NITH: An Open-Source Framework for Accessible Interaction Design for Quadriplegic Users

NICOLA DAVANZO, University of Milan, Laboratory of Music Informatics, Department of Computer Science, Italy
FEDERICO AVANZINI, University of Milan, Laboratory of Music Informatics, Department of Computer Science, Italy

Individuals with motor disabilities such as quadriplegia, cerebral palsy, and other conditions which impair hands movement, encounter significant challenges when using computers and technological devices. Although advancements like voice interaction offer some relief, there remains a need for diverse and adaptable strategies tailored to each user's specific abilities. Suitable interaction methods for users with quadriplegia include eye, head, and tongue movements, along with voice, breath, whistles, mouth, and facial muscles actions.

The NITH framework provides a comprehensive software and hardware ecosystem aimed at developing accessible applications that exploit these channels. It consists of an extendable collection of peripherals capable of detecting such inputs, along with software tools and libraries for input processing, enabling rapid prototyping of new applications. Its hardware features include easy-to-construct peripherals and software adapters to interface with existing market devices, all functioning through a unified protocol. The framework includes interdependent libraries for sensor communication, data manipulation, emulation, testing, prototyping tools, and templates.

NITH's open-source philosophy ensures that all designs are freely available, prospecting community-driven improvements and enhancing affordability, customizability, and accessibility while adhering to simplicity principles.

This paper presents a comprehensive analysis of the framework, focusing on its implementation, development philosophy, case studies, and future directions.

CCS Concepts: • **Human-centered computing** → **Accessibility systems and tools**; **Accessibility technologies**; **Accessibility theory, concepts and paradigms**; **Systems and tools for interaction design**.

Additional Key Words and Phrases: Assistive Technology, Open-Source Hardware, Open-Source Software, DIY Technologies, Human-Computer Interaction, Accessibility, Interaction Design, Quadriplegia

ACM Reference Format:
Nicola Davanzo and Federico Avanzini. 2018. NITH: An Open-Source Framework for Accessible Interaction Design for Quadriplegic Users. *J. ACM* 37, 4, Article 111 (August 2018), 42 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Quadriplegia, also known as tetraplegia, is a condition marked by a partial or complete loss of motor function and sensation in all four limbs and torso [69]. Among the various issues associated with this condition, it profoundly restricts the ability of individuals to access various technological

Authors' Contact Information: Nicola Davanzo, University of Milan, Laboratory of Music Informatics, Department of Computer Science, Milan, Italy, nicola.davanzo@unimi.it; Federico Avanzini, University of Milan, Laboratory of Music Informatics, Department of Computer Science, Milan, Italy, federico.avanzini@unimi.it.

ACM 1557-735X/2018/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

Fig. 1. NITH framework logo.

resources including computers and other devices, since most of them are designed to be operated through manual interaction.

Consequently, individuals affected by quadriplegia often require assistive technologies for effective interaction with computer systems. Often, the severe limitations imposed by this condition require adaptive solutions that meet their unique needs.

Numerous pathological conditions can cause quadriplegia [85], including spinal cord injuries, amyotrophic lateral sclerosis, multiple sclerosis, and cerebral palsy. Each of these conditions presents unique challenges and requires tailored solutions to address the specific needs of affected individuals.

Providing standardized definitions for quadriplegia and other conditions is crucial for informing effective interventions. However, the capabilities of affected individuals can vary significantly. While mass-produced technologies may benefit many users, numerous accounts emphasize the need for customization, as a one-size-fits-all approach often overlooks individual needs and abilities. This oversight can lead caregivers and users to customize technologies independently through DIY methods or, in some cases, result in technological abandonment. We will provide supportive evidence and testimonies for these theses in Sec. 2.

