Received 12 November 2025, accepted 4 December 2025, date of publication 15 December 2025,
date of current version 23 December 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3644237

RESEARCH ARTICLE

Introducing MOSAIC: A Modular, Open-Source Suite for Assistive Intelligent Control

HANNAH BRAUN[1], (Student Member, IEEE), MAREK SIEROTOWICZ[1,2], (Member, IEEE), FABIO EGLE[1], (Graduate Student Member, IEEE), MARC-ANTON SCHEIDL[1], (Graduate Student Member, IEEE), SILVANA MIRANDA MONTENEGRO[1], (Student Member, IEEE), SABINE THUERAUF[1], (Member, IEEE), AND CLAUDIO CASTELLINI[1,2], (Member, IEEE)

[1]Department Artificial Intelligence in Biomedical Engineering, Friedrich-Alexander-Universität Erlangen-Nürnberg, 91052 Erlangen, Germany
[2]Institute of Robotics and Mechatronics, German Aerospace Center (DLR), 82234 Weßling, Germany

Corresponding author: Hannah Braun (hannah.b.braun@fau.de)

This work was supported in part by the Istituto Nazionale per l'Assicurazione contro gli Infortuni sul Lavoro (INAIL) under Grant PR23-PAS-P1; in part by the Project DexterHand; in part by European Commission's Horizon Europe Framework Program with the Project IntelliMan (AI-Powered Manipulation System for Advanced Robotic Service, Manufacturing and Prosthetics) under Grant 101070136; in part by German Federal Ministry of Education and Research (BMBF) under the Robotics Institute Germany (RIG) and through Project AutoAssist under Grant 13GW0716B; and in part by the German Research Foundation (DFG) through the individual research grant HIT-Reha—Human Impedance Control for Tailored Rehabilitation, DFG Sachbeihilfe CA1389/3-1, under Project 505327336.

ABSTRACT Modern assistive technologies, including prosthetic limbs, orthotic devices, and rehabilitation robots, are increasingly being driven by biosignal-based control systems. However, developing and validating such systems remains challenging owing to the complex signal-processing pipelines, heterogeneous hardware requirements, and limited support for real-time prototyping. We present *MOSAIC*, an open-source, modular software suite written in C# for real-time biosignal processing and intelligent device control. Configured via human-readable files, *MOSAIC* enables rapid prototyping through functional units called blocks, which support asynchronous data flow and precise execution timing. The system is hardware-independent and supports a wide range of sensors and output interfaces, including electromyography, inertial sensors, ultrasound, functional electrical stimulation, Internet Protocol over User Datagram Protocol, serial communication, and Bluetooth streaming. A live graphical interface enables interactive experimentation and immediate inspection of system behavior. We demonstrate *MOSAIC* across real-world applications, including prosthetic limb control, gesture-guided medical systems, and muscle-controlled orthotic devices. Its extensible design supports reproducible experimentation in assistive technologies.

INDEX TERMS Assistive robotics, biosignal processing, human–machine interaction, machine learning, modular software architecture, open-source software, prosthesis control, real-time systems.

I. INTRODUCTION

The global demand for assistive technologies, including prosthetic limbs [1], [2], [3] and assistive robots, is increasing due to demographic shifts, rising rates of chronic conditions, and traumatic injuries [4], [5]. For instance, in 2017 alone, an estimated 57.7 million individuals were living with traumatic limb amputation worldwide [2], underscoring the pressing need for scalable and accessible prosthetic rehabilitation solutions. In the context of assistive robotics, adoption remains limited despite technological maturity, due to technical, institutional, and financial barriers to healthcare integration [6]. End-users of these technologies often present heterogeneous physiological and cognitive profiles, necessitating personalized solutions that can adapt to individual needs and limitations. Designing and evaluating effective assistive robots and prosthetic systems requires an iterative development process that emphasizes rapid prototyping, continuous user feedback, and adaptation to individual needs and preferences [7], [8], [9]. To support such

The associate editor coordinating the review of this manuscript and approving it for publication was Yizhang Jiang.

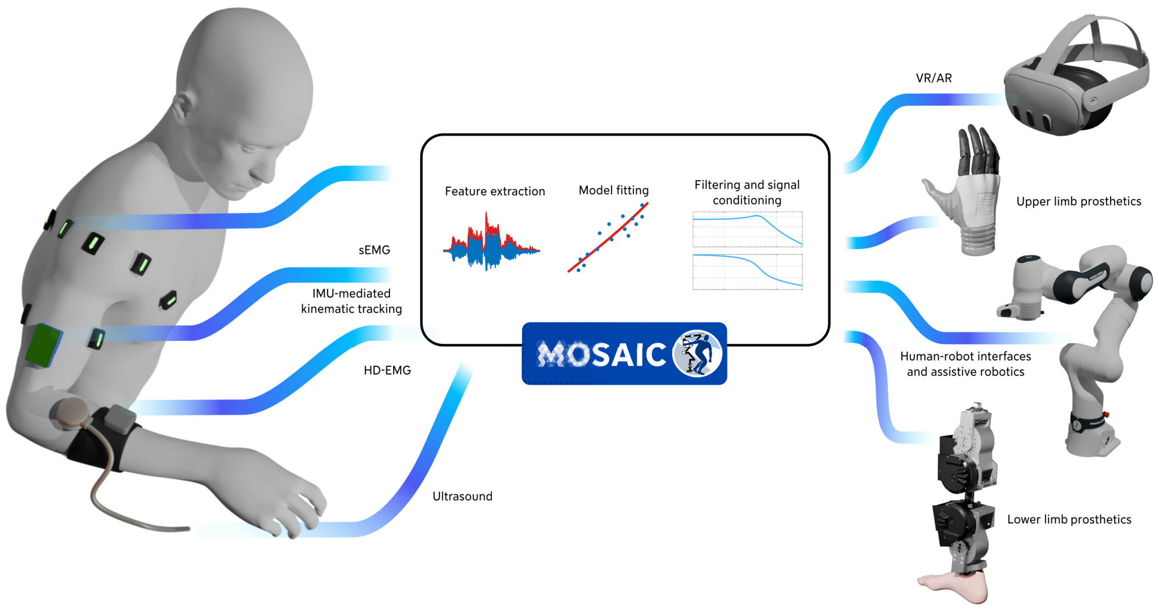
VR/AR
Feature extraction
Model fitting
Filtering and signal conditioning
Upper limb prosthetics
sEMG
IMU-mediated kinematic tracking
HD-EMG
MOSAIC
Human-robot interfaces and assistive robotics
Ultrasound
Lower limb prosthetics

Figure 1. Conceptual block diagram of *MOSAIC*. As it takes in data from various biosensors, it translates them into control signals for various robotic devices.

workflows, software platforms must be flexible, modular, and easily reconfigurable, enabling researchers and engineers to efficiently explore different designs, sensor combinations, and control strategies in collaboration with target user groups [8].

In this context, sensor modalities such as electromyography (EMG) [10], inertial measurement units (IMUs) [11], force myography (FMG) [12], ultrasound [13] and electroencephalography (EEG) [14] are crucial for enabling natural, intention-driven control [15], [16]. The resulting physiological signals are widely utilized in assistive and rehabilitation technologies to infer user intent, adjust assistance levels, and facilitate personalized interactions, typically through machine learning algorithms that map biosignal patterns to control commands [17]. Surface EMG (sEMG) and EEG are the most commonly employed input modalities in biosignal-driven human–machine interfaces (HMIs), owing to their ability to capture muscular and cognitive activity, respectively [16]. For example, in upper-limb prosthetics, multichannel sEMG combined with machine learning enables real-time decoding of muscle activity to control multi-degree-of-freedom devices [18].

Despite advances in sensor technology and control algorithms, the development of effective biosignal-driven systems remains a significant challenge [19]. Applications often rely on carefully tuned combinations of sensing modalities, feature extraction pipelines, machine learning models, and feedback mechanisms, all of which must be adapted to individual users, target devices, and experimental goals. Variability in biosignal patterns over time, sensor placement, and environmental conditions calls for adaptive and modular control architectures [19]. Furthermore, each device usually provides data or requires control signals at different independent sampling rates, which poses problems in terms of synchronisation, data flow control, and CPU requirements. This complexity makes the development process time-consuming and error-prone. Existing tools frequently lack this flexibility, relying instead on rigid or monolithic software stacks, which hinder reproducibility and slow development cycles. A few providers deliver semi-flexible software solutions for clinical and home use in specific applications, such as prosthetic control [20].

These challenges highlight the need for a flexible, modular, and extendable framework that supports rapid development, real-time feedback, and hardware interoperability. To address these limitations, we present *MOSAIC* - a Modular, Open-Source Suite for Assistive Intelligent Control - designed with a C#.NET Framework 4.8 [21] specifically for biosignal-driven real-time human–machine interactions (see Fig. 1). The platform supports rapid assembly and visualization of experimental pipelines composed of interoperable building blocks. Each block encapsulates a self-contained function such as signal acquisition, filtering, feature computation, learning, or control. Pipelines are defined using a human-readable Yet Another Markup Language (YAML) file and executed in real-time through an event-driven scheduler. *MOSAIC* provides several core features that facilitate experimentation and deployment: a global graphical visualization "table" graph for live data-flow monitoring; integrated control panels for each block, to assess data flow problems and bottlenecks; batch and incremental machine learning modules; mathematical operations; and multi-protocol support for interfacing with physical or virtual hardware via, for example, User Datagram Protocol (UDP), serial-over-Bluetooth and Bluetooth-Low-Energy transmission. Moreover, *MOSAIC* supports a graph-like dynamic structure, allowing for the real-time creation of loops and closed-loop feedback control systems.

Our main contributions are:

- **A GUI-instrumented, soft real-time runtime** for biosignal acquisition, processing, inference, actuation, with event-driven multi-rate scheduling and per-block timing diagnostics via the *BlockTable*.
- **YAML-first, versionable pipelines** that enable rapid prototyping, schema-validated configuration, and record–replay for reproducibility.
- **Integrated batch and online learning** through *BatchPredictor* and *IncrementalPredictor*, supporting flexible human-in-the-loop adaptation during training and stable inference at runtime.
- **Hardware-agnostic I/O and communication** with modular device managers for EMG/IMU/ultrasound and connectors for UDP/Serial, enabling plug-and-play sensing and actuation.
- **Budget-aware Python integration** via the *PythonNetManager* block: Poetry-managed environments and direct reuse of NumPy/PyTorch/Scikit-learn code, with timing overruns surfaced as on-time/lagging/stumbling states in the *BlockTable*.
- **ROS connectivity** through the *ROS* block (WebSocket/rosbridge) for real-time topic publish/subscribe, allowing closed-loop experiments with robot stacks and simulators, and complementing UDP/Serial links.
- **Quantitative runtime evaluation** on representative pipelines, Myo (8×200 Hz) and Delsys Trigno (32×1926 Hz)—showing sub-millisecond inter-sample jitter and median CPU loads of $\sim$12% and $\sim$29%, plus per-block on-time/lagging/stumbling distributions; a controlled record–replay comparison against Simulink reports similar CPU and markedly lower RAM under identical timing.
- **Validated applications across published studies**, including upper-limb prosthesis control, C-arm gesture control, lower-limb orthosis assistance, 3D tele-impedance stiffness estimation, FES-mediated movement, and virtual-hand control in spinal cord injury.
- **Open artifacts**—source code, example YAML pipelines, and documentation.

II. BACKGROUND

Modern rehabilitation robotics and biosignal-driven control systems increasingly rely on multi-stage data pipelines combining signal acquisition, filtering, machine learning, and device control. Traditionally, developing these systems requires both domain expertise and extensive coding, creating a barrier for rapid experimentation and reproducibility.

A. BLOCK-BASED SOFTWARE ARCHITECTURES

To mitigate complexity and promote modularity of processing and control, several frameworks allow users to build data processing pipelines by graphically connecting functional units ("blocks"), each one performing a specific, single, well-isolated task. This modular structure supports rapid prototyping, intuitive design, and the reuse of components. Several open-source and commercial block-based systems are available and are utilized in the rehabilitation robotics community.

Simulink [22] is a well-known Graphical User Interface (GUI)-based design environment used for developing and verifying control systems. It provides a graphical environment for modeling and simulating dynamic systems and is widely used in control system engineering. Users drag and drop functional blocks (e.g., integrators, filters, and gain modules) to construct complex systems. Despite its power, Simulink is proprietary and tightly integrated with the MATLAB ecosystem, making it less accessible to small research groups and education-focused environments.

The Laboratory Virtual Instrument Engineering Workbench (LabVIEW) [23], developed by National Instruments, is a widely used graphical programming environment tailored for real-time data acquisition, instrument control, and hardware interfacing. It employs a dataflow-based programming language called G, in which functional blocks are connected through wires to represent the flow of data. LabVIEW is particularly prevalent in biomedical engineering, mechatronics, and embedded systems because of its native compatibility with a wide range of National Instruments (NI) hardware, including data acquisition (DAQ) devices, field-programmable gate arrays (FPGAs), and real-time controllers. The environment excels in sensor integration, feedback loops, and closed-loop control systems. However, despite its strengths, LabVIEW remains a closed-source and commercial solution, which limits its accessibility to low-resource research groups, open science initiatives, and educational institutions.

The Robot Operating System (ROS) [24], [25] is a widely used open-source middleware framework for developing robotic systems. Although not block-based in the traditional sense of a drag-and-drop GUI, ROS follows a graph-based, modular architecture. It decomposes robotic applications into coupled components, called *nodes*, that communicate via publish–subscribe messaging over standardized topics. ROS supports real-time sensor data handling, hardware abstraction, parameter configuration, and the integration of diverse control algorithms. Its modularity and support for distributed execution have made it the de facto standard for academic and industrial robotics research. Tools such as `rqt_graph`, `rosbag`, and the ROS launch system allow the visualization, recording, and configuration of processing pipelines, aligning closely with the goals of visual modular programming frameworks. Recent efforts, such as ROS-Neuro [26], [27], [28], extend the ROS ecosystem to specifically include biosignal processing and neurorobotics. ROS-Neuro provides standardized interfaces for EEG and EMG devices, decoding algorithms, and control modules, allowing researchers to prototype neurally controlled robots with reproducible pipelines. However, ROS's steep learning curve, reliance on C++/Python development, and lack of dedicated real-time GUI environments can limit accessibility for clinical and human-subject-focused experimentation.

