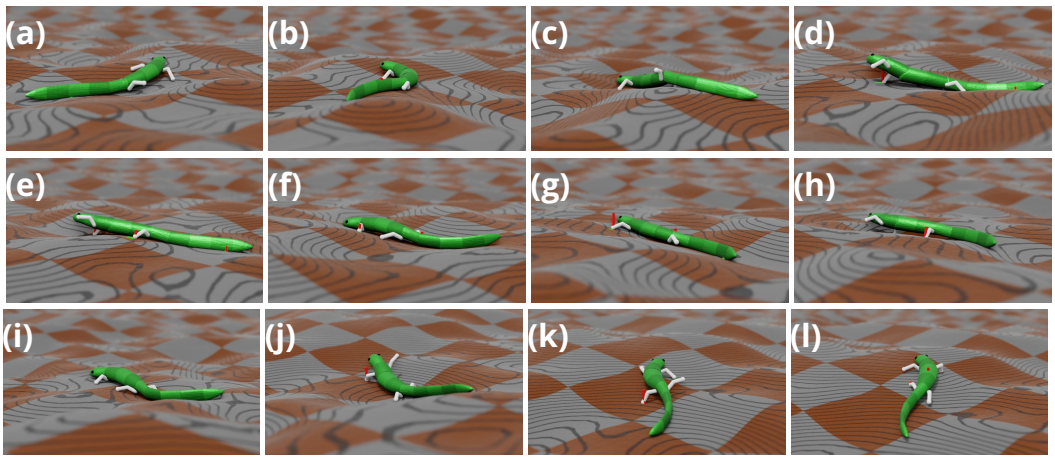


Figure 11. Simulated salamander walking in rough terrain. Sequence of rendered snapshots separated by 5.0 [s]. A salamander crosses a terrain made with a heightmap generated with Perlin noise. Using the network shown in ***Figure 9***, the salamander is capable of robustly walking across the terrain. Its ability to navigate this terrain is jointly due to its sprawling posture and passive muscle properties that allow it to bend along the terrain deformation.

Snake

To extend the salamander results, we also demonstrate how the oscillator network and the passive properties of a muscle model (***Ekeberg, 1993***) can lead to obstacle-based locomotion. A flat arena is set with regularly placed vertical circular pegs with a fixed radius along a grid, as shown in ***Figure 12***. The friction coefficient between the body and the arena is isotropic and set to 0.2 so that the snake model can slip with ease, but can not advance without pushing on the surrounding pegs. Using a simple double-chained oscillator network for a limbless snake-like body, i.e. the same controller as the salamander but without the limb oscillators, the model is capable of traversing the obstacle configuration. As the model escapes the pegs area, it is unable to move beyond as it cannot push itself against obstacles anymore. Additional experiments for different arena configurations with

different values of pegs radius and spacing are provided in the appendix in ***Figure 17***, ***Figure 18*** and ***Figure 19***. Each of these experiments were performed with no changes in descending signals using the same oscillator network with fixed parameters. This showcases how passive properties of the body can be simulated and interact with the environment with many points of contacts, both with the ground and the pegs, when simulated using the MuJoCo physics engine.

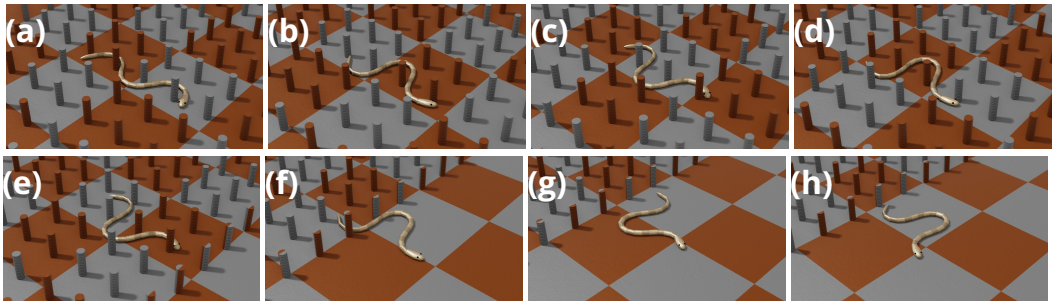


Figure 12. Snake obstacle-aided locomotion in arena 1 Sequence of rendered snapshots separated by 2.5 [s]. The pegs have a diameter of 4 [cm] and are separated by 21 [cm] from centre-to-centre.

Mouse

This section illustrates how FARMS can be applied to develop complex musculoskeletal models from animal data. As a case study, we present the development of a whole-body musculoskeletal model of a mouse, as detailed in ***Tata Ramalingasetty et al.*** (***2021***). Creating a musculoskeletal model involves: (i) modeling the skeletal segments (links and inertials), (ii) identifying the relationship between segments (joints) for articulation, and (iii) developing the body musculature (actuators). For the mouse model, 108 skeletal segments were identified, and clean mesh data set were created based on CT scans and sample zoological references. Individual bone mesh segments were then loaded in Blender, and using the farms_blender plugin the relationship between the bones were identified and constraints between them were introduced as joints to create an articulated mouse skeletal system. In total, the mouse skeletal system consisted of 225 DoFs. Next, for each bone, the inertial properties were automatically estimated based on the convex hulls that represent the net volume encapsulated by the skin, soft tissues, muscles, and the bone with a constant density of water (1000 kg/m^3). The mouse model was then exported from Blender using the farms_blender plugin to the simulation description format (SDF) supported by FARMS. Along with the skeletal system, the mouse skin with texture was also made available. Currently, the mouse skin is only for visual purposes and does not influence the dynamics of the simulations.