In this article, we introduce the NITH framework, designed to address these issues at a fundamental level by facilitating rapid customization and development. NITH functions as a modular platform for designing and prototyping accessible human–computer interaction systems tailored for users with disabilities that impair the use of hands and feet as means of interaction. It integrates a set of elements, such as open-source libraries, DIY hardware sensors, and hardware wrappers that communicate via a unified protocol. The framework supports a wide range of interaction channels—including head movements, facial features, gaze tracking, and bite pressure—empowering developers to quickly create modular and customizable assistive applications, from computer control interfaces to accessible musical instruments and augmentative communication systems.

Based on principles of simplicity, modularity, and affordability, our framework aims to simplify the development of hardware and software components, allowing each element to be independently customized or replaced. On the hardware side, NITH provides a collection of DIY assistive technology (DIY-AT) devices, such as an head tracker, a breath sensor, and bite pressure detectors, accompanied by clear instructions and open designs to ensure easy replication with readily available components. The core libraries, implemented in C# with .NET 8, offer essential functionalities for parsing sensor data, emulating standard input devices, and facilitating the rapid integration of new peripherals. Moreover, the set of software tools includes wrappers for interfacing with existing hardware peripherals, sensor testing facilities, as well as tools for data manipulation, peripherals emulation, and application templates for rapid prototyping.

This paper details the architectural design of the framework, describes the implementation of its components, and demonstrates its potential through illustrative case studies that showcase both computer control and digital musical interaction.

In synthesis, the proposed framework aims to achieve several key objectives:

- Develop a comprehensive collection of peripherals that encompass multiple interaction channels.
- Facilitate rapid prototyping for software applications specifically designed for quadriplegic users.
- Establish a user-friendly human-machine interaction environment.
- Provide a structured approach for creating new sensory peripherals, which includes a standardized communication protocol to enhance functionality and interoperability.

We will further discuss how the framework's design philosophy adheres to the principles proposed in relevant literature while providing solutions to the issues identified therein.

The present paper has the following structure: in Sec. 2, we discuss the background and context of assistive technologies, focusing on the challenges faced by quadriplegic users and the need for customized solutions. Sec. 3 outlines the implementation of the NITH framework, detailing its architecture, interaction channels, and core components, including the NITH library, sensors, and wrappers. In Sec. 4, we introduce additional software tools, such as NITHtester and NITHtemplate, which support the development and testing of applications. Sec. 5 presents potential applications of the NITH framework, highlighting its versatility in creating accessible systems. Finally, in Sec. 6, we reflect on the limitations and challenges of the current framework while proposing future directions for research and development, ensuring that the framework continues to evolve to meet the needs of its users.

2 Background

We will draw some considerations on topics which are related to our framework proposal. Our work could be linked to the topics of Accessible Technologies (AT) in general, but specifically to Do-It-Yourself Assistive Technologies (DIY-AT) and Open-Source Assistive Technologies (OS-AT), since our framework's philosophy lies in giving the possibility to develop them.

This section discusses topics related to our framework proposal, particularly focusing on Accessible Technologies (AT), Do-It-Yourself Assistive Technologies (DIY-AT), and Open-Source Assistive Technologies (OS-AT). We will also explore the challenges faced by quadriplegic users when interacting with computers and technological devices, and support our thesis about the need for tailored solutions that adapt to individual capabilities.

2.1 Causes of Quadriplegia: detailing the stakeholders

As mentioned in the introductive section, quadriplegia can result from various medical conditions.

Spinal cord injury is the leading cause, particularly high-level cervical injuries (C1-C7), which result in significant functional impairment, while thoracic injuries (T1-T2) mainly affect hand and finger mobility. Incidence rates range from 13.1 to 163.4 cases per million in developed countries and 13.0 to 220.0 cases per million in developing nations [53]. Annually, an estimated 250,000 to 500,000 individuals are affected globally [99], with prevalence rates between 236 and 4,187 per million [98].