B. BIOSIGNAL PROCESSING FRAMEWORKS

Various open-source frameworks and toolkits for processing biosignal data have been released, including essential pipeline components, such as data handling, filtering, feature extraction and reduction, and classification. Given their popularity in the community, such frameworks are mainly oriented towards EMG.

Ortiz-Catalán et al. introduced BioPatRec (Biopotential Pattern Recognition) [29], an open-source research platform based on MATLAB, designed to develop and benchmark myocontrol pattern recognition algorithms for prosthetic control. This platform aims to address the lack of standardization in signal processing pipelines across different studies. BioPatRec features a modular structure that encompasses the entire signal processing chain from acquisition, filtering, and segmentation to feature extraction, classification, and real-time control. The platform supports the integration of various classifiers, including Linear Discriminant Analysis (LDA), Multi-Layer Perceptron (MLP), and Regulatory Feedback Networks (RFN). Additionally, it includes a virtual reality test environment. Despite these capabilities, BioPatRec has not been widely adopted within the community, possibly due to its reliance on MATLAB.

A more recent addition to myoelectric control is LibEMG [30], which is an open-source Python library designed for developing myoelectric control systems. The library provides both offline and real-time modules, a collection of validated features and feature sets, and a hardware-agnostic UDP-based streaming interface. This enables integration with a range of commercial and custom EMG devices. LibEMG simplifies many domain-specific complexities, making it easier to build prototypes and broader applications for EMG-based interactions, not only in prosthesis control but also in the context of human-computer interaction. However, LibEMG focuses exclusively on EMG, with pipelines and interfaces built around sEMG-based pattern recognition and classification.

The open-source Python toolbox BioSPPy [31] allows the offline processing of various biosignals, including EMG. Users can create pipelines for data loading, processing, and feature extraction, with a range of options and interactive visualizations. However, it lacks real-time data collection and streaming from devices, as well as the necessary control steps for rehabilitation robotics and assistive technologies.

Complementary closed-loop approaches—ranging from wearable vibrotactile stimulation on 3D-printed hands [32] to real-time EMG feedback that enhances grasp–force modulation even in high-level amputation [33]—highlight that performance and usability depend not only on decoding but also on how feedback is delivered to the user. These studies point to practical, low-cost feedback channels that can be driven directly from controller outputs, yet they also expose the need for toolchains that make it straightforward to prototype the entire loop.

C. ROBOTICS MIDDLEWARE AND SIMULATION

In addition to biosignal frameworks, mature robotics-control frameworks exist; notable examples are Gazebo, Orocos, and ROS 2 with `ros_control`.

Gazebo [34], [35] is an open-source framework for physics-based 3D simulation. It combines rigid-body dynamics with realistic sensors to test perception and control in repeatable conditions. Gazebo supports multi-robot scenarios, offers plugin-based extensibility, and enables cloud or clustered deployments for large experiments. It is commonly used to prototype algorithms before hardware trials and to evaluate performance under controlled latency, bandwidth, and environmental variability.

Orocos [36] is a C++ framework designed for building real-time controllers using a component-based design approach. It facilitates task scheduling in real-time, enabling deterministic timing. The framework includes kinematics and dynamics libraries along with a strict scheduling mechanism. It is well-suited for low-level, time-critical control loops and industrial deployments.

ROS 2, along with the `ros_control` package [24], [37], provides a middleware framework for distributed robotics. It emphasizes message-passing, executors, and a controller manager with standardized interfaces to hardware. While it is not block-based in a GUI sense, ROS 2 encourages the use of modular graph topologies and integrates with both Gazebo and Orocos for simulation and deployment purposes.

Accordingly, there remains a need for a modern, open-source, hardware-agnostic framework that supports live biosignal streaming, soft real-time processing with observable timing, multimodal I/O (including feedback), and human-readable configuration to enable rapid, reproducible experimentation in clinical and academic settings.

III. INTRODUCING MOSAIC

MOSAIC, a Modular, Open-source Suite for Assistive Intelligent Control, is designed to overcome, at least partially, the drawbacks mentioned above. It is an intuitive graphical, modular tool primarily designed for experimental user studies in the control of assistive devices. It follows a block-based paradigm, meaning that each function required in the data processing pipeline is implemented as an independent module (block), individually connected to one or more blocks, based on the specific needs of the study.

MOSAIC allows users to configure, construct, visualize, and execute signal processing, machine learning, and device control pipelines in real-time. Users can create new studies in *MOSAIC* by defining blocks and their interconnections using human-readable YAML configuration files. Additionally, blocks can dynamically change their connections during execution, which is helpful in situations where precisely

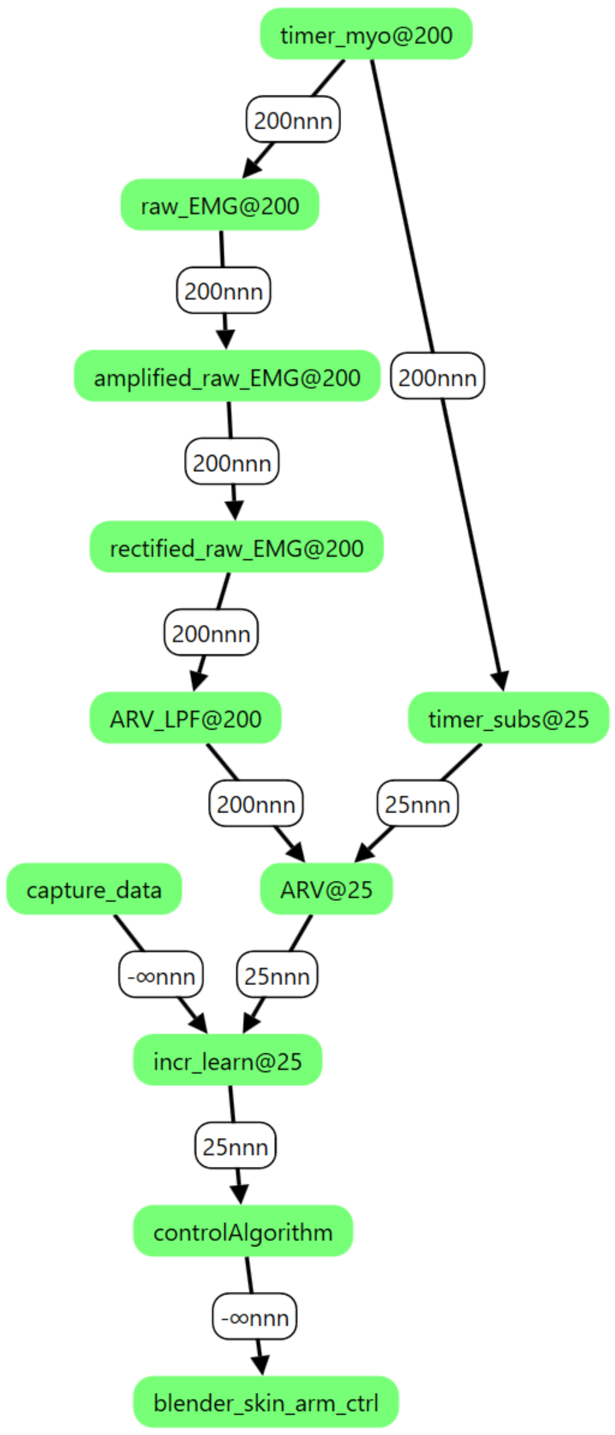

Figure 2. Live *BlockTable* visualization in *MOSAIC*. Each node represents a functional block, color-coded according to its runtime status. This example shows a *BlockTable* used for acquiring data from a Myo armband, applying filtering, feature extraction, subsampling, Ridge Regression, and feeding predictions into a control algorithm that transmits commands via UDP to control a virtual hand.

timed data storage is required, such as when triggered by an external event or user input.

To ensure correct data timing and flow, users can inspect a visual representation of the blocks and their interconnections, defined as the *BlockTable* (see Fig. 2). This graph-like view indicates instabilities and data leakage through color coding. Each block also features a *Control Panel*, which provides a window for users and/or study participants to monitor the data flowing through the block, check its timing and quality, and read or adjust parameters related to the block's function. For example, they can set timing parameters as required, start data sampling, and adjust gains. This combination of visualization, monitoring, and configurability is designed to make MOSAIC both a practical research tool and an accessible environment for clinicians and educators to explore biosignal-based control.

A. ARCHITECTURAL PRINCIPLES AND CLASS ABSTRACTION

The core of *MOSAIC* is implemented entirely in the.NET Framework 4.8 in C#, a language well-suited for building graphical environments, easily accessing physical devices, and supporting multi-threaded, object-oriented, and event-driven programming with runtime execution. In this framework, blocks are instances of concrete classes that inherit from the abstract class *Block*. This class includes an event handler for sending output data, a callback mechanism for receiving data, and a simple method to evaluate its own data processing rate based on the frequency of firing the output event. Each block is defined at startup using human-readable YAML configuration files, which must specify at least the following parameters of the abstract class:

- A *name* uniquely identifying this block.
- A *type*, that is this block's concrete class, enforcing a specific functionality, for example, a filter, a device driver, a machine-learning module.
- The *block's desired rate*, that is, the target running frequency.
- A *set of blocks* this block may receive input data from.
- A *set of parameters* specific to this block's function, for example, filter coefficients, machine-learning hyperparameters.
- A *path* on which to log the block's output, allowing for post-experiment analysis.

```
Name:
{
Type: Block Type Name, DesiredRate:

Desired Rate in Hz,

Inputs: [ List of Input Blocks],

Params: {},

Path: ``C:/path/''
}
```

The key aspect of connectivity between blocks is achieved by subscribing or unsubscribing to an input block's event handler at runtime. This subscription can be established at the beginning via the configuration file or triggered later by specific events, such as an external trigger or user interaction with a graphical element. Generally, each block subscribes to one or more of its input blocks, processes the data received from those blocks, and broadcasts the results of its own computations by firing its output event. The block network is visually represented in the *BlockTable*, which provides a graphical depiction of the system's connectivity and activity. Typically, each block performs only the necessary computations when new input is available, thereby minimizing computational costs. This approach allows the *MOSAIC* architecture to take advantage of the native multi-threading and event-handling capabilities of C#, ensuring responsive and concurrent behavior.

It also minimizes the shared state and favors loosely coupled communication, making it easy to add new functionality or hardware drivers by implementing new block types. Overall, implemented abstraction ensures flexibility, extensibility, and clarity—critical properties for educational use, experimental reproducibility, and collaborative development.

B. THE BLOCKTABLE: LIVE VISUALIZATION AND INTERACTION LAYER

The *BlockTable* (see Fig. 2) serves as the central graphical feature of the *MOSAIC* environment. It offers a live, interactive visualization of the block graph defined in the YAML file. Built using GraphX [38] and QuickGraph [39], *BlockTable* visualises each block as a vertex and each data connection as an arrow in a graph. A global timer that runs at 20 Hz triggers regular updates of the *BlockTable*, ensuring a balance between GUI responsiveness and computational efficiency.

During operation, the *BlockTable* provides real-time feedback on the state of each block. Blocks are color-coded such that:

- *Grey*: Denotes that the block is "idle" and has not produced recent outputs. The processing pipeline is currently not in operation.
- *Green*: The block is operating above 95% of its desired rate.
- *Red*: The block is "lagging"; its current rate is less than 95% of the desired rate.
- *Magenta*: The block is "stumbling" — its own computation time exceeds the interval specified as the inverse of the desired rate. Notice that this condition is independent of lagging.

Each edge connecting the blocks is labelled with the actual rate of the source block and graphically shows lagging, stumbling, and subscription; along with the color coding, this makes it easy to determine if the processing pipeline is keeping up with real-time demands or if the computational power is struggling to maintain pace.

Interaction with the *BlockTable* is intuitive:

- Left-clicking a block opens its *Control Panel* (CP) for real-time monitoring and control.
- Right-clicking performs block-specific actions, such as starting or stopping a timer.
- Blocks can be rearranged via drag-and-drop to improve layout clarity.