Modeling the musculature as Hill-type muscles to actuate the skeletal model is a tedious job. It involves recreating muscle geometry spanning over the bones and estimating muscle parameters that dictate the dynamics of force generation in the muscles, as in ***Figure 13*** panels 2 and 3. ***Tata Ramalingasetty et al.*** (***2021***) modeled the detailed musculature for actuating mouse hindlimbs and forelimbs. The study highlights the features of modeling muscles; (i) transferring available muscle attachments and parameters from one model to another (ii) creating muscle paths from CT scan data and (ii) estimating unknown muscle properties using numerical optimization techniques.

Detailed mouse hindlimb musculature was first developed by ***Charles et al.*** (***2016a***). The full skeletal model described in ***Tata Ramalingasetty et al.*** (***2021***) differed in the bone geometry from the Charles model due to differences in the mouse specimens used to develop the respective models. Mesh registration computes a coordinate transformation matrix from any point on one mesh to another. ***Tata Ramalingasetty et al.*** (***2021***) used CloudCompare, an open-source software to perform mesh registration on the hindlimbs of the two mouse models. For complex meshes, it is possible and recommended to include reference landmarks on each of the meshes. Mesh registration feature has now been implemented in farms_blender and allows users to use it directly within

the blender environment. The affine transformation obtained from the mesh registration was then used to transfer the attachment points between the models. This automated pipeline eases the transfer process and limits the errors when working with different animal specimens. ***Figure 13*** panel 1 shows an example of using the mesh registration technique to register the femur bones between the mouse models.

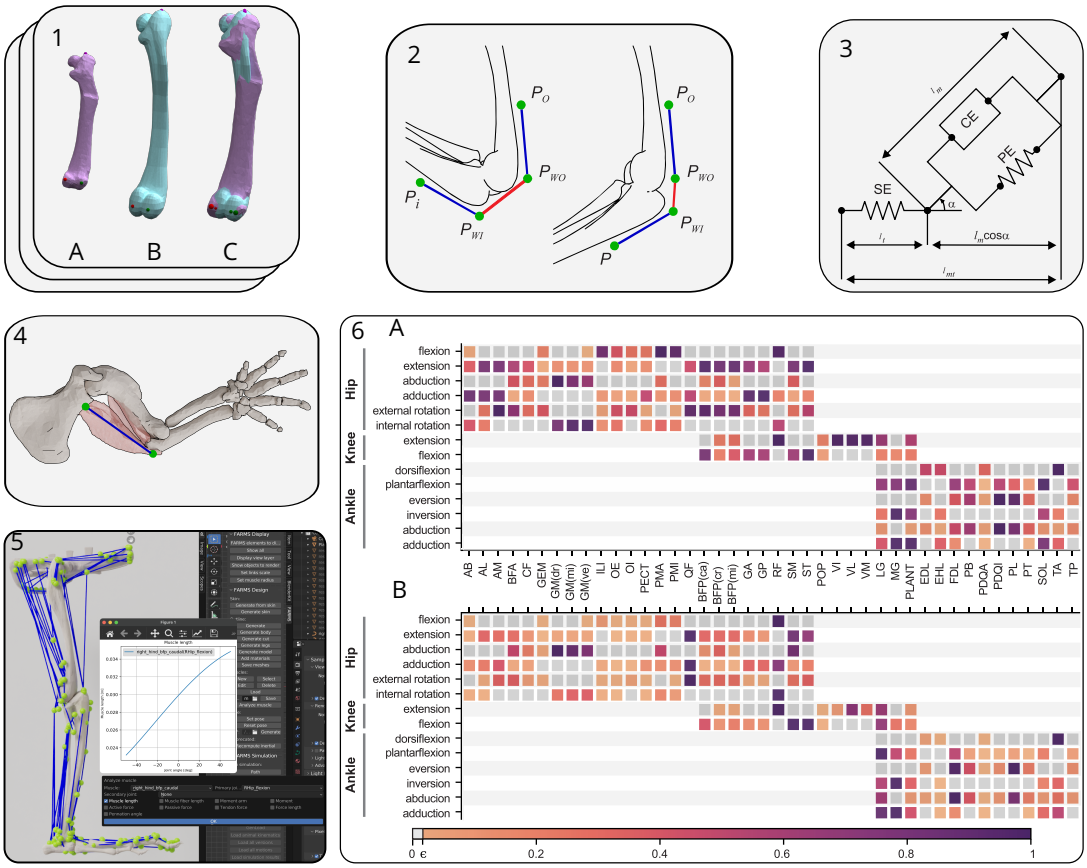


Figure 13. Development of mouse whole-body musucloskeletal model. Panel 1: Example of femur bone mesh registration process to transfer muscle attachment landmarks from femur model described by ***Charles et al.*** (***2016b***) (1A) to whole-body mouse model described in ***Tata Ramalingasetty et al.*** (***2021***) (1B). 1C validates the successful transfer process. **Panel 2**: Polyline approximation of muscle paths used in FARMS to describe muscle paths. The example shows the change in extensor muscle length spanning across the knee joint. (Figure reproduced from ***Tata Ramalingasetty et al.*** (***2021***)). **Panel 3**: Schematic representation of the Hill-type muscle model comprising active and passive force producing elements. **Panel 4**: Identification of muscle paths for forelimb muscles based on CT data from ***DeLaurier et al.*** (***2008***). **Panel 5**: FARMS plugin in Blender to model and analyze musculoskeletal models. The example shows the hindlimb muscusloskeletal model loaded in blender using farms_blender and the feature to analyze generate plots for performing muscle analysis. **Panel 6**: Comprehensive overview of hindlimb muscle and joint relationship based on maximum muscle moment-arms (A) and moments (B). Grey boxes indicate joint functions for joints the muscle spans but has zero influence over. ϵ indicates a very low non-zero positive value.