Several other medical condition may result in tetraplegia. One prominent example is amyotrophic lateral sclerosis (ALS), a degenerative disease that progressively damages nerve cells, causing a decline in essential functions such as movement, speech, and respiration [68]. Data from the National ALS Registry shows approximately 15,000 ALS diagnoses in the U.S. in 2013 [14]. Recent

studies report a global prevalence of 4.42 cases per 100,000 individuals and an incidence rate of 1.59 per 100,000 person-years [100]. Both prevalence and incidence rise with age, especially among those aged 70 to 79 years.

Multiple sclerosis is a neurological disorder characterized by the degeneration of myelin sheaths around nerve cells, resulting in sensory impairments. In 2019, there were 59,345 new MS cases and 22,439 related deaths. The condition's prevalence has increased, particularly in high socio-demographic areas like Western Europe and North America, while lower rates are observed in Asia and Africa [75].

Cerebral palsy refers to permanent movement disorders emerging in early childhood, marked by motor impairment from non-progressive brain development disturbances. It includes various movement and posture issues, often linked with cognitive, communicative, sensory, and behavioral challenges [2, 37, 72, 81]. The incidence rate in children aged 5 to 7 years is about 2.7 cases per 1,000 births, with 36% affecting infants under 2500 grams [79]. In China, the prevalence among children aged 0 to 18 years is 2.07, increasing from 1988 to 2020 [101]. Congenital disorders are a significant contributor, accounting for 53% of quadriplegic cerebral palsy cases [67]

Other potential causes of quadriplegia cases include polio, amelia, transverse myelitis [85].

2.2 Which technologies are used by quadriplegic users?

While comprehensive statistics on computer usage in the quadriplegic population are limited due to documented challenges in collecting disability data [10], a clearer understanding of digital engagement by individuals with quadriplegia emerges from research on the broader spinal cord injury (SCI) community. This data indicates a high rate of technology adoption. For instance, a 2008 study found that 69.2% of individuals with SCI used a computer and, crucially, reported no statistically significant difference in computer use based on the level of neurologic injury [36]. More recent studies confirm this trend, with one from 2017 reporting that across all age groups, computer ownership for individuals with SCI was above 80% [61].

Reinforcing these findings, proxy data from the amyotrophic lateral sclerosis (ALS) community—a condition often resulting in quadriplegia—shows high ownership rates for laptops or desktop computers (86.1%), alongside smartphones (93.5%), and tablets (75.0%) [74]. Beyond simple usage, the perceived importance of these devices is significant. A 2021 study found that computers were considered the most important assistive device by 29% of individuals with complete tetraplegia, a figure substantially higher than for those with paraplegia [97]. It is therefore evident that a substantial segment of this demographic utilizes computers and other technological devices in their daily routines.

Feng et al. [30] provided a review on how individuals with quadriplegia employ various methods to effectively use information technologies. The authors emphasize the significance of pointing devices and text entry in this context. These functions are typically supported by various systems, including head tracking, camera-based, gaze-based, and speech-based technologies. For users who retain some hand movement, input devices such as trackballs and joysticks are preferred, as they require less hand movement compared to standard mice. Additionally, while customizable interfaces featuring larger fonts and buttons are not widely adopted, they enhance user satisfaction for those who utilize them. In terms of text entry, dynamic interfaces like Dasher have been shown to increase engagement and alleviate the monotony often associated with conventional methods[1].

[1]Dasher on The Inference Group website: http://www.inference.org.uk/dasher/

Speech recognition and gaze-based systems have gained significant prominence in recent years. For instance, Nuance's MouseGrid[2] enable voice-based mouse pointing by subdividing the screen into quadrants which are selected through voice commands.