C. CONTROL PANELS AND VISUALIZATION TOOLS

Depending on the functionality and necessity of the block, it may include a CP, which is a Windows Form that provides access to the block's parameters, runtime data, and control interfaces. The CP serves as a dedicated GUI extension for each block, allowing developers and experimenters to inspect signals in real-time, adjust parameters on the fly, and interact with the system during runtime. CPs are designed to be lightweight yet robust, updating at a fixed rate through a shared dispatcher timer with the *BlockTable*, as described in detail in Subsection III-B, to ensure low CPU usage. The GUI components within a CP vary depending on the block's function. For example, a filter block may include a scope plot for monitoring the filtered signal. In contrast, a learning block may offer buttons to train, reset, or pause a machine learning model, along with indicators for prediction confidence. To maintain consistency, each block's CP is inherited from an abstract base *ControlPanel* class and must override the *cpRefresh* method, which is responsible for redrawing the GUI. Updates occur only when the panel is visible and stop when the *Control Panel* is hidden, reducing unnecessary processing. Users can toggle the panels open and closed by left-clicking the corresponding block in the *BlockTable*. Standard visualization tools integrated into *Control Panels* include:

- *Scope Monitors*: Display time-series plots of signal vectors in real time (see Fig. 3).
- *Spider Plots*: Offer radial visualizations for multidimensional data (see Fig. 3).
- *Heatmap*: Enable real-time heatmap visualization of data matrices.
- *Label Displays*: Show numeric or textual indicators for debugging and status monitoring

D. SCHEDULING AND TIMING ACCURACY

Ensuring precise timing for signal processing is challenging unless a real-time operating system is used. Standard Windows timers are prone to low resolution and unpredictability, often introducing jitter or delays that exceed the acceptable thresholds for interactive systems. To address this, *MOSAIC* includes a custom-built *Scheduler* class that serves as the backbone of all time-critical operations. The *Scheduler* implements a single, continuously running asynchronous loop that manages the function callbacks registered with a specified target rate. Each function is scheduled based on an internal timestamp and rescheduled by increasing its firing time by $1/f$, where f is the desired frequency. This approach bypasses the limitations of default system timers, thereby enabling high-resolution execution. Blocks that require periodic activation (e.g., timers and signal generators) register their callback functions with the *Scheduler*. Execution is

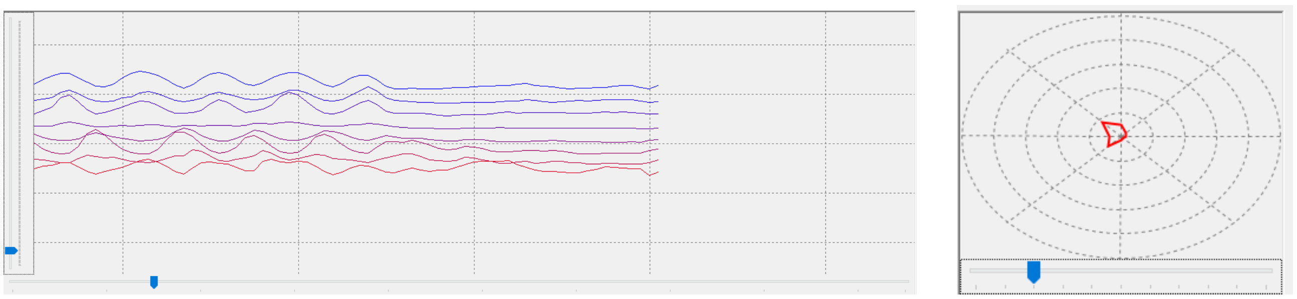

Figure 3. **Scope Monitor (left) and Spider Plot (right) visualization.**

driven in an event-based manner, which minimizes CPU usage and enables smooth performance even at high sampling rates (e.g., 2000 Hz). Furthermore, each block maintains its own circular timestamp buffer to estimate its real execution rate over the most recent 100 samples. This allows *MOSAIC* to detect and visualize lagging or stumbling behavior in the *BlockTable*. By combining centralized scheduling, local timestamp buffers, and a visual diagnostics system, *MOSAIC* achieves a practical compromise between the responsiveness of real-time systems and the accessibility of general-purpose operating environments.

To provide a consistent pacing mechanism for the pipeline, *MOSAIC* includes timer blocks. Each block inherits from an abstract base class *Timer*, to ensure consistency. The following timers are implemented:

- *ScheduledTimer*: Registers its firing function with the scheduler to provide a regular pacing signal that drives the activation of other blocks. It serves as the primary timing source to start and coordinate the pipeline execution.
- *DecimatorTimer*: Subscribes to another timer's output and forwards only part of its pacing events, effectively downsampling the pace for parts of the pipeline that require a lower activation rate.

These Timer blocks provide a modular and flexible means to control the cadence of data acquisition, processing, and output within the block graph.

E. STREAMING DATA

The *MOSAIC* software suite supports real-time data acquisition from a wide range of biosignal sensing devices. To accommodate the diverse landscape of biomedical research and commercial hardware, each device is encapsulated in a dedicated block that abstracts communication, streaming parameters, and signal visualization. The modular approach simplifies device integration and ensures uniform interaction with the processing pipeline, regardless of the underlying hardware. For devices that are not natively supported, *MOSAIC* offers hardware-agnostic interfaces, including UDP and serial communication.

1) ELECTROMYOGRAPHY SENSORS

In the field of assistive robotics, sEMG is one of the most widely used sensing modalities; consequently, *MOSAIC* has incorporated various sEMG sensors.

The *Myo* bracelet (Thalmic Labs, Canada) [40] is a commercially available wireless sEMG sensor array comprising eight bipolar channels and an onboard IMU. *MOSAIC* can connect to one or more Myo armbands via Bluetooth Low Energy (BLE), maintaining a stable sample rate of 200 Hz. The corresponding Myo block offers automatic device scanning, connection management, and live signal preview through its integrated CP, which includes a scope monitor and connection status indicators.

The *MuoviPro* (OT Bioelettronica, Turin, Italy) [41] high-density system supports up to 128 channels via four Wi-Fi-connected probes managed through a SyncStation. *MOSAIC* interfaces with the MuoviPro, allowing up to 2000 Hz sampling across multiple channels. Visualization options include real-time heatmaps, spider plots, and classic scope views. Each data stream is timestamped using the internal scheduler, ensuring high-precision analysis. Although OT Bioelettronica also offers a 20-sensor EEG cap compatible with MuoviPro, this functionality has not been tested with the present version of the software suite.

For high-resolution, clinical-grade sEMG, *MOSAIC* integrates with the *Trigno Delsys System* (Delsys, Natick, USA) [42] product line (Base Station, Maize, and Quattro sensors) using the manufacturer's software development kit (SDK). The Delsys block handles device discovery, configuration, and channel-specific streaming modes. Visual feedback includes time-series plotting, channel selection, and streaming diagnostics, all of which are accessible through the device's dedicated control panel.

2) DIGITAL TO ANALOG CONVERTERS

To integrate data from various sensor modalities, *MOSAIC* includes interfaces for specific models of commercially available and self-produced data acquisition boards. These include GSV-8 (ME-Meßsysteme, Hennigsdorf, Germany), MCC USB-1208FS-Plus/1408FS-Plus Series (Measurement Computing Corporation, Norton, USA), and a wireless data acquisition board of the German Aerospace Center (DLR, Wessling, Germany) [43]. These devices enable the connection to various analog sensors. These include the K6D and F6D force torque sensor series (ME-Meßsysteme, Hennigsdorf, Germany), the Finger Force Linear Sensor (FFLS) [44], FMG sensors, and force sensitive resistors [43], [45], [46].

3) ULTRASOUND

The Fraunhofer IBMT Mobile Ultrasound Equipment (MoUsE) (Fraunhofer Institute for Biomedical Engineering, Sulzbach, Germany) [47] system is integrated into *MOSAIC* to support real-time ultrasound-based biosignal acquisition. This compact research-grade device includes a 32-element phased-array transducer operating at 3 MHz, with per-channel sampling rates of up to 50 MHz at 12-bit resolution.

4) INERTIAL MEASUREMENT UNIT

In addition to the IMUs found in the aforementioned sEMG devices, *MOSAIC* interfaces with a custom wearable IMU platform for body tracking called BodyRig [48]. The BodyRig system consists of multiple Bluetooth Low Energy (BLE) modules, each equipped with an IMU, a BLE microcontroller, and a battery. These modules communicate wirelessly with a host PC and are used to track human kinematics in real-time. The number of sensors used can be customized, allowing for tracking of individual body parts or the entire body.

MOSAIC interfaces with BodyRig through the Bluetooth Serial Port Protocol. The block then performs forward kinematics calculations based on the orientations of the IMU sensors, as well as a stored set of parameters. These define the user's kinematic chain through parameters such as link length, relative orientation of the sensors with respect to the body segment, and the parent of each link. The same type of forward kinematics calculation may also be applied to data gathered from other types of IMU, such as the ones present in the MUOVI probes. The modular design of *MOSAIC* enables kinematic chain reconstruction, phase tracking, and avatar mapping within custom virtual reality environments.

5) HARDWARE AGNOSTIC INTERFACES

MOSAIC offers a suite of hardware-agnostic interface blocks that facilitate integration with a diverse selection of sensors and devices. These blocks abstract the complexities of device communication, allowing seamless incorporation into the *MOSAIC* processing pipeline.

The *SerialPort* base class provides a straightforward interface for serial communication with microcontroller-based sensors and wearable devices. It supports configurable parameters such as baud rate, data bits, stop bits, and parity, ensuring compatibility with various serial devices.

A *UDP* base class enables real-time data reception over network interfaces. It parses structured data packets transmitted via UDP, supporting both local and network-wide communications. This is particularly useful for integrating custom hardware or software that streams data over a network.

The *BluetoothLEManager* base class facilitates communication with BLE devices. It handles device scanning, connection establishment, and data exchange, adhering to the Generic Attribute Profile (GATT) protocol. This block is instrumental in integrating BLE peripherals, such as heart rate monitors, motion sensors, or custom BLE-enabled devices, into the *MOSAIC* ecosystem.

F. SIGNAL PROCESSING AND TRANSFORMATION BLOCKS

MOSAIC provides a range of signal processing blocks designed to transform, segment, or condition data in preparation for downstream machine learning or control modules. These blocks operate independently for each data stream, supporting both real-time and batch pipelines. Each processing block encapsulates a specific signal operation, such as filtering, normalization, feature generation, or windowing, and exposes its behavior through the YAML parameters. The most commonly used processing blocks are described below.

The *Filter* block implements a digital Infinite Impulse Response (IIR) or Finite Impulse Response (FIR) filter, i.e., it realizes an FIR if $Q = 0$ and an IIR if $Q > 0$. It applies a recursive filter to each channel independently using the standard difference equation:

$$y[n] = \sum_{i=0}^{P} m_i\, x[n-i] - \sum_{j=1}^{Q} d_j\, y[n-j] \tag{1}$$

where

- $y[n]$ is the output signal at sample n,
- $x[n]$ is the input signal at sample n,
- m_i are the numerator (feedforward) coefficients, for $i = 0, \ldots, P$,
- d_j are the denominator (feedback) coefficients, for $j = 1, \ldots, Q$,
- P is the numerator order (number of feedforward taps minus one),
- Q is the denominator order (number of feedback taps),
- n is the discrete time index (sample number).

The coefficients are provided explicitly as numeric arrays via the YAML configuration: $m = [m_0, \ldots, m_P]$ and $d = [1, d_1, \ldots, d_Q]$ (with $d_0 = 1$ by convention; FIR if $Q = 0$). The implementation uses a circular buffer to efficiently store and apply the required input and output samples for each channel independently.

The *AdaptiveFilter* block implements a single-pole IIR filter with a cutoff frequency that dynamically adapts to the characteristics of the input signal [49]. In contrast to fixed-parameter low-pass filters, this filter adjusts its smoothing behavior in real time based on both the amplitude and rate of change of the input signal, exploiting the heteroscedastic nature of sEMG data.

At each timestep, the cutoff frequency $F_c \in \mathbb{R}$ is recalculated using the instantaneous magnitude of the input signal $x(t)$ and the magnitude of its first derivative $\dot{x}(t)$ as follows:

$$\begin{aligned} y[n] &= LP(x[n], \alpha[n]) \\ &= \alpha[n]\, x[n] + \big(1 - \alpha[n]\big) y[n-1] \end{aligned} \tag{2}$$

$$\alpha[n] = \frac{2\pi F_c[n] \Delta T}{1 + 2\pi F_c[n] \Delta T} \tag{3}$$

Table 1. Supported operations of the *Function* block.

Operation	Description
Addition	Adds a constant or another vector element-wise.
Multiplication	Multiplies by a constant or another vector element-wise.
Absolute value	Converts each element to its absolute value.
Exponentiation	Raises each element to a specified power.
Clipping	Restricts each element within a specified minimum and maximum range.

$$F_c[n] = \exp\left(G_2 \cdot |x[n]| + G_1 \cdot LP\left(|\dot{x}[n]|, B_1\right) + B_2\right) \quad (4)$$

With additional parameters:

- $\dot{x}[n]$ is the time derivative of $x[n]$,
- ΔT is the sampling period,
- G_1, G_2 are gain parameters,
- B_1, B_2 are bias parameters,
- $LP(\cdot, p)$ denotes a low-pass filter with smoothing parameter p,
- $\alpha[n]$ is the adaptive smoothing factor determined at each time step n.

Equation (2) defines the recursive smoothing operation, which blends the current input $x[n]$ with the previous output $y[n-1]$ using the adaptively calculated α. This adaptive approach allows the filter to respond rapidly to sudden, high-energy transitions (such as muscle activation bursts) while maintaining strong attenuation of background noise during periods of low changes in muscular activity, such as a quasi-isotonic contraction. As a result, the *AdaptiveFilter* block is particularly well suited for mobile or wearable applications, where signal fluctuations and artifacts are common.

The *Buffer* block temporally segments data into labeled windows or epochs based on external trigger signals. When activated by a trigger block, it subscribes to a data provider, collects multidimensional data samples, and unsubscribes when instructed. The collected data is stored as a list of (target value, data cluster) pairs, which is used for batch machine learning. The buffer's control panel allows users to inspect, clear, or save collected data during runtime.

The *Function* block applies a user-defined mathematical operation to its input vector element-wise. The specific function is configured dynamically via the YAML file. Upon receiving a new input, the function evaluates each vector component independently, enabling flexible feature computation or signal transformation. A summary of supported operations is presented in 1.

The *VectorialFunction* block extends the capabilities of the *Function* block to support more complex vector operations, including operations involving one or two input sources. Supported operations include rotation (applying roll, pitch, and yaw transforms), absolute value (computing the L2 norm), wrench projection, element-wise addition and subtraction of vectors, cross product (for 3D vectors), scalar (dot) product, matrix multiplication, and time integration (see 2). The specific operation is defined in the YAML configuration, and the block processes inputs dynamically to compute the resulting vector.

Table 2. Supported operations of the *VectorialFunction* block.

Operation	Description
Rotation	Applies roll, pitch, yaw transform to input vector.
Absolute value	Computes the L2 norm of the input vector.
Wrench projection	Projects a wrench using a configurable matrix.
Add vectors	Adds two input vectors element-wise.
Subtract vectors	Subtracts the second input vector from the first.
Cross product	Computes the cross product of two 3D vectors.
Scalar (dot) product	Computes the dot product of two input vectors.
Matrix multiply	Multiplies the input vector by a projection matrix.
Integrate	Integrates the input vector over time.