For the forelimbs, there were no previously existing musculoskeletal models to begin with. In the study by ***DeLaurier et al.*** (***2008***), a three-dimensional atlas of the mouse embryo's forelimb at embryonic day 14.5 was created using Optical Projection Tomography and digital segmentation techniques. Using the mesh registration technique described earlier, we could identify insertion and origin points on the full model forelimb based on the mouse forelimb atlas (***Figure 13*** panel 4). The plugin farms_blender provides the necessary tools to easily add intermediate attachment points for muscles in 3D over the bones to reconstruct muscle geometry based on scans such as the mouse forelimb atlas.

Hill-type muscles are parameterized by the optimal fiber length (l_m^o; the length of muscle-fiber at which the muscle produces maximal active force), the tendon slack length (l_t^s; the length of the tendon below which the muscle transfers no force to the attached bones), the maximum muscle-fiber velocity (v_m^{max}), the pennation angle when the fiber length is at its optimal (α_o), and the maximum isometric force (F_0^m).

For each hindlimb muscle α_o and F_0^m remained the same as reported in ***Charles et al.*** (***2016b***). Length-dependent parameters l_m^o and l_t^s were scaled based on the algorithm described in ***Modenese et al.*** (***2016***). The algorithm has been re-implemented and is now currently available with farms_blender for use in other musculoskeletal model developments. The forelimb musculoskeletal model was the first of its kind for a mouse and different strategies had to be combined to identify forelimb muscle parameters: parameters l_m^o, F_0^m and α_o were extracted and scaled accordingly to the full model from measurements made by ***Mathewson et al.*** (***2012***); an adaptation of the numerical optimization technique formulated by ***Manal and Buchanan*** (***2004***) was implemented in farms_blender to estimate l_t^s. The algorithm now considers all the DoFs a muscle operates on and removes the restriction that muscle fibers only operate in the ascending region of the force-length curve while optimizing for l_t^s.

The fully parameterized whole-body mouse musculoskeletal model was then used to perform a comprehensive analysis to understand the relationship between muscles and joint movements. ***Figure 13*** panel 6 illustrates the mechanical action of each muscle in the mouse hindlimb across all the joints it spans, presented in terms of normalized muscle moment arms and resulting moments. For a more detailed discussion on the analysis, refer to ***Tata Ramalingasetty et al.*** (***2021***).

We adapted the mouse musculoskeletal model developed by ***Tata Ramalingasetty et al.*** (***2021***) to simulate hindlimb locomotion with a reduced set of muscles (***Tata Ramalingasetty et al., 2023***). Here, we reduced the model complexity both at the level of the skeletal system and the musculature. Tools provided by the FARMS framework facilitated the process of model simplification. The modeled skeletal system had 18 degrees-of-freedom (DoFs): 6 of these were actively controlled by Hill-type muscles and the remaining 12 represented passive DoFs modeled as spring-damper systems. Each hindlimb had 3 active DoFs (hip, knee and ankle) in the sagittal plane. Every muscle in the hindlimb represented a group of synergistic muscles. Overall, each hindlimb had 8 muscles with 6 mono-articular and 2 bi-articular muscles. A demonstration of this model's locomotion is presented in ***Figure 14***.

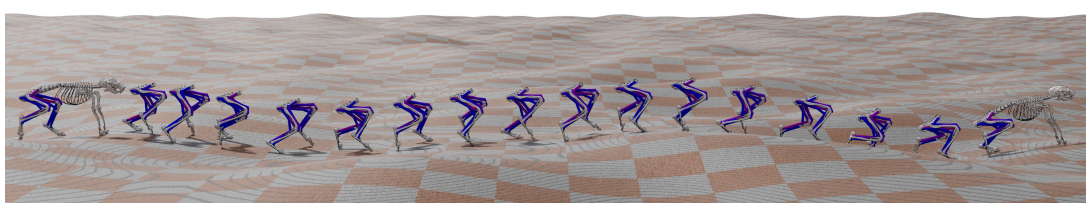


Figure 14. Mouse hindlimb locomotion over uneven terrain. A sequence gait snapshots over time of hindlimb mouse locomotion on uneven terrain controlled by a sensory driven spinal neural model similar to one described in ***Markin et al.*** (***2016***).

The spinal circuitry for locomotion was modeled as a two-layered neuronal network, as initially proposed by ***McCrea and Rybak*** (***2008***) and later implemented in a neuromechanical simulation by ***Markin et al.*** (***2016***). It included rhythm generators, each controlling one limb, that defined the locomotor frequency and flexor-extensor alternation. The rhythm generators projected their activity to the pattern formation circuits, which in turn generated muscle synergies and created muscle-specific activation patterns. These activation patterns were then used to drive the muscles of the musculoskeletal model. Each population was described as a non-spiking activity-based neuron model with persistent (slowly inactivating) sodium current (I_{NaP}) for excitatory rhythm and pattern centers.

Proprioceptive (spindle Ia and II, and Golgi tendon Ib) and exteroceptive feedback signals provided sensory feedback to the spinal circuit model. Spindle group II stretch feedback from tibialis anterior and iliopsoas and Golgi tendon Ib feedbacks from medial gastrocnemius and soleus were connected to the rhythm generators (affecting timing of phase-transitions). Basic Ia and Ib reflexes were organized at the motoneuron level. Proprioceptive feedback signals were modeled as described by ***Markin et al.*** (***2010***) and are currently available as part of farms_muscle package. The package farms_blender along with farms_muscle extends support to simulate complex Hill-type based musculoskeletal models in both Bullet and MuJoCo physics engines.

Conclusion

In this work, we have introduced a novel framework, FARMS, built for developing and performing neuromechanical simulations of animals and robots to study movement control. Its modular design allows independent use of its components depending on individual research requirements. FARMS is driven to maximize the utility of existing open-source projects, thereby harnessing their strengths and contributing back to existing infrastructure.