2.3 Difficulties with ATs usage

According to [28, 30], quadriplegics face several challenges when using assistive technologies (ATs) due to physical limitations and the design of available devices, thus suggesting that various improvements could be made in this field. The key issues are outlined as follows:

- **Efficiency and accuracy**: Communication inefficiencies emerge as a significant concern. For example, gaze-based systems often offer sequential input, hindering swift text entry.
- **Control precision**: Joystick-operated devices may lead to missed targets due to tremors, while gaze-based systems can struggle with tasks requiring high precision, such as photo editing.
- **Fatigue**: Users experience physical exhaustion during operation. Interactions can be demanding. For example, a prolonged use of speech recognition software can diminish voice clarity, complicating communication.
- **Dependence on others**: Many quadriplegics have significant reliance on others for setup and troubleshooting of assistive technologies. Although they can operate devices independently after initial configuration, they often express concerns about needing help with technical issues.
- **Keyboard and mouse usage**: Many users lack precision when typing, leading to difficulties in hitting the correct keys and slowing typing speed, which impacts productivity. Additionally, users frequently struggle with mouse interactions, finding it challenging to push buttons, scroll, and move the cursor efficiently.
- **Accessory and cable management**: Handling computer accessories and cables poses challenges, complicating the overall computer usage experience. It is noteworthy that only one user reported being able to use a computer independently with an assistive device.

2.4 How quadriplegics get involved with ATs

Feng et al. [30] interviewed a group of quadriplegic people, finding how they get involved with ATs.

Awareness and acquisition. The majority of the interviewed learned about assistive technologies through care facilities and educational institutions, acquiring devices primarily from these sources. In contrast, individuals not residing in such facilities often find these technologies through hospitals or research affiliated with universities, with some being introduced by family members. Importantly, none of the interviewed sought these technologies independently, with many expressing anxiety about changes and a lack of awareness regarding available options.

Learning to use. The skills required to use assistive devices were mainly developed through direct, hands-on assistance from friends, family, volunteers, or therapists, rather than relying on manuals or video tutorials. Instruction from peers in care centers was especially effective, as experienced users actively demonstrated the tools to newcomers.

[2]MouseGrid on Nuance's website: https://www.nuance.com/products/help/dragon/dragon-for-mac/enx/Content/Navigation/MouseGrid.html

2.5 Issues and abandonment

The study by Feng et al. [30] highlights that the decision to continue or abandon assistive technologies is influenced by individual symptoms, which can vary significantly. For instance, patients with progressive conditions like ALS often find themselves needing to adapt to new devices as their symptoms advance. Moreover, the effectiveness of technologies, specifically speech recognition systems, can be compromised by medical equipment like nasal pillow masks, which can hinder voice recognition and lead some users to abandon the technology entirely.

Financial considerations are also a crucial factor in the selection of assistive devices. Care centers often encounter budgetary constraints that restrict access to advanced technologies, as emphasized by Feng et al. [30]. Similarly, Orejuela and Zapata [71] report that the complexity and cost of multimodal devices significantly limit their implementation. Approximately 74% of quadriplegic individuals lack access to assistive technologies due to financial barriers, prompting a call for low-cost yet functional solutions, supported by policies. Tobias [94] highlights scale issues: the high cost of accessible technologies, combined with insufficient communication between developers and end users, contributes to products reaching only a small segment of the intended audience. The limited user base, in turn, leads to high production costs due to a lack of large-scale manufacturing.

The literature review by Jafar et al. [49] points to usability issues as a significant contributor to the high abandonment rate of assistive devices. Many users experience difficulties in effectively utilizing available technologies, which raises concerns about the appropriateness of current offerings. Orejuela and Zapata [71] further assert that many devices' specific designs often limit functionality, requiring users to manage multiple devices daily. They advocate for a unified system approach to ameliorate these challenges.

Systems have been proposed to address these issues and promote sustained use of ATs. For example, Meng et al. [63] proposed the Computer Access Approach (CAA). This multidisciplinary method assesses each individual's motor control and computer operation challenges, enabling tailored interventions that focus on specific limitations and strengths. Key components of this approach include modifying control sites, improving environmental accessibility, providing skills training, and offering ongoing support.

2.6 On the importance of customization

A key objective of the NITH framework is to facilitate customization. We aimed to develop a solution that enables the tailoring of technologies for individuals with quadriplegia, based on diverse evidence and considerations.