G. MACHINE LEARNING INTEGRATION

MOSAIC supports both batch and incremental machine learning through dedicated, modular predictors (see Fig. 4). The *BatchPredictor* module enables offline model training using buffered signal segments, whereas the *Incremental-Predictor* module allows for continuous model adaptation during runtime. These learning blocks are structured under a unified *PredictingMachine* abstraction, which standardizes the interface behavior for prediction, training, and configuration. Concrete implementations include Ridge Regression (see Equations (5) and (6)) and its nonlinear extension using Random Fourier Features (RFF) [50]. Both models are available in batch and incremental variants, facilitating flexible experimentation with linear and kernelized estimators.

$$\hat{y} = \mathbf{x}^{\mathsf{T}}\mathbf{W} \quad (5)$$

$$\mathbf{W} = \left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^{\mathsf{T}}\mathbf{y} \quad (6)$$

where:

- $\mathbf{X} \in \mathbb{R}^{N\times D}$: matrix of N training samples with D features
- $\mathbf{y} \in \mathbb{R}^{N}$: vector of target values
- $\lambda > 0$: regularization parameter
- $\mathbf{I}$ is the $D \times D$: identity matrix
- $\mathbf{W} \in \mathbb{R}^{D}$: learned weight vector
- $\mathbf{x} \in \mathbb{R}^{D}$: new input sample
- $\hat{y}$: predicted target output

To support supervised learning pipelines, utility blocks such as *Buffer* and *Trigger* manage windowing, class labeling, and training initiation. Each learning block includes a live control panel for parameter adjustment, training

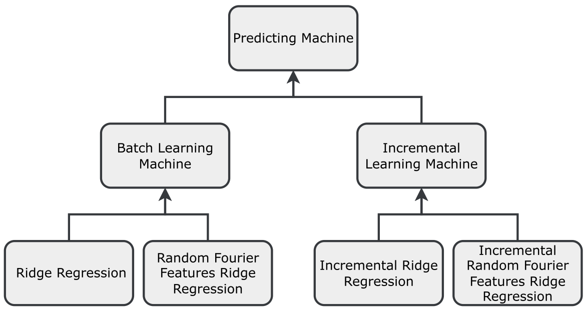
Predicting Machine
Batch Learning Machine
Incremental Learning Machine
Ridge Regression
Random Fourier Features Ridge Regression
Incremental Ridge Regression
Incremental Random Fourier Features Ridge Regression

Figure 4. Simplified class diagram showing the hierarchical structure of the learning modules in *MOSAIC*. All models inherit from a common PredictingMachine base, with batch and incremental learning variants implemented through ridge regression and random fourier feature extensions.

feedback, and visualization of prediction results. Models can be trained, updated, or reloaded during live sessions, enabling human-in-the-loop training and subject-specific model personalization. This modular structure promotes reproducibility and cross-study comparability across a wide range of biosignal-driven applications.

H. CONTROL ALGORITHM BLOCK FOR ACTUATION MAPPING

The *ControlAlgorithm* block provides the interface between the prediction outputs and device-level actuation in *MOSAIC*. It transforms prediction vectors, for example, from a *RidgeRegression* model, into control commands that drive external devices such as prosthetic hands or orthoses. This block seamlessly integrates into the event-driven architecture of *MOSAIC*, ensuring that actuation logic remains modular and independent from the upstream signal processing and prediction components.

The *ControlAlgorithm* block currently supports two paradigms, selectable via YAML configuration:

- *DirectControl*: Map predictions proportionally to actuation commands, enabling position-based or force-based control.
- *StepwiseControl*: Adjusts actuation values incrementally in response to predictions, supporting velocity-based or discrete force adjustments.

The design follows the software design structure of a C# interface. Each control strategy implements a common interface, ensuring consistent behavior and allowing the system to dynamically select and apply different control strategies at runtime. This interface-based paradigm promotes modularity and extendability, as new control strategies can be integrated without altering the core structure of the block or the surrounding pipeline.

I. SAVING AND SENDING DATA

In *MOSAIC*, data saving is designed to be flexible and fully integrated into the modular architecture. Any block in the processing pipeline can be configured to log its output by specifying a file path using the *Path* parameter in the YAML configuration file (see Subsection III-A). When enabled, the block writes its output to a plain text file, where each line contains a UNIX timestamp, followed by the output values. This approach allows users to capture raw or processed signals at specific stages of a pipeline for offline analysis, debugging, or benchmarking purposes. To ensure performance and data integrity, output logging is implemented with memory-buffered disk writes, enabling high-throughput storage without interfering with real-time execution.

MOSAIC provides dedicated communication blocks to send data to external systems. In particular, the *UDP_StreamlinedSender* block [51] implements a standardized protocol for transmitting data over UDP to external systems, such as simulators or robotic platforms. The block subscribes to upstream components (e.g., *Myo* or *ControlAlgorithm*) and packages data according to a predefined structure that includes metadata on data type, category, timestamp, and counter information. This ensures that downstream systems can reliably interpret the transmitted information. The *UDP_StreamlinedSender* is configured via YAML, allowing users to specify the target IP addresses, ports, and connected input sources without altering the underlying code. This modular design facilitates hardware-agnostic integration with both physical devices and virtual environments, thereby supporting a wide range of assistive and rehabilitation applications.

The *ROS* block provides native integration between *MOSAIC* and ROS. It establishes a WebSocket connection to a ROS bridge server, enabling the software to either publish data or subscribe to data from specified ROS topics in real-time. The block supports standard ROS message types, such as *std_msgs/Float64MultiArray*, and can be configured via YAML to define the target IP address, topic name, message type, and communication direction (publish or subscribe). This modular design enables seamless interoperability with robotic systems, simulators, and external control frameworks, supporting advanced assistive and rehabilitation applications without requiring additional middleware development. The ROS block integrates seamlessly into the event-driven architecture of *MOSAIC*, enabling flexible real-time communication between the two frameworks.

J. INTEGRATION OF PYTHON LIBRARIES AND CUSTOM PYTHON SCRIPTS

Many widely used libraries for data processing, statistics, and machine learning have been written in Python. Additionally, various research groups have developed custom code or open-source packages in Python, such as LibEMG [30]. To facilitate the reuse of these resources within the *MOSAIC* environment, we integrated Python.Net Version 3.0.4 [56]. The PythonNet integration is managed by the *PythonNetManager*, which configures the Python environment. Dependency management is handled via a Poetry environment [57], providing a robust and reproducible setup. This approach

Table 3. **Overview of available blocks in *MOSAIC*: categorized and listed alphabetically.**

Actuation and Control Blocks	
ControlAlgorithm	Maps prediction outputs to actuation commands using selectable strategies.
Communication Blocks	
BluetoothLEManager	Manages BLE connections.
HannesHand	Communicates with and controls the IIT Hannes hand prosthetic [52] via serial/Bluetooth, transmitting reference commands for multiple degrees of freedom.
ROS	Connects to ROS via WebSocket for real-time topic publish/subscribe.
SerialPort	Provides serial communication for external device integration.
UDP_StreamlinedSender [51]	Sends data via UDP in a standardized format.
Device Interface Blocks	
BodyRig	Streams orientation data from multiple IMUs for body tracking. Alternatively, it takes orientation data streaming through e.g. UDP and performs forward kinematics calculations.
Delsys	Clinical-grade sEMG acquisition with Delsys devices.
MuoviPro	High-density sEMG acquisition via a single Wi-Fi probe.
MuoviSyncstation	High-density sEMG acquisition via an OT Bioelettronica Syncstation with up to four Wi-Fi probes.
Myo	Acquires sEMG and IMU data from Myo armband.
Flow Control and Utility Blocks	
ArtifactRejection	Performs real-time, rule-based artifact detection on windowed signals (e.g., clipping, flatline, impulsive jumps; optional low-frequency drift). Outputs a per-channel mask and a cleaned stream; supports hard (zero bad channels) or soft (pass with mask) modes with configurable thresholds and cumulative counters for reproducibility.
ChannelSelector	Dynamically activates, deactivates, or zeroes input vector channels based on user selection, supporting flexible signal shaping.
DecimatorTimer	Downsamples pacing signals from other timers.
FilePlayer	Reads and plays back data from file sources in sync with timer ticks, outputting one vector per tick.
Joiner	Combines multiple vector streams into one output.
Logger	Flexible data logging block.
Projector	Extracts and forwards a specified subset of elements from an input vector, reordering as defined in the configuration.
PythonNetManager	Sets up the PythonNet Environment for MOSAIC and imports the poetry-managed dependencies and custom scripts to be available in all blocks.
Resampler	Adjusts the data flow rate using a sample-and-hold method, resampling input signals according to a timer-driven schedule.
RMS	Computes channel-wise root-mean-square over a sliding window to estimate signal magnitude; window length and hop are configurable, outputting one value per channel.
ScheduledTimer	Provides regular pacing signals to drive the pipeline.
Selector	Receives a tuple as input and forwards only the second element (e.g., a vector) to downstream blocks.
SinGenerator	Generates synthetic sinusoidal signals with configurable amplitude, frequency, phase, and component count for testing and simulation.
Splitter	Divides a vector stream into multiple outputs.
Stimulus	Stimulates and manages task timing, presenting visual cues.
Switch	Dynamically toggles between two input blocks, forwarding data from the active input to downstream blocks.
TAC	Target Achievement Control (TAC) test [53], evaluates task performance by comparing predicted output to target values, monitoring distance and time in target to trigger state transitions.
Machine Learning Blocks	
BatchPredictor	Provides batch-trained machine learning model inference.
IncrementalPredictor	Supports online adaptive learning for real-time prediction.
Signal Processing Blocks	
AdaptiveFilter	Dynamically adjusts the cut-off frequency based on signal characteristics.
Buffer	Segments data into labelled epochs for machine learning.
Filter	Applies IIR or FIR digital filters to signals.
Function	Applies element-wise mathematical operations to input vectors.
MeanAbsoluteValue	Computes the mean absolute value [54] of each channel in a windowed signal.
SlidingWindow	Applies a window function over streaming data (feature extraction, smoothing, framing).
SlopeSignChanges	Computes the number of slope sign changes [55] in a windowed signal.
VectorialFunction	Performs vector-level operations (e.g. cross product, matrix multiplication).
WaveformLength	Computes the waveform length [55] for EMG feature extraction.
ZeroCrossings	Counts zero crossings [54] in each channel of a windowed input.
Visualization Blocks	
HeatmapVisualizer	Displays multidimensional data as a real-time heatmap.
SpiderPlotVisualizer	Radial plot of multidimensional data (spider plot).
ScopeMonitor	Real-time time-series plotting of signals.

allows Python scripts and libraries to be embedded as modular components within *MOSAIC* pipelines, thereby enhancing the flexibility of the framework while maintaining its intended modular block structure.

After initializing the *PythonNetManager*, users can import custom Python scripts or libraries as modules and access the functions within any block. This capability enables the application of advanced machine learning algorithms or pre-existing Python functions in *MOSAIC*. However, it is essential to note that real-time performance depends significantly on the efficiency of the Python functions being executed. For computationally intensive tasks, performance testing is advised to ensure that Python code meets timing constraints within *MOSAIC's* real-time environment. To demonstrate the versatility of this integration, we include four representative use cases:

- *Example 1*: demonstrates the online application of a Butterworth filter using `scipy.signal` [58], showcasing standard Digital Signal Processing operations inside *MOSAIC*.
- *Example 2*: implements a simple Python script that allows the user to choose between Independent Component Analysis, Principal Component Analysis, or LDA based on their Scikit-Learn [59] implementation.
- *Example 3*: shows a minimal example of loading a PyTorch [60] model and using it for inference inside *MOSAIC*.
- *Example 4*: illustrates the integration of the Incremental Nonnegative Matrix Factorization (iNMF) algorithm for the unsupervised extraction of muscle synergies. This approach is based on [61] and was further developed and integrated in [62].

It is important to note that *MOSAIC* guarantees deterministic scheduling for its core blocks; however, the pipeline operates on an event-based system where downstream blocks are triggered by outputs from upstream blocks. Therefore, a user-defined Python Code called in the *PythonNetManager* block must complete its tasks within a specified time budget for each update (for example, less than or equal to 40 ms at 25 Hz). If the Python block exceeds this time limit, its own processing rate decreases, causing all downstream subscribers to that block to be delayed and marked as lagging or stumbling in the *BlockTable*. Meanwhile, other branches that do not depend on the Python block will proceed on time. The runtime does not deadlock the graph; instead, it highlights the violation and propagates the delayed events along the affected branch.

K. DOCUMENTATION

Although *MOSAIC* was initially developed and maintained as an internal research and teaching tool, it is thoroughly documented to support long-term use, reproducibility, and future collaboration. The codebase includes extensive inline comments, as well as a structured Application Programming Interface (API) reference that covers core classes, parameters, and integration patterns. Table 3 summarizes the available blocks, including both major components described in detail in this paper and additional utility blocks that support pipeline design, data handling, and visualization.

To support onboarding and experimentation, example YAML files and explanations are available for common workflows, including configuring signal pipelines, designing custom blocks, and integrating machine learning models. These resources collectively aim to make *MOSAIC* accessible and expandable.

The Documentation and API, including examples and installation guidelines, can be found https://github.com/AIROB-Lab/MOSAIC.

IV. USE CASES

Multiple published studies have adopted *MOSAIC* as their execution backbone, reusing the YAML-specified block graphs to ensure reproducibility and live timing transparency. Applications include myoelectric prosthesis control, C-arm gesture control, lower-limb orthosis assistance, tele-impedance, and FES. A structured summary of hardware, modules, and target systems is given in 4; brief narratives follow.

A. UPPER-LIMB PROSTHESIS CONTROL

Egle et al. [63], in collaboration with the Istituto Italiano di Tecnologia (IIT, Genoa, Italy), compared two incremental-learning myocontrol strategies for 3-DoF prosthetic hand control: direct position control and stepwise velocity control. A 16-channel sEMG pipeline (two Myo armbands) fed an incrementally updated Ridge Regression model. Two non-disabled participants completed eight sessions (four per strategy) using a TAC protocol. *MOSAIC* handled acquisition, feature extraction, online learning, and UDP streaming to a virtual prosthesis; *IncrementalPredictor*, *Trigger*, *Buffer*, and *Stimulus* coordinated training and guidance, while the *BlockTable* and control panels supported monitoring. Success increased from 37%→71% (position) and 48%→91% (velocity); SUS scores were 75 and 82.5, and NASA-TLX decreased to 38 and 35, respectively—demonstrating practical real-time adaptation with *MOSAIC*.

