École polytechnique fédérale de Lausanne
Rehabilitation and Assistive Robotics Laboratory

RESEARCH REPORT

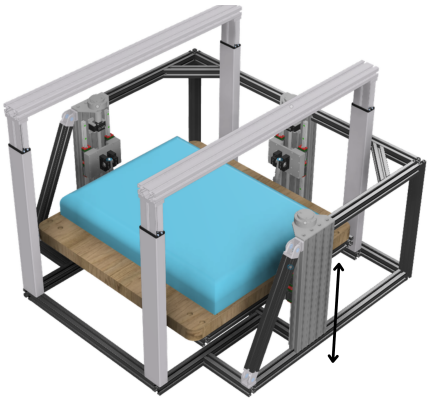
Development of a Control Architecture for a 3-DOF Robotic Platform for Standing Rehabilitation

Leith Kochbati

Laboratory REHAssist, EPFL

Supervisors Imane Ben M'Rad
Dr. Mohamed Bouri

Program Laidlaw Scholars Program



REHAssist

LAIDLAW SCHOLARS

Abstract

Spinal cord injury (SCI) is a debilitating neurological condition affecting an estimated 15–20 million people worldwide and severely impacting mobility, independence, and quality of life.

This research revolves around one fascinating question: what if engaging in upper-body activities while standing could be made possible again for paraplegic patients?

In recent years, epidural electrical stimulation (EES) has emerged as a promising approach for restoring locomotion. In contrast, stable standing has been less extensively studied.

Therefore, this research aims to design a 3-degrees-of-freedom robotic experimental platform that provides a safe and controlled environment for standing training and patient performance assessment. Through electrical stimulation of the spinal cord, patients will regain the ability to stand and will be trained to maintain their balance on a platform that can be tilted and elevated. Their reaching abilities will additionally be trained and assessed through tasks performed in a virtual reality environment.

The first iteration was focused on developing a user interface and control architecture capable of communicating with the platform's motors, acquiring their real-time positions, and coordinating their movements. The communication between the control software and the robotic platform, as well as the mechanisms required to start, monitor, and safely stop the system have been successfully implemented. This resulted in a software providing controlled and reliable operation of the motors of the robotic platform.

Future work will involve simulation and the integration of virtual reality and additional sensors, including foot and hand force sensors and hand tracking, to enable quantitative assessment of patient performance.

Abbreviations

SCI	Spinal cord injury
EES	Epidural electrical stimulation
DOF	Degree of freedom
GUI	Graphical user interface
API	Application programming interface
CAN	Controller Area Network
MIT mode	Compact position/velocity/gain/torque command format used by the motor drives
VR	Virtual reality
IK	Inverse kinematics

Contents

Abstract i

Abbreviations i

1 Introduction and scientific motivation 1
- 1.1 Spinal cord injury and the challenge of standing . . . 1
- 1.2 Epidural electrical stimulation as an enabling technology . . . 1
- 1.3 Research question . . . 1

2 Project objectives and scope 2

3 Mechanical model of the robotic platform and intended experimental use 2

4 Hardware configuration and physical connections 4
- 4.1 Motor and drive units . . . 4
- 4.2 CAN bus and addressing . . . 4
- 4.3 Hardware connections overview . . . 5

5 Software architecture: motor-control API 5
- 5.1 API design and static-library integration . . . 6
- 5.2 Graphic User Interface . . . 8
- 5.3 Threading and movement lifecycle . . . 8
- 5.4 Geometric modelling and inverse kinematics . . . 9

6 Results of the first development iteration 10

7 Discussion and limitations 10
- 7.1 **Experimental limitations.** . . . 11
- 7.2 **Integration and clinical requirements.** . . . 11

8 Future work 11
- 8.1 **Mechanical assembly and validation** . . . 11
- 8.2 **Pressure sensing and patient assessment.** . . . 12
- 8.3 **Virtual reality integration.** . . . 12

9 Conclusion 12

Acknowledgements 13

References 14

1 Introduction and scientific motivation

1.1 Spinal cord injury and the challenge of standing

Spinal cord injury (SCI) disrupts neural pathways responsible for motor and sensory functions, potentially resulting in paralysis below the lesion. More than 15 million people worldwide live with SCI [1].

For individuals with paraplegia, standing requires coordinated muscle activation, postural stabilization, and continuous adaptation to sensory feedback. Restoring this function therefore represents a significant rehabilitation challenge.

1.2 Epidural electrical stimulation as an enabling technology

Epidural electrical stimulation (EES) delivers electrical pulses to spinal neural structures through implanted electrodes, modulating the excitability of sensorimotor networks.