Although FARMS currently provides the necessary components for the demonstrations presented previously, we outline several avenues of future work in the pursuit of the continual enhancement of the framework. First, we aim to enrich the diversity of models by incorporating a wider variety of animals, robots, and arenas. This will further elevate the comprehensive nature of the framework and its applicability to a broader range of studies. We are also committed to expanding the biomechanical and musculoskeletal models in the framework, in order to foster a more accurate and detailed representation of biological systems. This will be complemented by the integration of spiking neural models based on the work in ***Pazzaglia et al.*** (***2025***), allowing for a more faithful portrayal of the complexity of neural information processing. Recognizing the importance of sensory inputs in both animal locomotion and robotic control, we plan to expand support for various sensors such as cameras and Inertial Measurement Unit (IMUs). We also intend to continue improving the SPH integration found in farms_sph based on recent works (***Angelidis et al., 2022***, ***2025***), and validating this feature to ensure reliable and accurate fluid simulation. In line with this, we aim to broaden the support of particle simulations for complex terrains, such as sand, to better model and understand locomotion in these environments. Additionally, we also plan to support soft skins and bodies, reflecting the compliance and versatility seen in many biological organisms. Concerning new file formats, we are considering transitioning from the current default SDF file format to the more versatile Universal Scene Description (USD) format by default for better integration with other modeling tools, while still providing support for SDF, URDF and MJCF within FARMS. To establish the fidelity of our simulations, we intend to undertake rigorous sim-to-real validations. Moreover, we will support reinforcement learning algorithms based on ***Raffin et al.*** (***2021***) to allow for the exploration of efficient and adaptive control strategies in both biological and robotic systems. We also aim to provide support for workflow managers such as Snakemake (***Mölder et al., 2021***) or SciLuigi (***Lampa et al., 2016***), which will facilitate the reproducibility and scalability of studies conducted using FARMS. Finally, with the goal of democratizing access to this interdisciplinary research, we are currently developing a user-friendly interface that further lowers the barrier to entry for researchers interested in leveraging FARMS for their work.

To conclude, our aim is for FARMS to serve as an open platform for advancing the study of animal and robotics systems' motion. It provides an array of tools, designed to facilitate the creation of models with realistic dynamics, the development of traditional and neural controllers, and comparative studies among animals and robotic systems. Our aspiration is to cultivate FARMS as an open-source, community-driven project, continually evolving to meet the dynamic demands of the scientific community. Thus, we hope that FARMS will help bridge the gap among various fields of research, including biomechanics, neuroscience, and robotics.

Acknowledgments

This research was supported in part by the Human Frontier Science Program (HFSP-RGP0027/2017); in part by the European Research Council (ERC) under the European Union's Horizon 2020 research (Synergia project, Salamandra); in part by the innovation programme (grant agreement No 951477); European Union's Horizon 2020 Research and Innovation Program under Agreement 720270917 (SGA1) and Agreement 785907 (SGA2); and in part by the National Institutes of Health (NIH) under Grant R01NS112304, Grant R01NS115900 and in part by 284289 Jekkal Fellowship, Drexel University.

We would like to thank members from Biorobotics Laboratory (BioRob) - EPFL, Neuroengineering Laboratory - EPFL, Danner Lab - Drexel University and Camilla Carta, for valuable feedback on the framework development.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). {TensorFlow}: A System for {Large-Scale} Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283.

Angelidis, E., Arreguit, J., Bender, J., Berggold, P., Liu, Z., Knoll, A., Crespi, A., and Ijspeert, A. J. (2025). A smoothed particle hydrodynamics framework for fluid simulation in robotics. *Robotics and Autonomous Systems*, 185:104885.

Angelidis, E., Bender, J., Arreguit, J., Gleim, L., Wang, W., Axenie, C., Knoll, A., and Ijspeert, A. (2022). Gazebo Fluids: SPH-based simulation of fluid interaction with articulated rigid body dynamics. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11238–11245.

Aoi, S. and Funato, T. (2016). Neuromusculoskeletal models based on the muscle synergy hypothesis for the investigation of adaptive motor control in locomotion via sensory-motor coordination. *Neuroscience Research*, 104:88–95.

Arreguit, J., Ramalingasetty, T. S., Gupta, A., and Ijspeert, A. (2021). Multi-Objective Optimization based 3D Walking of a Neuromuscular Driven Salamander Model in Simulation. In *Adaptive Motion of Animals and Machines*, page 29. Adaptive Motion of Animals and Machines Organizing Committee.

Auerbach, J., Aydin, D., Maesani, A., Kornatowski, P., Cieslewski, T., Heitz, G., Fernando, P., Loshchilov, I., Daler, L., and Floreano, D. (2014). RoboGen: Robot Generation through Artificial Evolution. In *ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, pages 136–137. MIT Press.

Bender, J. (2017). SPlisHSPlasH Library.

Blank, J. and Deb, K. (2020). Pymoo: Multi-Objective Optimization in Python. *IEEE Access*, 8:89497–89509.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym.

Caggiano, V., Wang, H., Durandau, G., Sartori, M., and Kumar, V. (2022). MyoSuite – A contact-rich simulation suite for musculoskeletal motor control.

Charles, J. P., Cappellari, O., Spence, A. J., Hutchinson, J. R., and Wells, D. J. (2016a). Musculoskeletal Geometry, Muscle Architecture and Functional Specialisations of the Mouse Hindlimb. *PLOS ONE*, 11(4):e0147669.

Charles, J. P., Cappellari, O., Spence, A. J., Wells, D. J., and Hutchinson, J. R. (2016b). Muscle moment arms and sensitivity analysis of a mouse hindlimb musculoskeletal model. *Journal of Anatomy*, 229(4):514–535.