B. HAND GESTURE C-ARM CONTROL

Ouyang et al. [64], with Siemens Healthineers (Forchheim, Germany), introduced touchless positioning of mobile surgical C-arms using a Mindrove armband (sEMG+IMU). A Ridge Regression–based myocontrol predicted gesture activations from eight-channel sEMG; a safety/robustness layer permitted only the dominant gesture and applied motion constraints. Forearm orientation from the IMU switched modes between platform translation and C-arm rotation. Implemented in *MOSAIC*, the system used *IncrementalPredictor*, *Buffer*, and *Stimulus*, with live visualization via control panels and the *BlockTable*. In a TAC study with six participants (three rounds, five randomized target poses each), average success rate was 11.1% but position/rotation

Table 4. **Summary of published studies implemented using *MOSAIC*.**

Use Case	Hardware Used	Key *MOSAIC* Modules	Target System
Upper-Limb Prosthesis Control	2× Myo bracelet (sEMG)	*Timer*, *Filter*, *Function*, *IncrementalPredictor*, *Buffer*, *Trigger*, *Stimulus*, *UDP Streamlined Sender*	3-DoF virtual prosthesis control in Blender
C-Arm Gesture Control	Mindrove Armband (sEMG + IMU)	*Timer*, *Filter*, *Function*, *IncrementalPredictor*, *Buffer*, *Trigger*, *Stimulus*, *UDP Streamlined Sender*	Simulated C-arm interface
Lower-Limb Orthosis Control	Myo bracelet (sEMG)	*Filter*, *Function*, *Buffer*, *IncrementalPredictor*, *Trigger*	Active orthosis model
3D Tele-Impedance Stiffness Estimation	Delsys Avanti sEMG (8 channels), BodyRig IMUs, 6-axis force/torque sensor	*Timer*, *Filter*, *Buffer*, *Function*, *Vectorial Function*, *IncrementalPredictor*, *Kinematics (BodyRig)*, *UDP*	2× Franka Research 3 robotic arms
FES-mediated impedance control of human limbs	BodyRig IMUs, 6-axis force/torque sensor	*Timer*, *Filter*, *Buffer*, *Function*, *Vectorial Function*, *IncrementalPredictor*, *Kinematics (BodyRig)*, *UDP*	MyoCeption wearable FES system
Real-time control of virtual hand by spinal cord injury patients	OTBioelettronica Quattrocento hdEMG system	*Timer*, *Function*, *Filter*, *Adaptive Filter*, *IncrementalPredictor*, *UDP*	4-DoF Virtual hand control

errors decreased over time; perceived safety was high (3.9/5), indicating feasibility and a learning effect despite high workload.

C. LOWER-LIMB ORTHOSIS CONTROL

Scheidl et al. [65] developed adaptive control for a lower-limb orthosis during sit-to-stand (STS). Four Delsys Avanti EMG sensors per thigh streamed at 2148 Hz; *MOSAIC* handled filtering, rectification, and averaging, followed by Ridge Regression via a *BatchPredictor*. Control signals were sent via UDP to the orthosis controller. Control panels (scopes, spider plots, heatmaps) supported visualization; synchronized logging enabled analysis. Results showed promising real-time augmentation of knee extensor output and support for human-in-the-loop optimization, highlighting the portability of block reuse across lower-limb applications.

D. 3D CARTESIAN STIFFNESS ESTIMATION FOR TELE-IMPEDANCE

Thuerauf et al. [66] introduced a method to estimate human arm stiffness in 3D Cartesian space for tele-impedance. Two non-disabled participants wore eight-channel sEMG (Delsys Trigno Avanti) and multiple IMUs (BodyRig) while performing peg-in-hole tasks at orientations of 0°, 45°, and 90° around the sagittal axis. The protocol comprised (a) MVC calibration, (b) robot-driven perturbations for direct stiffness measurement (control), and (c) the task itself. *MOSAIC* handled synchronization of the relevant biosignals and the data streams coming from the Franka Research 3 robots used in the experiment. Offline, the pseudo-stiffness matrices were computed in from sEMG and transformed to Cartesian space using IMU-derived kinematics. Repeated-measures ANOVA and Wilcoxon tests showed significant orientation-dependent differences; pseudo-stiffness correlated with perturbation-based estimates ($R = 0.62$ and $R = 0.80$), indicating feasibility of 3D stiffness estimation with biosignal-driven pipelines.

E. IMPEDANCE CONTROL OF HUMAN LIMBS THROUGH FUNCTIONAL ELECTRICAL STIMULATION

Sierotowicz and Castellini [67] demonstrated the use of *MOSAIC* to implement a control system regulating the injection of Functional Electrical Stimulation (FES) currents. These were used to control the movements of a cohort of six non-disabled participants on a planar surface, for the purpose of acquiring normative data characterizing the implemented solver, which would determine which muscles to stimulate given a desired motion in the peri-personal space. The study employed an internally developed wearable FES system, called the MyoCeption [68], as well as the BodyRig kinematics tracker and a force torque sensor by ME-Messysteme. The study confirmed that the proposed solver can control human limb movements on a planar surface with a precision in the order of magnitude of 0.1 m.

F. REAL-TIME CONTROL OF VIRTUAL HAND BY SPINAL CORD INJURY PATIENTS

Oliveira et al. [69] employed *MOSAIC* to allow spinal cord injury patients to control a virtual hand in real time. Eight patients participated in the study, of whom three were rated A on the American Spinal Injury Association (ASIA) impairment scale, three were rated B, and one was rated C. The participants were fitted with hdEMG electrodes, and *MOSAIC* was used to control the virtual hand's index flexion and power grasp independently and proportionally by making use of incremental ridge regression. The setup also employed an adaptive filter [49] to stabilize the control signals sent to the virtual hand over UDP. The overall results of the study demonstrate that spinal cord injury patients rated as completely paretic in their hand movements could control the available DoFs independently and proportionally.

V. EVALUATION

We quantitatively evaluated *MOSAIC* with respect to timing precision, computational load, and robustness across three representative online pipelines. The primary application is upper-limb sEMG control using the sensor placement and workflow described in Subsection IV-A. We consider: (a) a Thalmic Labs *Myo* armband (8 channels, 200 Hz), (b) a Delsys Trigno Avanti system (32 channels, 1926 Hz), and (c) Delsys Trigno Avanti combined with a deep-learning (DL) predictor. Unless stated otherwise, the DL condition uses the same sampling configuration as its corresponding Delsys acquisition. In the DL condition, all model-related processing runs in Python and is accessed from the C# runtime through *MOSAIC*'s `PythonNetManager` bridge (see Subsection III-J). The C# side performs device acquisition, buffering, and scheduling. When a fixed-length window becomes available, the windowed signals (sEMG and, when present, IMU) are transferred to Python as native arrays. The Python code performs any required preprocessing/normalization and then executes a ResNet-18–inspired network [70]. The resulting predictions (class probabilities or continuous estimates, depending on the task) are returned to C#, which feeds them to the control loop and visualization modules. Thus, the DL block is cleanly decoupled from acquisition: C# handles real-time I/O and orchestration, while Python encapsulates feature extraction and inference. For each pipeline, we report: iming jitter of the inter-sample interval ΔT on the acquisition side; and computational consistency, summarized by steady-state CPU usage. This setup isolates the cost of multichannel acquisition from the cost of model inference and allows us to attribute performance differences to specific components.

A. TIMING SANITY CHECK

The timing stability of each block is a critical metric for evaluating real-time systems. To assess this, we conducted a preliminary check by analyzing the temporal consistency between consecutive block outputs. For this evaluation, we used a Windows 11 laptop equipped with a 13th Gen Intel® Core™ i7–1365U processor, 32 GB Random Access Memory (RAM), and an NVIDIA® GeForce RTX™ 3070 Ti Laptop GPU (8 GB Video RAM). All non-essential programs were closed to isolate the performance
of *MOSAIC*.

We executed the pipelines while connecting them to a recording device during the training of the prediction models. Throughout this process, we recorded five minutes of signals, automatically saving data to a file every ten seconds. This approach helps prevent excessive RAM usage, as the RAM is cleared every ten seconds via a double-buffer structure, ensuring consistent memory usage without interrupting recording.

Once the pipeline execution was complete, we loaded the generated dump files and calculated the sample interval ($\Delta T \in \mathbb{R}$), defined as the difference between successive timestamps. This analysis provides direct insight into whether the block operates at its intended frequency and whether significant jitter occurs. To visualize real-time performance, we compute a 250-sample moving mean and its standard deviation (Fig. 5). The raw sEMG recordings from a Myo armband exhibit a stable mean throughout the recording, with the standard deviation remaining below 1 ms.

In addition to timing consistency, we systematically logged and evaluated the operational status of each block.

Table 5. Timing consistency (ΔT between consecutive outputs) for three online sEMG control pipelines: (a) Myo armband (8 ch, 200 Hz), (b) Delsys Trigno Avanti (32 ch, 1926 Hz), and (c) Delsys Trigno Avanti + DL with a 200-sample window and stride 77 (nominal 25 Hz).

Block	**Rate** (Hz)	**Mean** ΔT (s)	**Freq.** (Hz)	**Std** ΔT (s)
(a) Myo pipeline (200 Hz, **8 channels**)				
Scheduled Timer	200	0.0050001	199.9955	0.0002417
Myo	200	0.0050001	199.9978	0.0002641
Amplification	200	0.0050000	199.9993	0.0002837
Rectification	200	0.0050000	199.9993	0.0003017
Low Pass Filter	200	0.0049999	200.0042	0.0003385
Resampler	25	0.0400009	24.9994	0.0002944
Incremental Predictor	25	0.0400005	24.9997	0.0004624
Control Algorithm	25	0.0400005	24.9997	0.0004755
UDP Sender	25	0.0399848	25.0095	0.0009457
(b) Delsys pipeline (1926 Hz, **32 channels**)				
Delsys Trigno Avanti	1926	0.0005192	1926.0572	0.0003156
Amplification	1926	0.0005192	1926.0736	0.0003523
Rectification	1926	0.0005192	1926.0744	0.0003641
Low Pass Filter	1926	0.0005192	1926.1146	0.0003830
Resampler	25	0.0399797	25.0127	0.0020564
Incremental Predictor	25	0.0399795	25.0128	0.0022413
Control Algorithm	25	0.0399796	25.0128	0.0022767
UDP Sender	25	0.0399674	25.0204	0.0025883
(c) Delsys + DL (1926 Hz, **32 channels**)				
Delsys Trigno Avanti	1926	0.0005192	1926.0620	0.0003706
Window	25	0.0399779	25.0138	0.0032517
DL (Python)	25	0.0399384	25.0386	0.0026575
Control Algorithm	25	0.0399384	25.0386	0.0032718
UDP Sender	25	0.0399384	25.0386	0.00125333

B. PIPELINE TIMING RESULTS

As shown in Tables 5 and 6, blocks configured with fixed rates (e.g., 200 Hz and 25 Hz) exhibit high timing consistency, with standard deviations in the sub-millisecond range. An exception is the *Trigger* block, which activates incremental learning on user command when new labeled data become available; it has no fixed rate by design. Nevertheless, the pipeline demonstrates robust and predictable behaviour under real-time constraints.

In addition to timing jitter, we evaluated the CPU usage of *MOSAIC* under three representative online pipelines: (a) the Myo armband (8 channels at 200 Hz), (b) the Delsys Trigno system with eight wireless Avanti sensors for sEMG/IMU recording (four channels per sensor; 32 channels total at 1926 Hz), and (c) a high-rate Delsys Trigno Avanti configuration with an online DL inference process (32

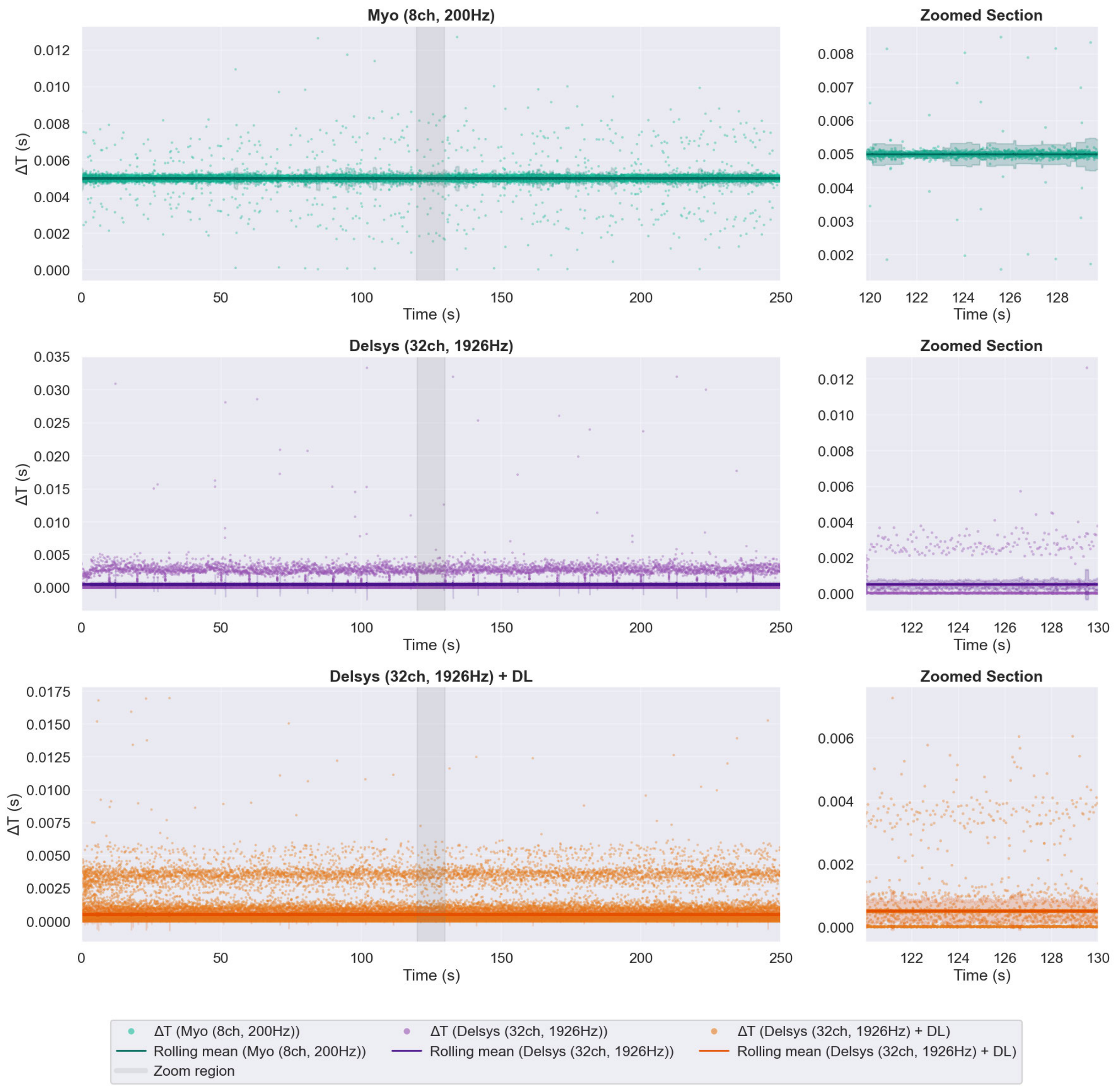

Figure 5. **Inter-sample-interval jitter Delta T for three online acquisition pipelines. Rows show (top) Myo armband (8 channels, 200 Hz), (middle) Delsys Trigno Avanti (32 channels, 1926 Hz), and (bottom) Delsys + DL (32 channels, 1926 Hz with decoder). Left panels cover the full recording; right panels zoom into the gray-highlighted window. Dots are raw Delta T values; solid curves are 250-sample rolling means; shaded envelopes denote the corresponding rolling standard deviation. All three pipelines operate close to their nominal sampling periods with limited, stationary jitter.**

channels at 1926 Hz). CPU usage was sampled at 1 Hz via Windows performance counters over 250 s of continuous operation; the first 25 s were discarded to exclude start-up and pipeline initialization. We report the steady-state median and one standard deviation over the remaining interval.