Harkema et al. demonstrated that EES combined with rehabilitation enabled standing and assisted stepping in an individual with motor-complete paraplegia [2]. Subsequent studies confirmed that standing performance depends on stimulation parameters, sensory feedback, and task-specific training [4, 5].

These findings motivate the development of experimental environments capable of reproducing controlled standing conditions and quantitatively evaluating postural responses.

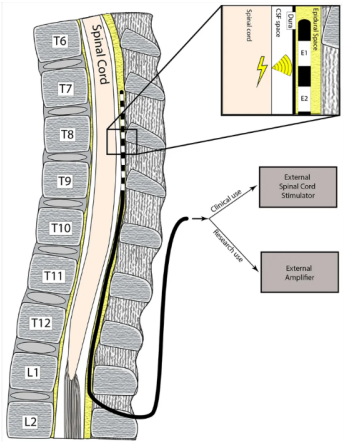


Figure 1: Schematic representation of epidural electrode implantation in the human spinal cord. Source: Burke et al. (2021) [3],CC BY 4.0.

1.3 Research question

In recent years, epidural electrical stimulation (EES) has emerged as a promising approach for restoring locomotion. In contrast, stable standing has been less extensively studied. Therefore, this research aims to design a 3-degrees-of-freedom robotic experimental platform that provides a safe and controlled environment for standing training and patient performance assessment. Through electrical stimulation of the spinal cord, patients will regain the ability to stand and will be trained to maintain their balance on a platform that can be tilted and elevated.

Their reaching abilities will additionally be trained and assessed through tasks performed in a virtual reality environment.

The central engineering question is:

> *How can a robotic platform provide safe, reproducible, and software-controlled standing conditions for rehabilitation experiments involving EES-enabled participants?*

This work focuses on developing the motor-control software architecture required to command and coordinate the three actuators and generate controlled movements, and acquire position feedback. It establishes the foundation for future integration of platform-level control, sensing, and virtual-reality rehabilitation tasks.

2 Project objectives and scope

The objective of this first development iteration was to establish the software foundation required to operate the three-actuator robotic platform.

The development focused on four main objectives:

- Establish communication with the motor drives, allowing position commands and feedback acquisition.
- Generate smooth, parabolic trajectories and coordinate the movements of the three actuators.
- Develop an operator interface (GUI) enabling motor configuration, movement execution, and start/stop control.
- Develop a first inverse kinematics (IK) model relating the desired platform orientation and elevation to the required actuator positions.

The motor-control functionalities were implemented through a C++ API integrated with a Qt graphical interface. The inverse kinematics model was developed separately and subsequently tested successfully after the completion of this project. Its integration into the motor-control architecture remains future work.

The current control interface therefore operates in motor coordinates rather than directly commanding platform inclination and elevation.

3 Mechanical model of the robotic platform and intended experimental use

The robotic platform, shown in Figure 2, consists of a movable standing surface supported by three vertically actuated mechanisms, mounted on a rigid aluminium frame. Each mechanism comprises a motor-driven lead screw and a guided carriage. Motor rotation is converted

into linear displacement through the screw-nut mechanism, allowing each carriage to move independently along its vertical axis.

The standing surface is connected to these mechanisms through a combination of revolute joints (pivots) and spherical joints (ball joints). The pivots allow rotation about a defined axis, while the spherical joints accommodate angular motion in multiple directions. These connections allow the platform to change orientation as the three carriages move to different heights, without imposing incompatible rotational constraints on the structure.

By coordinating the vertical positions of the three carriages, the mechanism provides three degrees of freedom: two rotational motions (pitch and roll) and one vertical translation. Equal carriage displacements produce a vertical movement, whereas differential displacements generate platform inclination.