Previous studies which advocate for customization. Previous studies emphasize the importance of individualization in interventions. Man and Wong [59] highlight that individualization is crucial for enhancing user comfort, optimizing system performance, and improving the overall user experience. Meng et al. [63] contend that existing methods often fail to provide sufficient support, primarily due to inadequate personalized and systematic evaluation processes. Davanzo and Avanzini [21] argue that to develop a truly accessible application for individuals with quadriplegic disabilities, it is essential to design the system according to the user's remaining motor abilities, adjusting interaction parameters to utilize the most effective physical channels available for each individual. Jafar et al. [49] also advocate for tailored solutions to address both the functional and non-functional requirements of quadriplegic individuals, indicating that this could improve effectiveness and reduce abandonment rates.

Occupational Therapists often modify Assistive Technologies. Aflatoony and Kolari [1] evidenced how occupational therapists often feel the need to modify mass-produced and universally-designed assistive technologies (ATs) to fulfill the specific needs of people with disabilities.

Different degrees of impairments. In quadriplegia, varying impairment degrees lead to distinct needs and abilities, necessitating customized interventions. The impairment from spinal cord injuries, a major cause of quadriplegia, can significantly differ. Some individuals retain limited movement or sensation, while others face total paralysis. A distinction is often made between complete and incomplete quadriplegia. In incomplete cases, individuals maintain some functions below the neck, allowing for some independence. The severity of the spinal cord injury ultimately influences the degree of residual movement available to affected individuals [13]. Central cord syndrome results in greater impairment in upper limbs, while anterior spinal cord syndrome leads to total paralysis below the lesion, preserving touch and positional senses. In Cerebral Palsy (CP), variability in functional abilities is notable. Majnemer et al. [58] note that children, including those with spastic quadriplegia, exhibit significant differences in mobility, self-care, and social function due to their heterogeneity. According to [16], terms like "diplegia" and "quadriplegia" oversimplify the complexity of CP, inadequately reflecting patients' functional abilities and needs. Quadriplegia includes extreme conditions such as locked-in syndrome, which manifests as quadriplegia and anarthria yet preserves consciousness. It involves the total loss of movement capability in all parts of the body except for the eyes, which still retain the ability to move, often only vertically [89].

Evolving abilities. Individuals with quadriplegia may experience evolving abilities, necessitating ongoing assessment and adaptation of support. For instance, as previously mentioned, ALS causes gradual mobility loss and increased daily assistance needs. According to [66], motoneuron loss is linear and symmetric, with bulbar function decline occurring slower than respiratory and limb functions, thus rendering eye tracking a common resource for communication.

To get an insight on how ALS evolves, one can see the various staging systems which have been proposed to standardize the assessment of functional decline in ALS, including the King's College Staging System and the Milano-Torino (MiToS) system [26].

2.7 DIY Assistive Technologies

A solution to achieve AT customization is represented by DIY Assistive Technologies (DIY-AT). These refer to customizable technological solutions that individuals, particularly those with disabilities, can design, fabricate, or modify by themselves. This approach empowers users to take control of their assistive technology, often resulting in more personalized and effective solutions compared to mass-produced alternatives [40].

We can argue that many components of the NITH framework fits in this category, as it provides a set of tools for developing customized ATs. Moreover, *NITHsensors* (Sec. 3.5) are essentially DIY-AT blueprints which users can fabricate and customize autonomously.

DIY-AT Literature. DIY-AT have an estabilished literature. We've found a corpus of works about DIY-AT projects which have been developed to empower users through customization and accessibility. As an example:

- **ProgramA11y** [44] is a mobile application tailored for blind and visually impaired (BVI) individuals. It allows users to create and customize visual information filters using direct input, camera input, and speech recognition interfaces. In their project, the authors explored the motivations behind BVI individuals' engagement in assistive technology customization and the associated challenges, finding that existing technologies often fail to meet their

unique needs. Thus, they become "domain experts" in customizing assistive technology to adapt it to their specific requirements.