As shown in Fig. 6, the Myo pipeline required a median CPU load of 12.01 % ± 1.73 %, while the Delsys Trigno Avanti pipeline required 29.29 % ± 2.19 %. With the DL inference step, the *combined* load (system + Python process) was 36.92 % ± 4.40 %. On an 8-core machine, these values correspond roughly to ∼1.0, ∼2.3, and ∼3.0 fully loaded cores, leaving ample headroom for real-time operation. Overall, computational demand scales primarily with channel count and sampling rate; adding the DL step increases the steady-state CPU by about 7.6 percentage points relative to the Delsys Trigno Avanti only configuration.

C. COMPARISON WITH SIMULINK

To contextualize *MOSAIC* within a standard engineering framework, we compared its *computational load* against

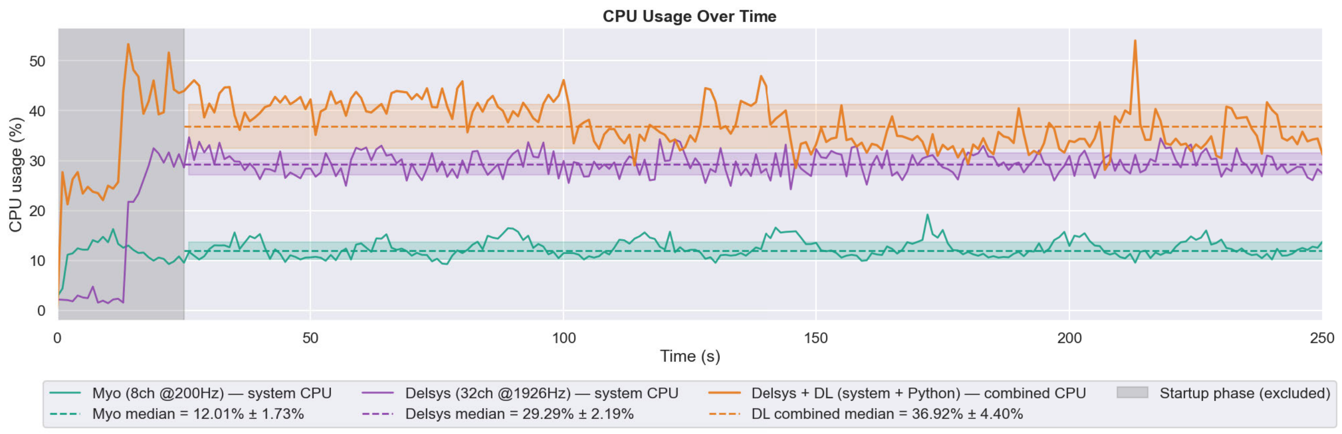

Figure 6. **CPU usage over time during online processing with *MOSAIC* for three pipelines: Myo (8 ch, 200 Hz), Delsys Trigno Avanti (32 ch, 1926 Hz), and Delsys Trigno Avanti + DL (32 ch, 1926 Hz). Solid lines show instantaneous CPU load; dashed lines and translucent bands show the steady-state median ± 1 SD after excluding the shaded start-up period (sensor registration and pipeline initialization). For Delsys Trigno Avanti + DL we report the *combined* CPU (system + Python process; an upper bound). Steady-state medians: Myo 12.01 % ± 1.73 %, Delsys Trigno Avanti 29.29 % ± 2.19 %, Delsys Trigno Avanti + DL 36.92 % ± 4.40 %.**

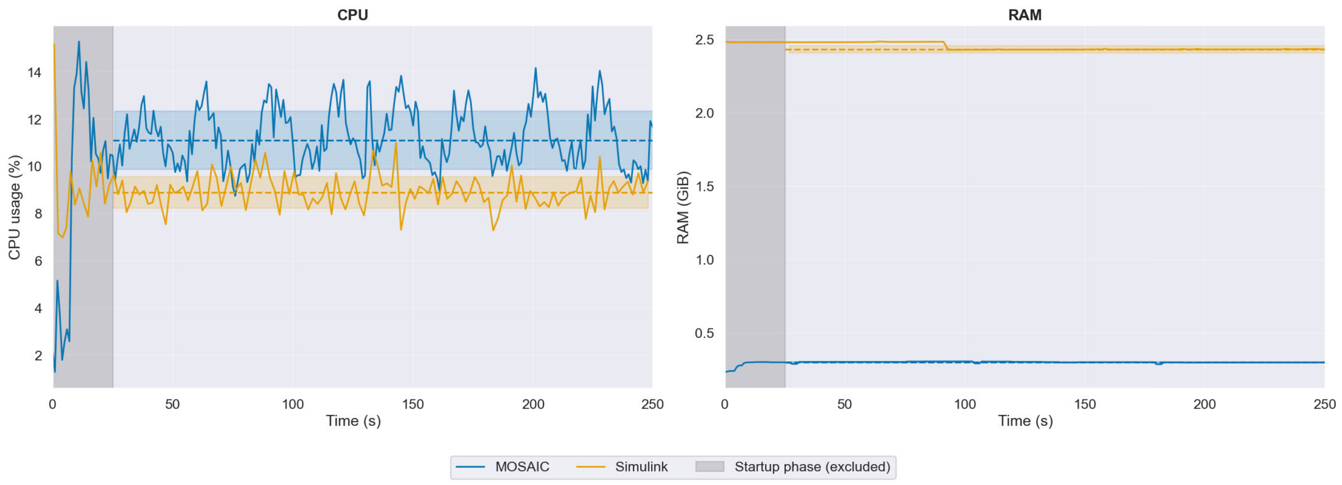

Figure 7. **CPU and RAM load comparison between *MOSAIC* and *Simulink* during 200 Hz Myo replay. Each trace shows instantaneous process load over 250 s of operation; the gray region marks the 25 s start-up phase excluded from steady-state statistics. Dashed lines indicate the steady-state median, and shaded bands denote ± 1 SD. On the same hardware, steady-state CPU usage was *MOSAIC*: 11.1 % ± 1.2 % and *Simulink*: 8.9 % ± 0.7 %, while RAM footprint averaged *MOSAIC*: 0.300 GiB ± 0.003 GiB and *Simulink*: 2.43 GiB ± 0.02 GiB.**

an equivalent Simulink pipeline using the same input data. Direct integration of the *Myo* armband into Simulink would have required substantial custom block development and dependency management. Therefore, we adopted a *record–replay* strategy: raw multichannel sEMG and IMU data were first recorded together with precise timestamps, and subsequently replayed in real time to both systems while preserving the original inter-sample timing. This design isolates processing and scheduling behavior from device-driver overhead, enabling a fair comparison of runtime efficiency.

We evaluated steady-state CPU usage and RAM footprint as indicators of computational load. Metrics were sampled at 1 Hz for a total duration of 275 s; we report steady-state over the last 250 s, excluding the initial 25 s start-up period (sensor registration and initialization). CPU usage is expressed as a percentage of the machine's total processing capacity (0–100%), and RAM as the process working set (in GiB). Both pipelines used equivalent signal-processing configurations, including identical windowing and filtering parameters. This comparison does not assess the integration of vendor DAQ systems within Simulink; rather, it quantifies the computational cost of executing equivalent control logic under identical timing constraints.

As shown in Fig. 7, itMOSAIC and Simulink exhibit complementary performance characteristics. Simulink achieved slightly lower steady-state CPU utilization (about 2 percentage points below *MOSAIC*), whereas its memory footprint was roughly an order of magnitude larger (2.43 GiB vs. 0.300 GiB). This difference reflects the MATLAB

Table 6. **Operational status distribution for each block in three online sEMG pipelines: (a) Myo (8 ch, 200 Hz), (b) Delsys Trigno Avanti (32 ch, 1926 Hz), and (c) Delsys Trigno Avanti + DL (32 ch, 1926 Hz; 200-sample window). Percentages indicate the proportion of the analyzed interval (first 250 s after excluding the initial 25 s start-up) spent in each status: G = green (on time), I = idle (not scheduled), L = lagging (behind schedule), S = stumbling (timing violation). "–" indicates that the status did not occur.**

Block	**G (%)**	**I (%)**	**L (%)**	**S (%)**
(a) Myo pipeline (200 Hz, **8 channels)**				
Scheduled Timer	100.000	–	–	–
Myo	100.000	–	–	–
Amplification	100.000	–	–	–
Rectification	100.000	–	–	–
Low Pass Filter	100.000	–	–	–
Resampler	100.000	–	–	–
Incremental Predictor	100.000	–	–	–
Control Algorithm	100.000	–	–	–
UDP Sender	100.000	–	–	–
(b) Delsys pipeline (1926 Hz, **32 channels)**				
Delsys Trigno	99.175	0.022	0.802	–
Amplification	98.800	0.037	1.163	–
Rectification	98.666	0.041	1.293	–
Low Pass Filter	98.319	0.046	1.635	–
Resampler	100.000	–	–	–
Incremental Predictor	100.000	–	–	–
Control Algorithm	100.000	–	–	–
UDP Sender	100.000	–	–	–
(c) Delsys + DL (1926 Hz, **32 channels)**				
Delsys Trigno	98.377	0.059	1.563	–
Window	100.000	–	–	–
DL (Python)	100.000	–	–	–
Control Algorithm	100.000	–	–	–
UDP Sender	100.000	–	–	–

runtime's higher baseline overhead. By contrast, *MOSAIC*'s C# execution engine and lightweight threading keep memory consumption low and predictable, which is advantageous for embedded, long-duration, or multi-process real-time control where computational headroom is constrained.

VI. DISCUSSION

MOSAIC is a YAML-configured, live-tunable runtime for biosignal-driven control. Pipelines are specified in human-readable YAML; each block exposes a lightweight control panel for in-run adjustments; and the runtime logs per-block timing status (on-time, lagging, stumbling). Both batch and incremental (online) learning are supported. The aim is a modular, extensible, hardware-agnostic platform enabling rapid experimentation and reproducible workflows across acquisition, processing, learning, and control. We situate *MOSAIC* among related ecosystems in 7.

MOSAIC offers block-based modularity comparable to Simulink while remaining open source. Simulink commonly relies on additional real-time products or external targets for hard real-time operation, whereas *MOSAIC* targets soft real-time on general-purpose operating systems using multithreading, asynchronous callbacks, and event-driven scheduling. YAML-based configuration is human-readable and version-controllable, in contrast to binary model files. A current limitation is the smaller catalog of pre-built blocks relative to commercial suites.

LabVIEW provides mature block-based design and low-latency I/O through close integration with NI hardware; it is commercial and vendor-centric. *MOSAIC* is hardware-agnostic and open source, which can reduce the entry barrier in education and research settings where vendor ecosystems are not a requirement.

Relative to ROS and ROS-Neuro, *MOSAIC* retains modularity with lower configuration overhead (YAML pipelines instead of launch files). A dedicated ROS communication block enables bidirectional exchange, allowing hybrid deployments that combine ROS tooling with *MOSAIC* pipelines. Python models are supported through the *PythonNetManager* block; concurrency-sensitive stages run in the.NET runtime, enabling use of libraries such as scikit-learn or PyTorch.

Compared with EMG-focused toolkits (BioPatRec, LibEMG), *MOSAIC* spans multiple biosignal modalities—EMG, IMU, ultrasound, force sensors, and generic streams via UDP/Serial/BLE—within a unified real-time GUI. BioPatRec depends on MATLAB and primarily supports offline or limited real-time operation. LibEMG targets interactive EMG control but does not provide integrated multi-modal sensor support and a unified GUI. In contrast, *MOSAIC* integrates streaming, processing, learning, and actuation mapping for interactive, human-in-the-loop systems. BioSPPy remains well-suited for offline analysis and feature extraction, but does not target real-time operation.