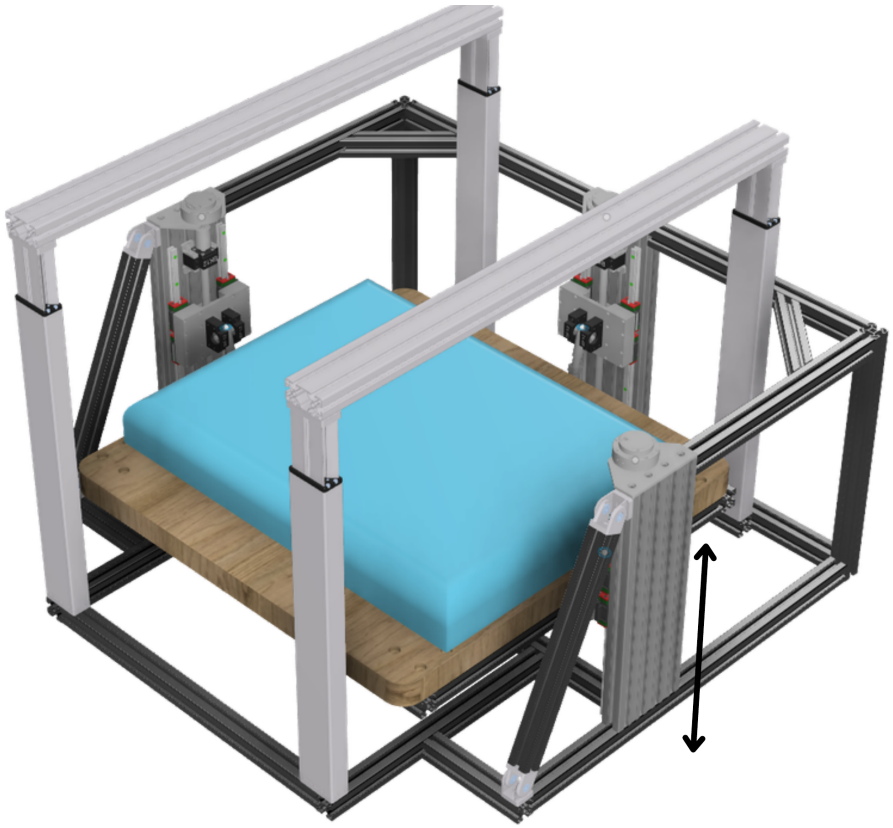


Figure 2: CAD rendering of the robotic platform used in the project.

Intended experimental use:

The experimental protocol is intended for patients with spinal cord injury who have regained the ability to stand with epidural electrical stimulation (EES). Initially, the platform is lowered to its minimum height, allowing the participant to step onto the standing surface.

Once the participant is safely positioned, the operator uses the graphical user interface (GUI) on the control computer to command a desired platform height and inclination. The software translates these commands into coordinated actuator movements, positioning the platform at the prescribed configuration.

The participant then performs standing balance exercises under EES, potentially combined with reaching tasks in a virtual-reality environment. Pressure sensors beneath the feet and

force sensors on the support bars are intended to provide quantitative measurements of weight distribution, support reliance, and upper-body movement.

These measurements will allow the operator to assess postural stability and collect data in order to assess the patient's performance as well as the EES' efficiency for the standing activity.

4 Hardware configuration and physical connections

In this section, an overview of the hardware used as well as the various connections and communication methods implemented is presented.

4.1 Motor and drive units

The platform is actuated by three SteadyWin GIM6010-6 brushless DC motors, each equipped with a GDZ468 driver. The GIM6010-6 family offers rated output torques ranging from 1.8 to 3.3 Nm depending on the configuration.

Each driver regulates its motor using position feedback and receives motion commands through the CAN interface.

The GDZ468 driver supports an operating voltage range of 12–40 V. The three motors are powered by an external DC power supply (24 V, 2 A maximum).

4.2 CAN bus and addressing

The three motor drivers communicate through a Controller Area Network (CAN) bus. CAN is a differential communication protocol allowing multiple devices to exchange messages over the same two wires, CAN High and CAN Low.

The three motors share a common bus, terminated by two 120 Ω resistors placed at its physical ends to reduce signal reflections.

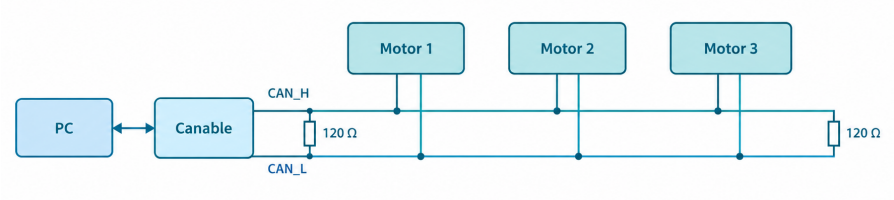


Figure 3: CAN bus topology

Each motor is assigned a unique CAN identifier to distinguish commands and responses: Motor 1: `[0x01]`, Motor 2: `[0x02]`, and Motor 3: `[0x03]`.

Motor addressing and communication parameters are configured using the manufacturer-provided ZE300 GUI software, available through SteadyWin's documentation and used with a USB-to-RS485 [6]

4.3 Hardware connections overview

The CAN network is connected to the control computer through a CANable 2.0 USB-to-CAN adapter, which acts as the interface between the computer's USB connection and the motor drivers' CAN bus.