- **iCareBot** [64] focuses on elderly users by enabling them to develop their own chatbot for personal assistance. Through workshops, elderly participants were able to assemble and configure the chatbot independently. Feedback indicated that the DIY approach enhanced the usability of the system.
- **A11yBits** [42] provides a low-cost, plug-and-play toolkit for BVI users to create DIY solutions for daily activities without requiring technical expertise. Workshops demonstrated the toolkit's ease of use, with participants successfully developing 33 DIY solutions across 12 use cases. According to the authors, this approach addresses the high abandonment rates of generic assistive technologies by offering customizable and user-friendly alternatives.
- **TalkBox** [41] is a customizable communication device implemented in educational settings in Kenya. The authors argue that the project fostered community building among users and stakeholders, emphasizing the importance of customization for effective integration into school environments. User investment in customizing TalkBox was found to enhance retention rates and sustained interest in the technology.

In addition, Ariza et al. [6] conducted a comprehensive systematic literature review focusing on the impact of open-source hardware and software in the development of assistive technologies. Bohre et al. [11] discussed the benefits and challenges of adopting DIY assistive technology (DIY-AT) on a large scale by examining 27 studies, arguing that DIY-AT can provide timely, customized, and cost-effective solutions for users.

Combining the insights from these systematic reviews and projects papers, we identified and collected some common themes and concepts, which we will list below.

Pros of DIY-AT Adoption. The World Health Organization (WHO) outlines the 4 A's of Assistive Technology: availability, accessibility, appropriateness, and affordability. According to [11], DIY assistive technology (DIY-AT) significantly enhances these aspects.

- DIY-AT improves availability, as users can create devices without the delays often associated with mass-produced assistive technologies.
- The accessibility of DIY solutions allows for customization tailored to individual needs, making these technologies more effective.
- DIY-AT aligns better with local contexts, ensuring that devices are designed according to specific user requirements, which enhances overall performance and satisfaction.
- Affordability is a key advantage of DIY-AT; these devices can often be produced at a lower cost, providing a viable option for users with limited financial resources.

The adoption of DIY-AT offers several benefits. The most important highlighted aspect is that they can address the shortcomings of traditional assistive technologies, which often provide generic solutions that overlook individual differences and evolving needs. This mismatch frequently results in high abandonment rates, a challenge that DIY-AT can effectively mitigate [42]. A case study on an initiative led by Hamidi et al. [41] evidenced that DIY-AT can foster community building by bringing together individuals and organizations that typically do not engage with one another, uniting them around a shared interest. Customization was identified as a vital element for the successful integration of ATs in school settings. User investment in DIY-AT projects has been shown to enhance the retention of assistive technologies, as users feel empowered, fostering their continued interest and commitment. In Hurst et al.'s work [47], the role of rapid prototyping is underscored as a fast and affordable method for individuals to test out and customize assistive devices. By creating and sharing prototypes, users can iteratively improve designs and better meet

their personal needs. This process also contributes to community knowledge by enabling others to learn from shared experiences and designs.

Problems of the DIY-AT landscape. Despite these benefits, the analysis of the literature reveals various challenges and limitations. Ariza et al. [6] underscore the prevalence of DIY-ATs being developed and validated without the involvement of disabled individuals, which raises questions about the effectiveness and relevance of such solutions. The same authors also reveal a disparity in research attention, with many projects addressing visual impairments: the practice of DIY-AT development seems to be particularly rooted in the community of blind and visually impaired users [44, 45], while a noticeable gap exists for cognitive and learning disabilities.

Bohre et al. [11] highlight the limited access to DIY facilities, which is predominantly found in developed countries, contributing to an inequitable landscape for potential users. The same authors are concerned with the disparity between medical professionals' risk perceptions and DIY enthusiasts' expectations, which can result in inconsistent outcomes for end-users.