Cofer, D., Cymbalyuk, G., Reid, J., Zhu, Y., Heitler, W. J., and Edwards, D. H. (2010). AnimatLab: A 3D graphics environment for neuromechanical simulations. *Journal of Neuroscience Methods*, 187(2):280–288.

Collins, J., Chand, S., Vanderkop, A., and Howard, D. (2021). A Review of Physics Simulators for Robotic Applications. *IEEE Access*, 9:51416–51431.

Community, B. O. (2022). Blender - a 3D modelling and rendering package. Blender Foundation.

Coumans, E. and Bai, Y. (2016). Pybullet, a python module for physics simulation for games, robotics and machine learning.

Crespi, A., Badertscher, A., Guignard, A., and Ijspeert, A. J. (2005). AmphiBot I: An amphibious snake-like robot. *Robotics and Autonomous Systems*, 50(4):163–175.

Crespi, A. and Ijspeert, A. J. (2006). AmphiBot II: An Amphibious Snake Robot that Crawls and Swims using a Central Pattern Generator. In *Proceedings of the 9th International Conference on Climbing and Walking Robots*, pages 19–27, Brussels, Belgium.

Crespi, A., Karakasiliotis, K., Guignard, A., and Ijspeert, A. J. (2013). Salamandra Robotica II: An Amphibious Robot to Study Salamander-Like Swimming and Walking Gaits. *IEEE Transactions on Robotics*, 29(2):308–320.

Danner, S. M., Shevtsova, N. A., Frigon, A., and Rybak, I. A. (2017). Computational modeling of spinal circuits controlling limb coordination and gaits in quadrupeds. *eLife*, 6:e31050.

Daun-Gruhn, S. (2011). A mathematical modeling study of inter-segmental coordination during stick insect walking. *Journal of Computational Neuroscience*, 30(2):255–278.

De Groote, F., Kinney, A. L., Rao, A. V., and Fregly, B. J. (2016). Evaluation of Direct Collocation Optimal Control Problem Formulations for Solving the Muscle Redundancy Problem. *Annals of Biomedical Engineering*, 44(10):2922–2936.

DeLaurier, A., Burton, N., Bennett, M., Baldock, R., Davidson, D., Mohun, T. J., and Logan, M. P. (2008). The Mouse Limb Anatomy Atlas: An interactive 3D tool for studying embryonic limb patterning. *BMC Developmental Biology*, 8(1):83.

Dickinson, M. H. (2000). How Animals Move: An Integrative View. *Science*, 288(5463):100–106.

Dzeladini, F., van den Kieboom, J., and Ijspeert, A. (2014). The contribution of a central pattern generator in a reflex-based neuromuscular model. *Frontiers in Human Neuroscience*, 8.

Edwards, D. H. (2010). Neuromechanical Simulation. *Frontiers in Behavioral Neuroscience*, 4.

Ekeberg, Ö. (1993). A combined neuronal and mechanical model of fish swimming. *Biological Cybernetics*, 69(5):363–374.

Ekeberg, Ö. and Pearson, K. (2005). Computer Simulation of Stepping in the Hind Legs of the Cat: An Examination of Mechanisms Regulating the Stance-to-Swing Transition. *Journal of Neurophysiology*, 94(6):4256–4268.

Ermentrout, B. (1994). Reduction of Conductance-Based Models with Slow Synapses to Neural Nets. *Neural Computation*, 6(4):679–695.

FitzHugh, R. (1961). Impulses and Physiological States in Theoretical Models of Nerve Membrane. *Biophysical Journal*, 1(6):445–466.

Geijtenbeek, T. (2013). *Animating Virtual Characters Using Physics-Based Simulation*. Utrecht University.

Geijtenbeek, T. (2019). SCONE: Open Source Software for Predictive Simulation of Biological Motion. *Journal of Open Source Software*, 4(38):1421.

Geijtenbeek, T. (2021). The Hyfydy simulation software.

Gewaltig, M.-O. and Diesmann, M. (2007). NEST (NEural Simulation Tool). *Scholarpedia*, 2(4):1430.

Geyer, H. and Herr, H. (2010). A Muscle-Reflex Model That Encodes Principles of Legged Mechanics Produces Human Walking Dynamics and Muscle Activities. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 18(3):263–273.

Goodman, D. and Brette, R. (2009). The Brian simulator. *Frontiers in Neuroscience*, 3.

Grillner, S., Deliagina, T., Manira, A. E., Hill, R. H., Lansner, A., and Orlovsky, G. N. (1995). Neural networks that co-ordinate locomotion and body orientation in lamprey. *Trends in neurosciences*, 18(6):270–279.

Guo, S., Lin, J., Wöhrl, T., and Liao, M. (2018). A Neuro-Musculo-Skeletal Model for Insects With Data-driven Optimization. *Scientific Reports*, 8(1):2129.

Hamlet, C. L., Hoffman, K. A., Tytell, E. D., and Fauci, L. J. (2018). The role of curvature feedback in the energetics and dynamics of lamprey swimming: A closed-loop model. *PLOS Computational Biology*, 14(8):e1006324.

Harischandra, N., Cabelguen, J.-M., and Ekeberg, Ö. (2010). A 3D Musculo-Mechanical Model of the Salamander for the Study of Different Gaits and Modes of Locomotion. *Frontiers in Neurorobotics*, 4.

Hines, M. L. and Carnevale, N. T. (1997). The NEURON simulation environment. *Neural Computation*, 9(6):1179–1209.

Hodgkin, A. L. and Huxley, A. F. (1952). Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo. *The Journal of Physiology*, 116(4):449–472.

Holmes, P., Full, R. J., Koditschek, D., and Guckenheimer, J. (2006). The Dynamics of Legged Locomotion: Models, Analyses, and Challenges. *SIAM Review*, 48(2):207–304.