The computer runs the C++/Qt control software developed during this project. Through the Linux SocketCAN interface (`can0`), the application generates motion trajectories and transmits motor commands over the CAN bus during execution. Motor position feedback is acquired through the same communication path.

The complete communication chain is therefore: PC (control software), USB, CANable, CAN bus, and the three motor drivers. Motor power is supplied independently through the external DC power supply.

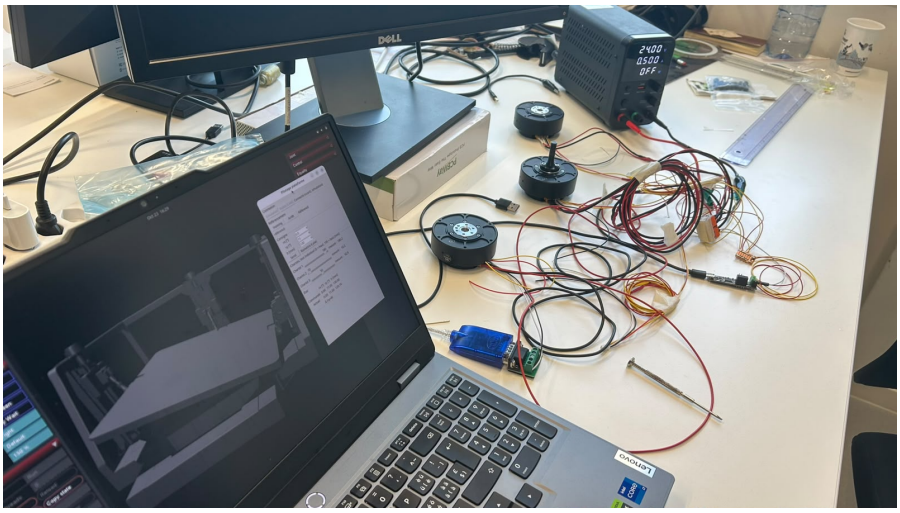


Figure 4: Experimental setup

5 Software architecture: motor-control API

The control software was developed in C++20, following an object-oriented programming (OOP) approach to separate motor control, trajectory generation, communication and user interaction into dedicated classes.

The architecture consists of two main components: a motor-control API implemented as a static C++ library, and a Qt graphical user interface (GUI) that uses this library to operate the platform.

The software targets Ubuntu Linux, using the native SocketCAN interface for communication with the motors. C++20 features, particularly `std::jthread`, are used to execute motor movements concurrently without blocking the graphical interface.

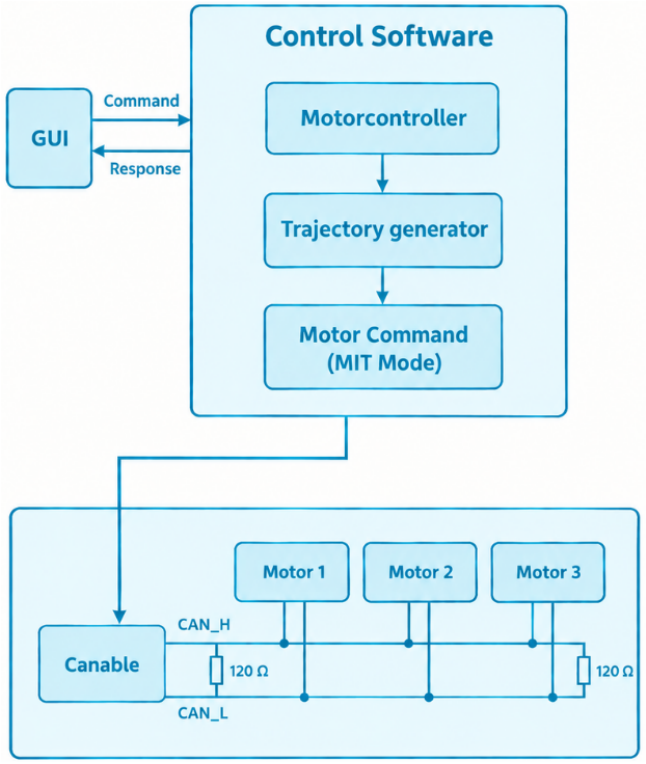


Figure 5: Illustration of the software architecture along with the hardware connections

5.1 API design and static-library integration

The motor-control API was implemented as a static C++ library written in C++20. Its role is to isolate all low-level control functions from the graphical interface and to provide a clear software structure for motor manipulation, trajectory generation, and coordinated execution. The API is organized around three main components: the `Motor` class, the trajectory-generation functions, and the `MotorController` class, which acts as the main entry point of the library.