Mettler et al. highlight difficulties in establishing standard evaluation criteria for DIY-ATs [64]. While initial interest from facilitators and participants is promising, the sustainability of such projects needs to be assessed through long-term studies, which are often lacking, according to Hamidi et al. [41]. Mettler et al. call for researchers to expand beyond traditional academic confines and engage more directly with the real world, thereby enriching the development process [64].

DIY-ATs can be disseminated through Open-Source practices. However, the challenge of documentation in open-source development is significant. Effective documentation requires organized community engagement and robust moderation to enhance usability and provide adequate support. Since documentation efforts typically do not produce immediate research outputs, the motivation for comprehensive documentation tends to diminish in rapidly evolving systems [43].

Dissemination. According to Bohre et al. [11], a path for large-scale adoption of DIY assistive technology (DIY-AT) could encompass several key strategies: (a) promoting the use of easily accessible household materials to facilitate broader personal replication of DIY-AT devices; (b) designing assistive technology (AT) devices with modularity to enhance user experience through personalization and improved utility; (c) establishing communication channels between DIY enthusiasts and the community of disabled users to foster innovation and knowledge dissemination; (d) embedding sensors in AT devices to enable performance tracking and data collection for future enhancements; and (e) implementing systemic changes to improve the accessibility of DIY-AT in developing countries and underrepresented communities.

In order to promote DIY-AT diffusion, some projects effectively promoted workshops aimed at teaching individuals how to create their own [64].

Additionally, online communities play a crucial role throughout the adoption process, as noted by Hurst [47]. Platforms such as Make Magazine Blog, Instructables, and Thingiverse facilitate the exchange of designs, inspiration, and experiences. These communities foster an environment for both novice and expert makers, allowing them to share digital files, access instructional content, and engage in meaningful discussions.

3 Implementation

NITH framework is subdivided in multiple components including libraries, sensors and wrappers, and development tools. Tables 1 and 2 provide an overview of the contents and links to the GitHub repositories for each element.

In this section we will deal with their implementation, starting from a digression about the NITH protocol and communication channels available to people with quadriplegia, then moving to the description of each component, providing an overview and examples of usage.

Table 1. Overview of the contents of the NITH framework. In the digital version of this paper, the names are clickable and lead to the respective GitHub repository.

Name	Description
Libraries	
NITHlibrary	The main library of the NITH framework. Provides input parsing from NITH-sensors and wrappers, and some basic tools for data manipulation.
NITHemulation	An extension which provides capabilities for basic I/O emulation (e.g. to receive and send keyboard and mouse events).
NITHdmis	An extension which provides tools for Accessible Digital Musical Instruments development.
Sensors and Wrappers	
NITHsensors	Contains all the firmwares for the NITHsensors, to be uploaded on Arduino and other microcontrollers.
NITHwebcamWrapper	A NITH wrapper for webcams. Based on Google's Mediapipe, extracts features from a webcam stream to provide head, mouth and blink tracking.
Development Tools	
NITHtester	A GUI application to test NITHsensors and NITHwrappers and visualize their output, equipped with gauges and indicators.
NITHexample	A very small example of an application built using the NITH framework, for developers to see the source code.
NITHtemplate	A template for an application based on NITH framework, to be copied and modified, which suggests a specific code design philosophy.
NITHmouseController	An application built using NITH, which provides mouse emulation through head, blink, and mouth tracking.

Table 2. GitHub repositories links for each component of the NITH framework, in plain text.

Name	GitHub Link
NITHlibrary	https://github.com/LIMUNIMI/NITHlibrary
NITHemulation	https://github.com/LIMUNIMI/NITHemulation
NITHdmis	https://github.com/LIMUNIMI/NITHdmis
NITHsensors	https://github.com/Neeqstock/NITHSensors
NITHwebcamWrapper	https://github.com/LIMUNIMI/NITHwebcamWrapper
NITHtester	https://github.com/LIMUNIMI/NITHtester
NITHexample	https://github.com/LIMUNIMI/NITHexample
NITHtemplate	https://github.com/LIMUNIMI/NITHtemplate
NITHmouseController	https://github.com/LIMUNIMI/NITHmouseController

3.1 Interaction channels

Before exploring the implementation of the NITH framework, it is crucial to discuss the concept of *interaction channels*. These channels signify the different methods through which an individual with quadriplegia or similar conditions which hinder the use of hands and feet can generate significant and detectable movements or signals, which can subsequently be employed to interact with a computer or a device.