The quantitative evaluation (see Section V) indicates robust soft real-time behavior across three pipelines (Myo 200 Hz; Delsys Trigno Avanti 1926 Hz; Delsys Trigno Avanti + DL). Fixed-rate blocks tracked targets closely: acquisition and preprocessing at 1.926 kHz showed sub-millisecond ΔT variability, and 25 Hz downstream stages (resampling, prediction, control, UDP) exhibited millisecond-scale jitter relative to a 40 ms period. Operational-status logs align with this: in the Myo pipeline, all blocks were Green 100 %; in the high-rate Delsys Trigno Avanti pipeline, acquisition and preprocessing were Green 98.3 % to 99.2 % with $\leq$1.6 % Lagging and no Stumbling; in Delsys Trigno Avanti + DL, Delsys Trigno Avanti acquisition was Green 98.38 % and windowing, DL, control, and UDP stages were Green 100 %. Steady-state CPU medians were Myo (12.01 $\pm$ 1.73) %, Delsys Trigno Avanti (29.29 $\pm$ 2.19) %, and Delsys Trigno Avanti + DL (36.92 $\pm$ 4.40) %, corresponding to approximately 1.0, 2.3, and 3.0 fully loaded cores on an 8-core system. The DL branch sustained an effective 25 Hz update rate. These observations suggest predictable timing with rare, brief lag at the highest acquisition rates.

The record-replay benchmark conducted against Simulink (see Fig. 7) revealed a small difference in steady-state CPU

Table 7. Comparison of biosignal processing frameworks.

Framework	Platform / Language	Real-Time	Open Source	Hardware Integration	Typical Domain
Simulink	MATLAB GUI	Partial (with RT add-ons)	No	MATLAB hardware support packages	Control systems, embedded devices, signal processing
LabVIEW	GUI (G language)	Partial (with RT modules)	No	NI DAQ, FPGA, sensor interfacing	Instrumentation, data acquisition, embedded systems
ROS-Neuro	C++ / Python (ROS)	Yes	Yes	EEG, EMG via ROS nodes	Neurorobotics, brain–computer interfaces
BioPatRec	MATLAB	Partial (limited real-time)	Yes (MATLAB required)	EMG only	Prosthetic control research
LibEMG	Python	Yes	Yes	EMG (UDP-based interface)	Myoelectric control, HCI applications
BioSPPy	Python	No (offline only)	Yes	Offline files only (no device streaming)	Signal analysis, feature extraction
MOSAIC	C#, .NET + YAML	Yes	Yes	EMG, IMU, BLE, ultrasound, UDP, Serial	Assistive robotics, prosthetics, rehabilitation

usage, but a significant difference in memory footprint. Simulink utilized 8.9 % ± 0.7 % of CPU resources, while *MOSAIC* used 11.1 % ± 1.2 % (95th percentile: 10.1 % vs. 13.4 %). In terms of RAM, the working set for Simulink was 2.43 GiB, compared to 0.300 GiB for *MOSAIC* (95th percentile: 2.48 GiB vs. 0.31 GiB), representing approximately an 8.1 times higher memory usage for Simulink. Both systems functioned stably for 250 s at a rate of 200 Hz, with the first 25 s excluded to account for start-up. The metrics reflect CPU usage as a percentage of total machine capacity and RAM indicated in GiB.

Several limitations remain. The current release targets Windows on.NET Framework 4.8, which limits portability. The block catalog is smaller than in commercial suites; domain-specific functions may require custom C# blocks. Built-in learning currently focuses on classical models (e.g., ridge regression, random Fourier features), with advanced models integrated through the Python bridge (`PythonManager`). Some device connectors rely on vendor SDKs whose redistribution terms may constrain out-of-the-box availability.

Taken together with the related-work review, our aim is complementary rather than competitive with established robotics and signal-processing ecosystems. Simulink, LabVIEW, ROS 2 with `ros_control`, Orocos, and Gazebo address broad classes of modeling, verification, and deployment problems. The niche addressed here is earlier in the pipeline: rapid prototyping of human-in-the-loop biosignal control with live, per-block timing diagnostics, soft real-time streaming and scheduling on general-purpose operating systems, multi-modal input/output (including feedback), and human-readable, versionable configuration. The empirical results indicate that these requirements can be met with predictable timing and modest CPU and memory usage, while remaining interoperable with Python and ROS where appropriate.

Overall, *MOSAIC* is intended to bridge high-level prototyping environments and low-level real-time frameworks by combining open-source licensing, modular blocks, soft real-time execution, and declarative configuration. In this study, it supported rapid prototyping, transparent timing telemetry (on-rate, lagging, stumbling, per-edge rates), incremental learning, and multi-modal I/O within a unified suite, with optional integration to ROS and Python-based machine learning via *PythonNetManager*. Planned cross-platform runtime and UI enhancements are outlined in the Future Work section.

VII. FUTURE WORK

We plan to migrate the codebase to modern.NET 9.0 with an Avalonia [71] based user interface. This will enable native, cross-platform desktop builds on Windows, Linux, and macOS, while preserving the current YAML-first workflow and live instrumentation. The Avalonia transition is intended to improve UI flexibility for real-time visualization (e.g., multi-panel dashboards, high-DPI scaling, multi-monitor layouts) and to simplify interaction with control panels during experiments. The feasibility of mobile targets (iOS/Android) will be evaluated using Avalonia's toolchain, where practical.

Beyond desktop deployments, we aim to support portable "edge" scenarios on lightweight devices (e.g., small form-factor PCs, tablets) to facilitate data collection and closed-loop experiments outside laboratory settings. This includes attention to packaging (self-contained builds), device access (e.g., BLE/Serial), offline operation, and power/thermal considerations relevant to field studies in rehabilitation, assistive technology, and mobile health.

To lower the entry barrier, a graph-based, drag-and-drop pipeline editor is planned to complement YAML configuration. The editor will round-trip to canonical YAML so that visual and text-based specifications remain interchangeable and version-controllable. Schema validation, templating, and example galleries are planned to streamline common acquisition/processing/control patterns.

Finally, we will extend Python and ROS integration beyond the current examples (see Subsection III-J). For Python, this includes reference pipelines for common model classes, environment management (dependency pinning/lockfiles), and clearer lifecycle controls for long-running models.

VIII. CONCLUSION

We presented *MOSAIC*, a YAML-first, block-based runtime for biosignal-driven human–machine interaction that unifies acquisition, processing, learning, and control with live per-block diagnostics (*BlockTable* and control panels). Across three online pipelines, *MOSAIC* exhibited predictable soft real-time behavior with low timing jitter, high on-time proportions, and modest CPU usage, while integrating Python-based models without disrupting acquisition. A record–replay benchmark against Simulink showed a small difference in steady-state CPU usage but a substantially lower memory footprint for *MOSAIC* at 200 Hz (cf. Fig. 7), under identical timing constraints.

Current limitations include a Windows/.NET Framework 4.8 dependency and a smaller block catalog than commercial suites. Planned work targets migration to modern.NET 9.0 with an Avalonia front end for cross-platform builds, a graph-based editor that round-trips to YAML, and extended Python/ROS integration with reference pipelines for advanced temporal and deep models.

Overall, *MOSAIC* aims to bridge high-level prototyping environments and low-level real-time frameworks by combining open configuration, modular blocks, soft real-time execution, and transparent runtime telemetry. In this study it supported rapid, reproducible experimentation for assistive robotics, rehabilitation, and human–machine interface research, and provides a foundation for portable, multimodal, and human-in-the-loop control systems.

ACKNOWLEDGMENT

The authors would like to acknowledge Donato Brusamento, who previously worked at DLR, for laying the groundwork for the graphical representation of the block pipeline. They also wish to recognize Dr. Andrea Gigli, also a former DLR employee, for his contributions to the iNMF algorithm. Special thanks go out to all their students who worked with their software, helping to improve and extend its functionality.

References

[1] J. Andrysek, "Lower-limb prosthetic technologies in the developing world: A review of literature from 1994–2010," *Prosthetics Orthotics Int.*, vol. 34, no. 4, pp. 378–398, 2010, doi: 10.3109/03093646.2010.520060.

[2] C. L. McDonald, S. Westcott-McCoy, M. R. Weaver, J. Haagsma, and D. Kartin, "Global prevalence of traumatic non-fatal limb amputation," *Prosthetics Orthotics Int.*, vol. 45, no. 2, pp. 105–114, 2021, doi: 10.1177/0309364620972258.

[3] B. Yuan, D. Hu, S. Gu, S. Xiao, and F. Song, "The global burden of traumatic amputation in 204 countries and territories," *Frontiers Public Health*, vol. 11, Oct. 2023.

[4] UNICEF. (2022). *Global Report on Assistive Technology*. Accessed: May 25, 2025. [Online]. Available: https://www.who.int/publications/i/item/9789240049451

[5] J. Danemayer, D. Boggs, E. M. Smith, V. D. Ramos, L. R. Battistella, C. Holloway, and S. Polack, "Measuring assistive technology supply and demand: A scoping review," *Assistive Technol.*, vol. 33, no. sup1, pp. S35–S49, Dec. 2021.

[6] G. A. Noury, A. Walmsley, R. B. Jones, and S. E. Gaudl, "The barriers of the assistive robotics market—What inhibits health innovation?" *Sensors*, vol. 21, no. 9, p. 3111, Apr. 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/9/3111

[7] C. McGinn, M. F. Cullinan, M. Culleton, and K. Kelly, "A human-oriented framework for developing assistive service robots," *Disab. Rehabil., Assistive Technol.*, vol. 13, no. 3, pp. 293–304, Apr. 2018.

[8] A. de Souza Leo Rodrigues, L. B. A. Martinez, and Z. C. Silveira, "An iterative design procedure for the development of assistive devices based on a participatory approach," *J. Brazilian Soc. Mech. Sci. Eng.*, vol. 46, no. 3, pp. 1–21, Mar. 2024.

[9] G. Tao, G. Charm, K. Kabacińska, W. C. Miller, and J. M. Robillard, "Evaluation tools for assistive technologies: A scoping review," *Arch. Phys. Med. Rehabil.*, vol. 101, no. 6, pp. 1025–1040, Jun. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000399932 0300836

[10] M. Simão, N. Mendes, O. Gibaru, and P. Neto, "A review on electromyography decoding and pattern recognition for human-machine interaction," *IEEE Access*, vol. 7, pp. 39564–39582, 2019.

[11] N. Ahmad, R. A. R. Ghazilla, N. M. Khairi, and V. Kasi, "Reviews on various inertial measurement unit (IMU) sensor applications," *Int. J. Signal Process. Syst.*, vol. 1, pp. 256–262, Jan. 2013.

[12] Z. G. Xiao and C. Menon, "A review of force myography research and development," *Sensors*, vol. 19, no. 20, p. 4557, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/20/4557

[13] X. Yang, C. Castellini, D. Farina, and H. Liu, "Ultrasound as a neurorobotic interface: A review," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 54, no. 6, pp. 3534–3546, Jun. 2024.

[14] H. Hayashi and T. Tsuji, "Human–machine interfaces based on bioelectric signals: A narrative review with a novel system proposal," *IEEJ Trans. Electr. Electron. Eng.*, vol. 17, no. 11, pp. 1536–1544, Nov. 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/tee.23646

[15] H. Zhou and G. Alici, "Non-invasive human-machine interface (HMI) systems with hybrid on-body sensors for controlling upper-limb prosthesis: A review," *IEEE Sensors J.*, vol. 22, no. 11, pp. 10292–10307, Jun. 2022.

[16] D. Esposito, J. Centracchio, E. Andreozzi, G. D. Gargiulo, G. R. Naik, and P. Bifulco, "Biosignal-based human–machine interfaces for assistance and rehabilitation: A survey," *Sensors*, vol. 21, no. 20, p. 6863, Oct. 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/20/6863

[17] S. Kumar, S. Datta, V. Singh, D. Datta, S. Kumar Singh, and R. Sharma, "Applications, challenges, and future directions of human-in-the-loop learning," *IEEE Access*, vol. 12, pp. 75735–75760, 2024.

[18] T. Bao, S. Q. Xie, P. Yang, P. Zhou, and Z.-Q. Zhang, "Toward robust, adaptiveand reliable upper-limb motion estimation using machine learning and deep learning—A survey in myoelectric control," *IEEE J. Biomed. Health Informat.*, vol. 26, no. 8, pp. 3822–3835, Aug. 2022.

[19] E. J. Rechy-Ramirez and H. Hu, "Bio-signal based control in assistive robots: A survey," *Digit. Commun. Netw.*, vol. 1, no. 2, pp. 85–101, Apr. 2015.

[20] A. M. Simon, K. L. Turner, L. A. Miller, L. J. Hargrove, and T. Kuiken, "Pattern recognition and direct control home use of a multi-articulating hand prosthesis," in *Proc. IEEE 16th Int. Conf. Rehabil. Robot. (ICORR)*, Mar. 2019, pp. 386–391.

[21] *.NET Framework Version 4.8*, Microsoft Corporation, Redmond, CA, USA, 2019.

[22] (2020). *Simulation and Model-Based Design*. [Online]. Available: https://www.mathworks.com/products/simulink.html

[23] National Instrum. (2023). *LabVIEW System Design Software*. Accessed: May 26, 2025. [Online]. Available: https://www.ni.com/en-us/shop/labview.html

[24] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Sci. Robot.*, vol. 7, no. 66, p. 6074, May 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

[25] S. Macenski, A. Soragna, M. Carroll, and Z. Ge, "Impact of ROS 2 node composition in robotic systems," *IEEE Robot. Autom. Lett.*, vol. 8, no. 7, pp. 3996–4003, Jul. 2023.

[26] L. Tonin, G. Beraldo, S. Tortora, L. Tagliapietra, J. D. R. Millán, and E. Menegatti, "ROS-neuro: A common middleware for BMI and robotics. The acquisition and recorder packages," in *Proc. IEEE Int. Conf. Syst., Man Cybern. (SMC)*, Oct. 2019, pp. 2767–2772.

[27] G. Beraldo, S. Tortora, E. Menegatti, and L. Tonin, "ROS-neuro: Implementation of a closed-loop BMI based on motor imagery," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2020, pp. 2031–2037.

[28] L. Tonin, G. Beraldo, S. Tortora, and E. Menegatti, "ROS-neuro: An open-source platform for neurorobotics," *Frontiers Neurorobotics*, vol. 16, May 2022, Art. no. 886050, doi: 10.3389/fnbot.2022.886050.