5.1.1 Motor commands and CAN communication

The first part of the API is dedicated to direct motor communication. Under Ubuntu Linux, communication is based on the native SocketCAN interface, which allows CAN messages to be exchanged through raw CAN sockets. In practice, the control software opens a CAN socket, binds it to the `can0` interface, and exchanges CAN frames with the motor drives.

The motors are controlled using the MIT command format described in the manufacturer documentation [6]. In this format, each CAN frame contains compactly encoded command variables such as desired position, desired velocity, proportional gain, derivative gain, and feed-forward torque. Feedback frames return measured motor quantities, including position, velocity, and torque. This low-level protocol forms the basis of the motor control loop implemented in the software.

The corresponding implementation is encapsulated in a `Motor` class. Each motor instance stores the main attributes required for communication, including its CAN identifier and the SocketCAN file descriptor. The class also provides the functions required to handle motor operations according to the custom CAN protocol, such as entering and exiting MIT mode,

setting the zero position, sending command frames, and reading feedback values. In this way, each motor is represented as an object combining both its communication parameters and its associated control operations.

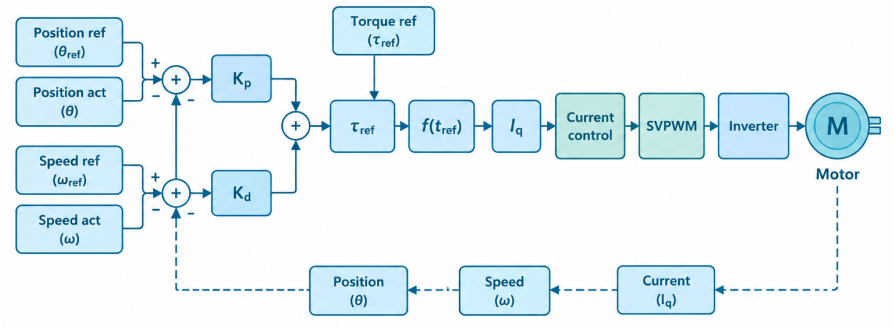


Figure 6: MIT Control loop

5.1.2 Trajectory generation

The second part of the API is the trajectory-generation module. For each motor, the GUI provides a desired final position together with motion parameters such as maximum velocity, acceleration, and deceleration. From these values, the trajectory-generation functions compute a parabolic position profile corresponding to a trapezoidal velocity law. This avoids abrupt position steps and produces a smoother and more controlled motor motion.

Because the three motors generally do not travel the same distance, their individual trajectories would naturally have different durations. The implemented trajectory-generation module therefore includes a synchronization step: after computing the three profiles, it adjusts the shorter ones so that all motors finish at the same time. Once this common duration is established, the generator can provide an intermediate desired position for each motor at every time step until the final destination is reached.

5.1.3 MotorController

The `MotorController` class is the main entry point of the API. It encapsulates the other components of the library and coordinates the complete motor-control process. Its role is to connect the high-level command received from the GUI to the low-level sequence of trajectory computation and CAN transmission.

In particular, the `start` function implements the main control loop. Starting from the current motor positions, it creates the required trajectories, synchronizes them, samples the intermediate position references as a function of time, and sends the corresponding commands to the three motor objects. The `MotorController` therefore coordinates motor manipulation, trajectory calculation, and command transmission in order to create a complete control loop from the initial motor positions to the desired final configuration.

5.2 Graphic User Interface

The graphical user interface (GUI) was developed using Qt Widgets to provide an intuitive way to operate and monitor the three motors. As a client of the static motor-control library, it acts as the front-end of the control system, relying on the motor-control API for all motor-related operations.