Ijspeert, A. J. (2001). A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander. *Biological Cybernetics*, 84(5):331–348.

Ijspeert, A. J. (2014). Biorobotics: Using robots to emulate and investigate agile locomotion. *Science*, 346(6206):196–203.

Ijspeert, A. J., Crespi, A., Ryczko, D., and Cabelguen, J.-M. (2007). From Swimming to Walking with a Salamander Robot Driven by a Spinal Cord Model. *Science*, 315(5817):1416–1420.

Ijspeert, A. J. and Daley, M. A. (2023). Integration of feedforward and feedback control in the neuromechanics of vertebrate locomotion: A review of experimental, simulation and robotic studies. *Journal of Experimental Biology*, 226(15):jeb245784.

Ijspeert, A. J. and Kodjabachian, J. (1999). Evolution and Development of a Central Pattern Generator for the Swimming of a Lamprey. *Artificial Life*, 5(3):247–269.

Karakasiliotis, K., Thandiackal, R., Melo, K., Horvat, T., Mahabadi, N. K., Tsitkov, S., Cabelguen, J. M., and Ijspeert, A. J. (2016). From cineradiography to biorobots: An approach for designing robots to emulate and study animal locomotion. *Journal of The Royal Society Interface*, 13(119):20151089.

Koenig, N. and Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3.

Kuramoto, Y. (1975). Self-entrainment of a population of coupled non-linear oscillators. In Araki, H., editor, *International Symposium on Mathematical Problems in Theoretical Physics*, Lecture Notes in Physics, pages 420–422, Berlin, Heidelberg. Springer.

Lampa, S., Alvarsson, J., and Spjuth, O. (2016). Towards agile large-scale predictive modelling in drug discovery with flow-based programming design principles. *Journal of Cheminformatics*, 8(1):67.

Lee, J., X. Grey, M., Ha, S., Kunz, T., Jain, S., Ye, Y., S. Srinivasa, S., Stilman, M., and Karen Liu, C. (2018). DART: Dynamic Animation and Robotics Toolkit. *The Journal of Open Source Software*, 3(22):500.

Liang, J., Makoviychuk, V., Handa, A., Chentanez, N., Macklin, M., and Fox, D. (2018). GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning. In *Proceedings of The 2nd Conference on Robot Learning*, pages 270–282. PMLR.

Lobato-Rios, V., Ramalingasetty, S. T., Özdil, P. G., Arreguit, J., Ijspeert, A. J., and Ramdya, P. (2022). NeuroMechFly, a neuromechanical model of adult Drosophila melanogaster. *Nature Methods*, 19(5):620–627.

Manal, K. and Buchanan, T. S. (2004). Subject-Specific Estimates of Tendon Slack Length: A Numerical Method. *Journal of Applied Biomechanics*, 20(2):195–203.

Markin, S. N., Klishko, A. N., Shevtsova, N. A., Lemay, M. A., Prilutsky, B. I., and Rybak, I. A. (2010). Afferent control of locomotor CPG: Insights from a simple neuromechanical model: Afferent control of locomotor CPG. *Annals of the New York Academy of Sciences*, 1198(1):21–34.

Markin, S. N., Klishko, A. N., Shevtsova, N. A., Lemay, M. A., Prilutsky, B. I., and Rybak, I. A. (2016). A Neuromechanical Model of Spinal Control of Locomotion. In Prilutsky, B. I. and Edwards, D. H., editors, *Neuromechanical Modeling of Posture and Locomotion*, Springer Series in Computational Neuroscience, pages 21–65. Springer, New York, NY.

Mathewson, M. A., Chapman, M. A., Hentzen, E. R., Fridén, J., and Lieber, R. L. (2012). Anatomical, architectural, and biochemical diversity of the murine forelimb muscles. *Journal of Anatomy*, 221(5):443–451.

Matsuoka, K. (1985). Sustained oscillations generated by mutually inhibiting neurons with adaptation. *Biological Cybernetics*, 52(6):367–376.

McCrea, D. A. and Rybak, I. A. (2008). Organization of mammalian locomotor rhythm and pattern generation. *Brain Research Reviews*, 57(1):134–146.

Melo, K., Horvat, T., and Ijspeert, A. (2017). K-Rock, a Bio-robot Outside the Lab, Back In Nature. *The 8th International Symposium on Adaptive Motion of Animals and Machines (AMAM2017)*.

Melo, K., Horvat, T., and Ijspeert, A. J. (2023). Animal robots in the African wilderness: Lessons learned and outlook for field robotics. *Science Robotics*, 8(85):eadd8662.

Michel, O. (2004). Cyberbotics Ltd. Webots™: Professional Mobile Robot Simulation. *International Journal of Advanced Robotic Systems*, 1(1):5.

Millard, M., Uchida, T., Seth, A., and Delp, S. L. (2013). Flexing Computational Muscle: Modeling and Simulation of Musculotendon Dynamics. *Journal of Biomechanical Engineering*, 135(2):0210051–02100511.

Modenese, L., Ceseracciu, E., Reggiani, M., and Lloyd, D. G. (2016). Estimation of musculotendon parameters for scaled and subject specific musculoskeletal models using an optimization technique. *Journal of Biomechanics*, 49(2):141–148.

Mölder, F., Jablonski, K. P., Letcher, B., Hall, M. B., Tomkins-Tinch, C. H., Sochat, V., Forster, J., Lee, S., Twardziok, S. O., Kanitz, A., Wilm, A., Holtgrewe, M., Rahmann, S., Nahnsen, S., and Köster, J. (2021). Sustainable data analysis with Snakemake.