[29] M. Ortiz-Catalan, R. Brånemark, and B. Håkansson, "BioPatRec: A modular research platform for the control of artificial limbs based on pattern recognition algorithms," *Source Code Biol. Med.*, vol. 8, no. 1, p. 11, Dec. 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:14710785

[30] E. Eddy, E. Campbell, A. Phinyomark, S. Bateman, and E. Scheme, "LibEMG: An open source library to facilitate the exploration of myoelectric control," *IEEE Access*, vol. 11, pp. 87380–87397, 2023.

[31] P. Bota, R. Silva, C. Carreiras, A. Fred, and H. P. da Silva, "BioSPPy: A Python toolbox for physiological signal processing," *SoftwareX*, vol. 26, May 2024, Art. no. 101712. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711024000839

[32] M. Nabeel, K. Aqeel, M. N. Ashraf, M. I. Awan, and M. Khurram, "Vibrotactile stimulation for 3D printed prosthetic hand," in *Proc. 2nd Int. Conf. Robot. Artif. Intell. (ICRAI)*, Nov. 2016, pp. 202–207.

[33] J. Tchimino, R. L. Hansen, P. H. Jørgensen, J. Dideriksen, and S. Dosen, "Application of EMG feedback for hand prosthesis control in high-level amputation: A case study," *Sci. Rep.*, vol. 14, no. 1, Dec. 2024, Art. no. 31676.

[34] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, vol. 3, Sendai, Japan, Sep. 2004, pp. 2149–2154.

[35] C. E. Agüero, N. Koenig, I. Chen, H. Boyer, S. Peters, J. Hsu, B. Gerkey, S. Paepcke, J. L. Rivero, J. Manzo, E. Krotkov, and G. Pratt, "Inside the virtual robotics challenge: Simulating real-time robotic disaster response," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 494–506, Apr. 2015.

[36] H. Bruyninckx, "Open robot control software: The OROCOS project," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 3, Mar. 2001, pp. 2523–2528.

[37] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. R. Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke, and E. F. Perdomo, "Ros_control: A generic and simple control framework for ROS," *J. Open Source Softw.*, vol. 2, no. 20, p. 456, 2017.

[38] A. Kiselew. (2024). *GraphX: Graph Layout and Visualization Library for .NET*. [Online]. Available: https://www.nuget.org/packages/GraphX

[39] (2024). *QuickGraph: Directed Graph Data Structures and Algorithms for .NET*. [Online]. Available: https://www.nuget.org/packages/QuickGraph

[40] (2025). *Thalmic Labs*. Accessed: Feb. 1, 2025. [Online]. Available: https://github.com/thalmiclabs

[41] OT Bioelettronica. (2024). *MuoviPro: Wireless High-Density EMG Acquisition System*. Accessed: May 25, 2025. [Online]. Available: https://otbioelettronica.it/en/muovi/

[42] (2024). *Delsys EMG and Sensor Systems*. Accessed: May 25, 2025. [Online]. Available: https://delsyseurope.com/

[43] M. Connan, E. R, Ramírez, B. Vodermayer, and C. Castellini, "Assessment of a wearable force- and electromyography device and comparison of the related signals for myocontrol," *Frontiers Neurorobotics*, vol. 10, p. 17, Nov. 2016.

[44] R. Koiva, B. Hilsenbeck, and C. Castellini, "FFLS: An accurate linear device for measuring synergistic finger contractions," in *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, Aug. 2012, pp. 531–534.

[45] M. Sierotowicz, D. Brusamento, B. Schirrmeister, M. Connan, J. Bornmann, J. Gonzalez-Vargas, and C. Castellini, "Unobtrusive, natural support control of an adaptive industrial exoskeleton using force myography," *Frontiers Robot. AI*, vol. 9, Sep. 2022, Art. no. 919370.

[46] M. Sierotowicz, C. Castellini, and K. Anam, "Force myography shows higher correlation to force output compared to average rectified value electromyography features," in *Proc. Int. Conf. NeuroRehabilitation*. Cham, Switzerland: Springer, 2024, pp. 627–631.

[47] M. Fournelle, T. Grün, D. Speicher, S. Weber, M. Yilmaz, D. Schoeb, A. Miernik, G. Reis, S. Tretbar, and H. Hewener, "Portable ultrasound research system for use in automated bladder monitoring with machine-learning-based segmentation," *Sensors*, vol. 21, no. 19, p. 6481, Sep. 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/19/6481

[48] M. Sierotowicz, M. Connan, and C. Castellini, "Human-in-the-loop assessment of an ultralight, low-cost body posture tracking device," *Sensors*, vol. 20, no. 3, p. 890, Feb. 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/3/890

[49] M. Sierotowicz, M.-A. Scheidl, and C. Castellini, "Adaptive filter for biosignal-driven force controls preserves predictive powers of sEMG," in *Proc. Int. Conf. Rehabil. Robot. (ICORR)*, Sep. 2023, pp. 1–6.

[50] A. Gijsberts, R. Bohra, D. Sierra González, A. Werner, M. Nowak, B. Caputo, M. A. Roa, and C. Castellini, "Stable myoelectric control of a hand prosthesis using non-linear incremental learning," *Frontiers Neurorobotics*, vol. 8, p. 8, Mar. 2014. [Online]. Available: https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2014.00008

[51] F. Egle and C. Bäker, "Airob-lab/sim—Streamlined-input-manager-UDP: V0.1.2-beta," FAU, Erlangen, Germany, Tech. Rep., Jun. 2025. Accessed: Jul. 21, 2025. [Online]. Available: http://dx.doi.org/10.5281/zenodo.15707420

[52] M. Laffranchi, N. Boccardo, S. Traverso, L. Lombardi, M. Canepa, A. Lince, M. Semprini, J. A. Saglia, A. Naceri, R. Sacchetti, E. Gruppioni, and L. De Michieli, "The hannes hand prosthesis replicates the key biological properties of the human hand," *Sci. Robot.*, vol. 5, no. 46, Sep. 2020, Art. no. eabb0467.

[53] A. M. Simon, L. J. Hargrove, B. A. Lock, and T. A. Kuiken, "Target achievement control test: Evaluating real-time myoelectric pattern-recognition control of multifunctional upper-limb prostheses," *J. Rehabil. Res. Develop.*, vol. 48, no. 6, p. 619, 2011.

[54] M. Zardoshti-Kermani, B. C. Wheeler, K. Badie, and R. M. Hashemi, "EMG feature evaluation for movement control of upper extremity prostheses," *IEEE Trans. Rehabil. Eng.*, vol. 3, no. 4, pp. 324–333, Apr. 1995.

[55] A. Phinyomark, P. Phukpattaranont, and C. Limsakul, "Feature reduction and selection for EMG signal classification," *Expert Syst. Appl.*, vol. 39, no. 8, pp. 7420–7431, Jun. 2012.

[56] *Pythonnet/Pythonnet*. Accessed: Nov. 1, 2025. [Online]. Available: https://github.com/pythonnet/pythonnet

[57] S. Eustace. *Poetry: Python Packaging and Dependency Management Made Easy*. Accessed: Nov. 1, 2025. [Online]. Available: https://github.com/python-poetry/poetry

[58] P. Virtanen et al., "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, 2020.

[59] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. J. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Apr. 2012.

[60] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 8026–8037. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[61] A. Gigli, A. Gijsberts, and C. Castellini, "Unsupervised myocontrol of a virtual hand based on a coadaptive abstract motor mapping," in *Proc. Int. Conf. Rehabil. Robot. (ICORR)*, Jul. 2022, pp. 1–6.

[62] A. Schulz, F. Egle, M. Osswald, A. D. Vecchio, and C. Castellini, "Towards unsupervised incremental and proportional myocontrol based on higher-density surface electromyography," in *Proc. Int. Conf. Rehabil. Robot. (ICORR)*, May 2025, pp. 1106–1111. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/11063080

[63] F. Egle, D. Di Domenico, A. Marinelli, N. Boccardo, M. Canepa, M. Laffranchi, L. De Michieli, and C. Castellini, "Preliminary assessment of two simultaneous and proportional myocontrol methods for 3-DoFs prostheses using incremental learning," in *Proc. Int. Conf. Rehabil. Robot. (ICORR)*, Sep. 2023, pp. 1–6.

[64] J. Ouyang, F. Egle, C. Igney, T. Mutzke, C. Dahmani, C. Castellini, and S. Thuerauf, "C-arm unleashed: Intuitive inter-operative positioning of C-arms using wearable gesture detection," in *Proc. 10th IEEE RAS/EMBS Int. Conf. Biomed. Robot. Biomechatronics (BioRob)*, Sep. 2024, pp. 1183–1189.

[65] M.-A. Scheidl, İ. B. Akarsu, F. Mehrkens, B. Czierlinski, and C. Castellini, "An impedance-controlled knee orthosis for assisted sit-to-stand," in *Converging Clinical and Engineering Research on Neurorehabilitation V*. Cham, Switzerland: Springer, 2025, pp. 279–283.

[66] S. Thuerauf, F. Mehrkens, C. Castellini, and M. Sierotowicz, "Back to the Cartesian: Pilot study for assessing human stiffness in 3D Cartesian space by transforming from muscle space in a peg-in-hole scenario for tele-impedance," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2025, pp. 5906–5913.

[67] M. Sierotowicz and C. Castellini, "Robot-inspired human impedance control through functional electrical stimulation," in *Proc. Int. Conf. Rehabil. Robot. (ICORR)*, Sep. 2023, pp. 1–6.
[68] M. Sierotowicz and C. Castellini, "Omnidirectional endpoint force control through functional electrical stimulation," *Biomed. Phys. Eng. Exp.*, vol. 9, no. 6, Nov. 2023, Art. no. 065008.
[69] D. S. Oliveira, M. Ponfick, D. I. Braun, M. Osswald, M. Sierotowicz, S. Chatterjee, D. Weber, B. Eskofier, C. Castellini, D. Farina, T. M. Kinfe, and A. Del Vecchio, "A direct spinal cord-computer interface enables the control of the paralysed hand in spinal cord injury," *Brain*, vol. 147, no. 10, pp. 3583–3595, Oct. 2024.
[70] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, *arXiv:1512.03385*.
[71] Avalonia UI Team. (2025). *Avalonia UI*. Accessed: Oct. 17, 2025. [Online]. Available: https://github.com/AvaloniaUI/Avalonia

HANNAH BRAUN (Student Member, IEEE) received the bachelor's and master's degrees in biomedical engineering from the Technical University of Applied Sciences Mannheim, Germany. Since November 2023, she has been a Researcher with the Assistive Intelligent Robotics Laboratory, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany. Her current research interests include intent detection for upper limb prostheses and the development of VR environments with virtual prostheses.

MAREK SIEROTOWICZ (Member, IEEE) received the bachelor's degree in electronic engineering from the Technical University of Vienna, in 2016, the master's degree in robotics from the Technical University of Munich, in 2019, and the Ph.D. degree from the Assistive Intelligent Robotics Laboratory, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany, in 2024. During his tenure at the Institute of Robotics and Mechatronics, German Aerospace Center, Weßling, Germany, his research focuses on human–machine interfaces and robotic applications, with a particular emphasis on human-in-the-loop systems. His work aims to advance the feasibility, effectiveness, and practicality of crewed space missions. His current research interests include novel actuation and sensor modalities for providing force feedback in teleoperation and virtual reality applications, as well as rehabilitation and assistive technologies for individuals with motor impairments.

FABIO EGLE (Graduate Student Member, IEEE) received the master's degree in medical engineering from Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, where he is currently pursuing the Ph.D. degree with the Assistive Intelligent Robotics (AIROB) Laboratory. His research interests include biosignal analysis, machine learning methods, and virtual reality in assistive robotics. Specifically, his work explores intent detection and assessment for controlling upper limb prosthetics in virtual environments.

MARC-ANTON SCHEIDL (Graduate Student Member, IEEE) received the B.Sc. degree in medical engineering and the M.Sc. degree in medical engineering with a focus on medical device technology, manufacturing engineering, and prosthetics from Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, in 2018 and 2022, respectively. Since August 2022, he has been a Doctoral Researcher with the Assistive Intelligent Robotics Laboratory, Friedrich-Alexander-Universität Erlangen-Nürnberg. His research interests include intention-driven control architectures and hardware for lower-limb exoskeletons and active orthoses. He is aiming to restore and augment mobility in individuals with neuromusculoskeletal impairments.

SILVANA MIRANDA MONTENEGRO (Student Member, IEEE) received the bachelor's degree in medical engineering from the Universities of Stuttgart and Tübingen, Germany, and the master's degree in medical engineering from the Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, where she is currently pursuing the Ph.D. degree with the Assistive Intelligent Robotics Laboratory, where her research focuses on signal processing for rehabilitation robotics control.

SABINE THUERAUF (Member, IEEE) received the bachelor's and master's degrees from the Technical University of Munich, Germany, in 2012 and 2014, respectively, and the Dr.rer.nat. degree from the Faculty of Informatics, Technical University of Munich, in 2018. Following her studies, she joined the Research Institute Fortiss, Munich, Germany, where she worked on the "RoboterRöntgen" project, focusing on the spatial calibration of a robotic C-arm system in collaboration with Siemens Healthcare. After several years in industry, she joined the Assistive Intelligent Robotics Laboratory, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany, as a Postdoctoral Researcher in September 2022 and is now focusing on human impedance measurements using biosignals and intuitive immersive robot control using biosignals. She is currently a Researcher specializing in robotics, with a primary focus on medical applications in recent years.

CLAUDIO CASTELLINI (Member, IEEE) is currently a Researcher of medical robotics, focusing on rehabilitation and assistive robotics, human–machine interfaces and interaction, and applied machine learning. He has been with German Aerospace Center, Weßling, Germany, since 2009, where he is currently a Senior Researcher and the joint Lab Leader of the Institute of Robotics and Mechatronics. In 2021, he was appointed as a Full Professor of medical robotics with Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, where he leads the Assistive Intelligent Robotics Laboratory. He has co-authored approximately 160 scientific articles. He is involved in several research projects at Bavarian, German, and European levels and serves as an Associate Editor for IEEE TRANSACTIONS ON NEURAL SYSTEMS AND REHABILITATION ENGINEERING, a Board Member for the International Consortium of Rehabilitation Robotics (ICORR), the organizer of the ICORR YouTube Channel, and an Editor and an Associate Editor for BioRob and RehabWeek.