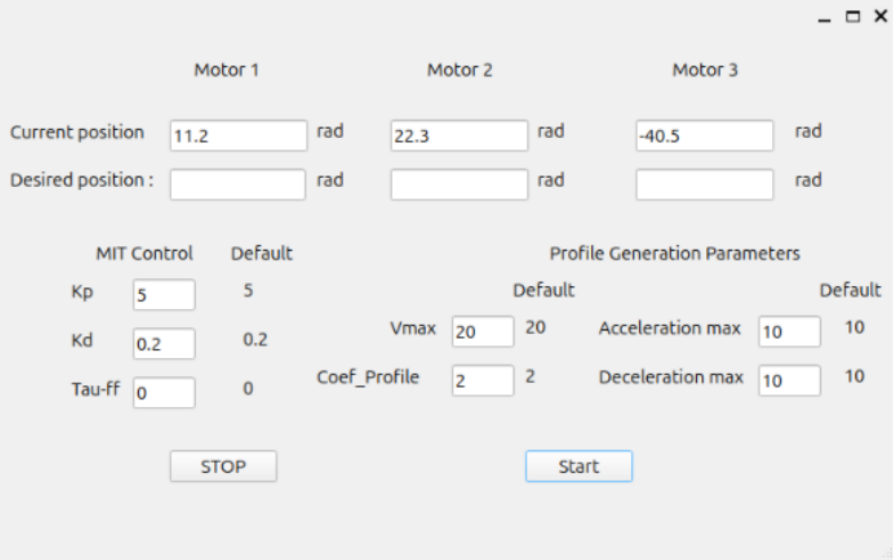


Figure 7: GUI illustration using the Qt tools [7]

As illustrated in Figure 7, the interface displays the current positions of the three motors and allows the operator to specify their desired positions. It also provides fields for configuring the MIT control parameters (proportional gain, derivative gain, and feed-forward torque) and the trajectory parameters (maximum velocity, acceleration, deceleration, and profile coefficient).

5.3 Threading and movement lifecycle

A central requirement was to keep the GUI responsive during motor movements. Since trajectory execution involves continuous command transmission over CAN, running the control loop directly in the Qt event loop would block user interaction.

To avoid this, the GUI uses `std::jthread`, a C++20 feature providing thread management and cooperative cancellation through `std::stop_token`.

When the Start button is pressed, the GUI collects the operator parameters and launches a dedicated worker thread executing the `start` function in `MotorController`. The Start button is disabled to prevent concurrent movements.

The controller computes the synchronized trajectories and enters the execution loop, continuously evaluating the intermediate position references and sending the corresponding commands to the motors.

During execution, the worker periodically checks its `stop_token`. If the movement reaches its final position, the loop terminates normally. Alternatively, pressing Stop requests cancellation through `request_stop()`, causing the loop to terminate once the stop request is detected.

In either case, the worker notifies the GUI through Qt's signal-slot mechanism. The interface can then refresh the motor positions and re-enable the Start button for subsequent commands.

This complete command lifecycle is illustrated in Figure 8.

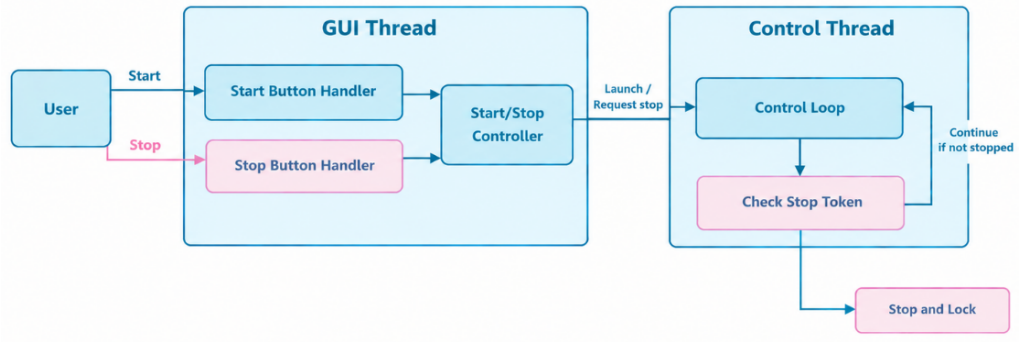


Figure 8: Illustation of the movement lifecycle

5.4 Geometric modelling and inverse kinematics

The inverse kinematics (IK) model was developed to determine the required actuator positions from a desired platform configuration, defined by two tilt angles (pitch and roll) and a vertical elevation.

The mechanical geometry is described using the plane of the standing surface, its attachment points, and the fixed lengths of the connecting links. For a given platform configuration, the model calculates the spatial positions of the attachment points and derives the corresponding vertical positions of the three carriages through geometric constraints.

The model was implemented in C++, independently of the motor-control API. A Python simulation was also developed to visualize the platform in three dimensions and interactively modify its height and inclination, as illustrated in Figure 9.