Nyakatura, J. A., Melo, K., Horvat, T., Karakasiliotis, K., Allen, V. R., Andikfar, A., Andrada, E., Arnold, P., Lauströer, J., Hutchinson, J. R., Fischer, M. S., and Ijspeert, A. J. (2019). Reverse-engineering the locomotion of a stem amniote. *Nature*, 565(7739):351–355.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Pazzaglia, A., Bicanski, A., Ferrario, A., Arreguit, J., Ryczko, D., and Ijspeert, A. (2025). Balancing central control and sensory feedback produces adaptable and robust locomotor patterns in a spiking, neuromechanical model of the salamander spinal cord. *PLOS Computational Biology*, 21(1):e1012101.

Puri, K., Ramachandran, P., Pandey, P., Kaushik, C., and Godbole, P. (2013). PySPH: A Python framework for SPH. In *8th International SPHERIC Workshop*, Trondheim, Norway.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8.

Ramachandran, P., Bhosale, A., Puri, K., Negi, P., Muta, A., Dinesh, A., Menon, D., Govind, R., Sanka, S., Sebastian, A. S., Sen, A., Kaushik, R., Kumar, A., Kurapati, V., Patil, M., Tavker, D., Pandey, P., Kaushik, C., Dutt, A., and Agarwal, A. (2021). PySPH: A Python-based framework for smoothed particle hydrodynamics. *ACM Transactions on Mathematical Software*, 47(4):1–38.

Righetti, L., Buchli, J., and Ijspeert, A. J. (2006). Dynamic Hebbian learning in adaptive frequency oscillators. *Physica D: Nonlinear Phenomena*, 216(2):269–281.

Seth, A., Hicks, J. L., Uchida, T. K., Habib, A., Dembia, C. L., Dunne, J. J., Ong, C. F., DeMers, M. S., Rajagopal, A., Millard, M., Hamner, S. R., Arnold, E. M., Yong, J. R., Lakshmikanth, S. K., Sherman, M. A., Ku, J. P., and Delp, S. L. (2018). OpenSim: Simulating musculoskeletal dynamics and neuromuscular control to study human and animal movement. *PLOS Computational Biology*, 14(7):e1006223.

Seth, A., Sherman, M., Reinbolt, J. A., and Delp, S. L. (2011). OpenSim: A musculoskeletal modeling and simulation framework for in silico investigations and exchange. *Procedia IUTAM*, 2:212–232.

Sherman, M. A., Seth, A., and Delp, S. L. (2011). Simbody: Multibody dynamics for biomedical research. *Procedia IUTAM*, 2:241–261.

Sims, K. (1994a). Evolving 3D Morphology and Behavior by Competition. *Artificial Life*, 1(4):353–372.

Sims, K. (1994b). Evolving virtual creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 15–22, New York, NY, USA. Association for Computing Machinery.

Smith, R. (2006). OPEN DYNAMICS ENGINE. Technical report.

Stimberg, M., Brette, R., and Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. *eLife*, 8:e47314.

Tata Ramalingasetty, S., Danner, S. M., Arreguit, J., Markin, S. N., Rodarie, D., Kathe, C., Courtine, G., Rybak, I. A., and Ijspeert, A. J. (2021). A Whole-Body Musculoskeletal Model of the Mouse. *IEEE Access*, 9:163861–163881.

Tata Ramalingasetty, S., Markin, S. N., Lockhart, A. B., Arreguit, J., Shevtsova, N. A., Ijspeert, A. J., Rybak, I. A., and Danner, S. M. (2023). An integrated neuromechanical model of the mouse to study neural control of locomotion.

Thandiackal, R., Melo, K., Paez, L., Herault, J., Kano, T., Akiyama, K., Boyer, F., Ryczko, D., Ishiguro, A., and Ijspeert, A. J. (2021). Emergence of robust self-organized undulatory swimming based on local hydrodynamic force sensing. *Science Robotics*, 6(57).

The HDF Group (1997/2023). Hierarchical data format, version 5.

Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033.

Yakovenko, S., Gritsenko, V., and Prochazka, A. (2004). Contribution of stretch reflexes to locomotor control: A modeling study. *Biological Cybernetics*, 90(2):146–155.

Appendix

Robots

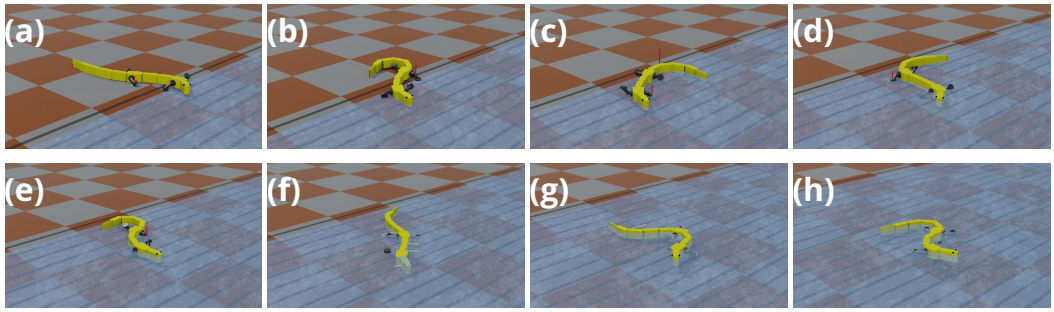
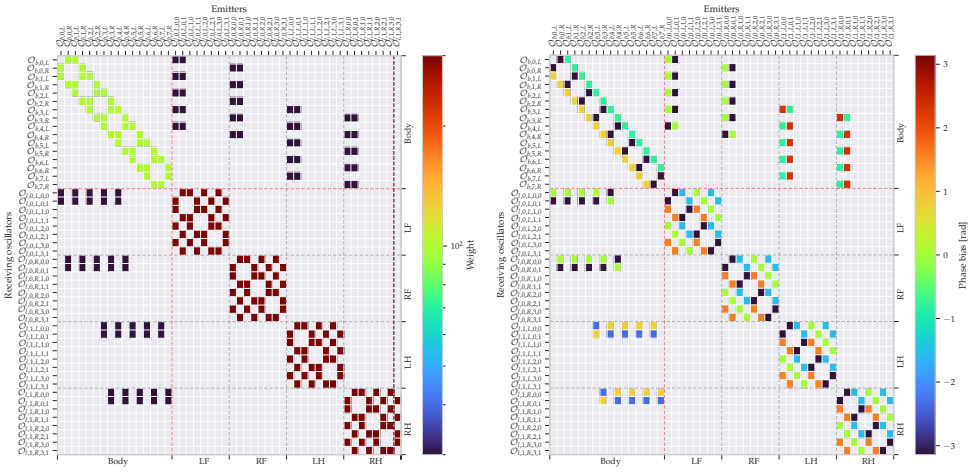