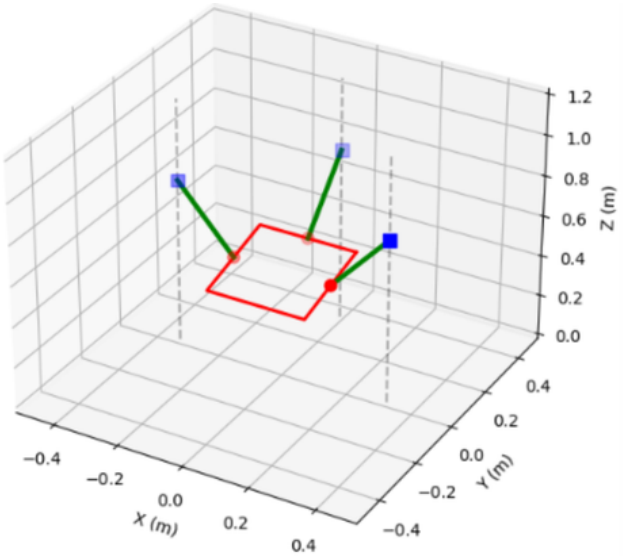


Figure 9: Geometric simulation of the robotic platform illustrating the relationship between platform orientation and actuator positions.

The model was subsequently tested successfully after the initial development period. However, its integration into the motor-control API remains future work, particularly the conversion of calculated carriage positions into calibrated motor references.

6 Results of the first development iteration

The main outcome of this project was the successful development of a C++ motor-control API, integrated with a Qt graphical user interface.

Communication with the three motor drives was established through CAN, allowing motor commands to be transmitted and position feedback to be acquired. The implemented trajectory generator enabled smooth, coordinated movements, while the GUI provided an operational interface for configuring motor parameters and executing movements without blocking user interaction.

The Start/Stop mechanism was also successfully implemented, allowing movements to be initiated, monitored and interrupted through cooperative thread cancellation.

In parallel, a first inverse kinematics model was developed and visualized through a Python simulation. The model was subsequently tested successfully after the completion of the initial development period.

Overall, the project established a functional software architecture for controlling the platform's three actuators.
Interpretation of "successful development": In this report, successful implementation means that the software mechanisms required to command, monitor, and stop the motor-control sequence are present and integrated. It does *not* mean that clinical safety or deterministic timing have already been demonstrated.

7 Discussion and limitations

The developed control architecture provides a functional foundation for the robotic platform. Its object-oriented design separates motor communication, trajectory generation, and user interaction into dedicated components, allowing each module to be independently modified or extended. Implementing the motor-control API as a static library also makes it possible to reuse the control functions without depending on the graphical interface.

The trajectory-generation module represents an important contribution to the control of the three-actuator mechanism. Rather than sending abrupt position commands, the software generates bounded trajectories and synchronizes their durations. This ensures that the three motors are commanded to reach their destinations simultaneously, reducing the risk of unintended intermediate platform configurations. Experimental tracking tests provided an initial assessment of the motors' ability to follow these references.

7.1 Experimental limitations.

All motor tests were conducted without mechanical loads. Consequently, the measured tracking performance cannot yet be considered representative of the fully assembled platform operating under patient loading. Mechanical assembly and subsequent testing under realistic conditions will be necessary to evaluate the influence of external loads, friction, and mechanical transmission on motor behaviour.

In particular, the MIT-mode control parameters, including the proportional and derivative gains, will need to be experimentally tuned to achieve stable and accurate motion under load. The current parameter settings should therefore be regarded as preliminary rather than optimized for the final application.

Another significant limitation concerns the execution time of the control loop. Individual iterations currently take excessively long, primarily due to delays associated with SocketCAN read and write operations. This restricts the effective command update frequency.

Future improvements should focus on optimizing CAN communication, particularly by reducing blocking operations, implementing appropriate receive timeouts, and improving the scheduling of motor commands. The control-loop period should then be measured to evaluate its stability and suitability for coordinated platform motion.

7.2 Integration and clinical requirements.

Finally, the current Stop functionality relies on cooperative thread cancellation and does not constitute a physical emergency-stop system. Before experiments involving patients, the platform will require independent hardware safety mechanisms, motion and load limits, communication-failure handling, and comprehensive safety validation of the complete electromechanical system.

The present results therefore demonstrate the feasibility of the proposed control architecture, while further mechanical integration, performance optimization, and safety validation are required before the system can be used in rehabilitation experiments.

8 Future work

Although the inverse kinematics model was subsequently tested successfully, it remains separate from the motor-control API. Its integration will enable the operator to command platform height and inclination directly rather than entering individual motor positions. This transition will also require geometric calibration and verification of mechanical workspace limits.

8.1 Mechanical assembly and validation

The next step is the mechanical assembly of the complete robotic platform and its integration with the developed control software. The motors will need to be tested under representative

loads, with appropriate tuning of the MIT-mode control parameters to ensure accurate and stable movements. The inverse kinematics model will also be integrated to allow direct control of platform height and inclination through the GUI.

The assembled system must then undergo extensive testing under realistic operating conditions, including mechanical loading, motion accuracy, and safety validation before any experiments involving patients.

8.2 Pressure sensing and patient assessment.

A major future development is the integration of pressure sensors beneath the participant's feet and force sensors on the support bars. These sensors will allow the acquisition of quantitative data on weight distribution, postural stability, and reliance on external support.

By recording these measurements across different platform configurations and EES parameters, researchers will be able to evaluate standing performance and investigate how stimulation influences the participant's ability to maintain balance.

8.3 Virtual reality integration.

Finally, a virtual-reality (VR) environment will be integrated to introduce interactive reaching tasks while participants maintain their standing position.

Combined with and pressure measurements, these tasks will allow researchers to assess upper-body movements and postural responses under controlled standing conditions, providing additional quantitative data for studying EES-enabled standing.

9 Conclusion

This project established a functional control architecture for the EES research robotic platform. The developed C++20 motor-control API integrates SocketCAN communication, MIT-mode motor commands, parabolic trajectory generation, and temporal synchronization of the three actuators. Its integration with a Qt graphical interface enables motor configuration, position monitoring, and asynchronous movement execution with cooperative Start/Stop control with a safety stop feature.

While further mechanical integration, control optimization, and safety validation are required, this work provides the software foundation for future experiments.

Ultimately, the robotic platform aims to provide a safe, controlled, and reproducible environment for studying EES-enabled standing. Combined with pressure sensing and virtual reality, it will enable quantitative assessment of postural control and contribute to a better understanding of the effects of epidural electrical stimulation on standing rehabilitation.

Acknowledgements

I would like to express my gratitude to Dr. Mohamed Bouri for giving me the opportunity to take my first steps into the fascinating field of biomedical robotics, a field to which I aspire to dedicate my future career.

I am sincerely grateful to Imane Ben M'Rad for her valuable guidance, support, and dedicated supervision throughout this project. Her advice contributed greatly to my learning experience.

I would also like to thank the entire REHAssist team for their warm welcome, their generosity in sharing their knowledge, and the wonderful working atmosphere that made this experience so enjoyable and rewarding.

Finally, my special thanks go to the EPFL Laidlaw Scholars Program team, whose support and commitment made this opportunity possible. Without them, none of this would have been possible.

References

[1] World Health Organization. *Spinal cord injury*. Fact sheet, 16 April 2024. https://www.who.int/news-room/fact-sheets/detail/spinal-cord-injury.

[2] S. Harkema, Y. Gerasimenko, J. Hodes, J. Burdick, C. Angeli, Y. Chen, C. Ferreira, A. Willhite, E. Rejc, R. G. Grossman, and V. R. Edgerton. Effect of epidural stimulation of the lumbosacral spinal cord on voluntary movement, standing, and assisted stepping after motor complete paraplegia: a case study. *The Lancet*, 377(9781):1938–1947, 2011. doi: 10.1016/S0140-6736(11)60547-3.

[3] J. F. Burke, N. Kunwar, M. S. Yaroshinsky, et al. Epidural Spinal Electrogram Provides Direct Spinal Recordings in Awake Human Participants. *Frontiers in Human Neuroscience*, 15:721076, 2021. doi: 10.3389/fnhum.2021.721076.

[4] E. Rejc, C. Angeli, and S. Harkema. Effects of lumbosacral spinal cord epidural stimulation for standing after chronic complete paralysis in humans. *PLOS ONE*, 10(7):e0133998, 2015. doi: 10.1371/journal.pone.0133998.

[5] E. Rejc, C. A. Angeli, N. Bryant, and S. J. Harkema. Effects of stand and step training with epidural stimulation on motor function for standing in chronic complete paraplegics. *Journal of Neurotrauma*, 34(9):1787–1802, 2017. doi: 10.1089/neu.2016.4516.

[6] SteadyWin *GIM6010-6* https://steadywin-motor.com/products/mini-cheetah-quadruped-robot-dynamic-module-robot-leg-high-torque-24v-bldc-reducer-gear-motor-for-cosmetic-instrument.

[7] The Qt Company. *Qt 6 Documentation: Qt Widgets*. Official documentation. https://doc.qt.io/qt-6/qtwidgets-index.html. Accessed 14 November 2026.