Figure 15. Salamandra Robotica II performing a walk-to-swim transition Sequence of rendered snapshots separated by 1.5 [s]. The robot is walking from right to left in 15a and comes into contact with the water. Throughout 15b to 15f, the robot keeps walking even though it is in contact with the water as it is still in contact with the ramp and the contact force threshold between the environments and the body has not been surpassed. Between 15f and 15g, the body is deep enough into the water to lift the body and the robot transitions from walking to swimming. The limbs saturate and move back and the trunk switches from a standing wave to a travelling wave, allowing it to propel itself forward in the water.

Salamander



(a) Connectivity weights

(b) Connectivity phase bias

Figure 16. Details for the salamander connectivity for an oscillator network. The coupling weights between pairs of oscillators are given in ***Figure 16a*** and the phase bias are provided in ***Figure 16a***.

Snake

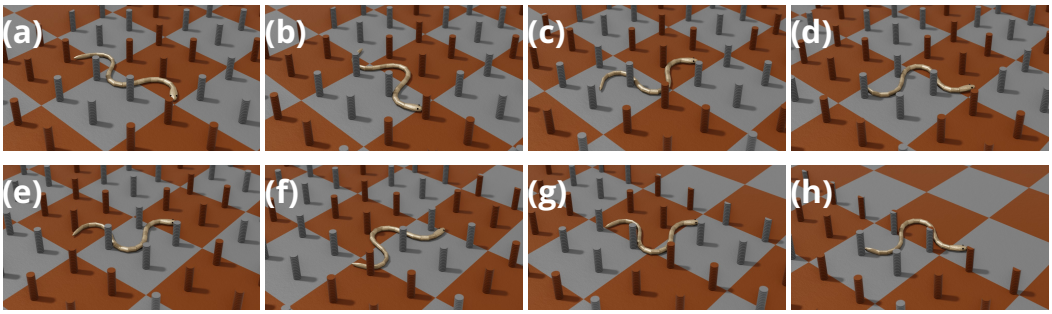


Figure 17. Snake obstacle-aided locomotion in arena 2 Sequence of rendered snapshots separated by 2.5 [s]. The pegs have a diameter of 4 [cm] and are separated by 27 [cm] from centre-to-centre.

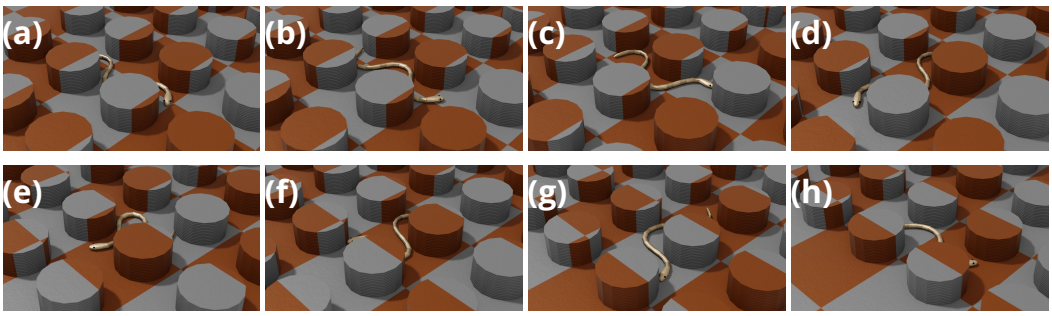


Figure 18. Snake obstacle-aided locomotion in arena 3 Sequence of rendered snapshots separated by 2.5 [s]. The pegs have a diameter of 30 [cm] and are separated by 45 [cm] from centre-to-centre.

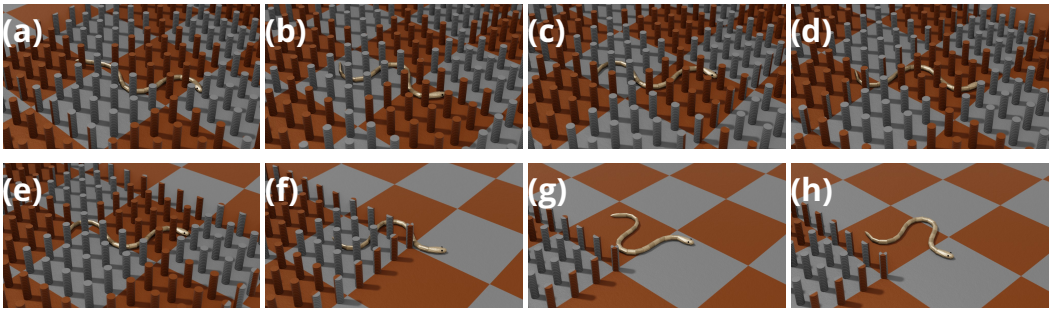


Figure 19. Snake obstacle-aided locomotion in arena 4 Sequence of rendered snapshots separated by 2.5 [s]. The pegs have a diameter of 4 [cm] and are separated by 14 [cm] from centre-to-centre.