The notion of interaction channels was presented in the work of Davanzo and Avanzini [22]. This study aimed to underscore critical aspects of interaction design specifically geared toward *musicians* with motor disabilities, but the same concepts can be applied to wider contexts. In that work, the significance of personalization from a Human-Computer Interaction perspective was stressed, framing the challenge as the identification of available interaction channels for individuals with disabilities and establishing effective mappings between users' bodily movements and machine commands.

Each channel is characterized by various *parameters*, which denote different degrees of freedom. For example, the channel *head movement* encompasses parameters such as rotation in the *yaw*, *pitch*, and *roll* axes, along with movement across the *x*, *y*, and *z* axes. These parameters are instrumental in defining the interaction space for each channel.

These channels were classified into four distinct categories based on the body regions involved: *Eyes*, *Mouth*, *Head/Neck*, and *Mind*. Although some overlaps and interactions between these channels exist, each one functions as an independent mode of communication; for instance, eye movements can operate separately from mouth movements. The findings of this analysis were summarized in Tab. 2 of the original paper, portions of which are reproduced (and extended) in Tab. 3, 4 and 5 of the present paper. The *Mind* category was excluded from the NITH framework due to two factors: (a) complexity and high cost of brain-sensing technologies, which are not currently feasible for DIY-AT development; (b) difficulty to define a set of fixed parameters, which may differ a lot according to the specific technology/peripheral involved. Nonetheless, we will aim at identifying solutions for incorporating this category in future iterations of the framework, alongside the integration of wrappers for existing commercial solutions.

A thorough description and analysis of these channels is provided in the original paper [22], along with a discussion on past usage, challenges and opportunities for their implementation in assistive technologies.

As detailed in Tab. 3, 4 and 5, and discussed in Sec. 2, some interaction channels can already be detected by existing solutions within the NITHsensors and NITHwrappers collections, while some of them are still not covered. Our objective is to develop additional sensors and wrappers to comprehensively cover all identified interaction channels, thus providing a complete and customizable solution for each user.

3.2 Protocol and channels

The NITH framework employs a specific protocol to enable communication with NITHsensors and NITHwrappers. This protocol is intentionally designed to be straightforward, allowing for easy implementation across various software solutions. Consequently, developers can rapidly create new peripherals that are compatible with NITH (i.e. which output is formatted accordingly), and related software tools.

The protocol syntax can be summarized by the following template:

```
$sensor_name-version|status_codes|parameter=value&parameter=[base/value/max]^extra
```

All standard NITH output lines commence with a `$`, distinguishing them from other types of output lines and enabling straightforward line start identification. The structure of the output is divided into three primary segments, delineated by the `|` symbol, and an *extra* segment, which may be empty, separated by the `^` symbol (which should be included in the line either way). The fields are as follows:

(1) The first field comprises the sensor name and version, separated by a `-` symbol. The sensor name typically consists of words separated by underscores `"_"` and may include information

regarding the specific sensor utilized. The version is conventionally formatted as `vX.X.X` but may also incorporate additional details separated by underscores (e.g., `v.0.2.3_full`).

(2) The second field includes three-letter status codes that provide information about the current status of the peripheral. The default ones are: `OPR` (Operative status), `ERR` (Generic error status), `ICA` (Status indicating that calibration is in progress), and `NCA` (which indicates that a sensor is not calibrated nor any calibration is in progress).

(3) The third field encompasses multiple output parameters, separated by `&`. Each parameter consists of the parameter name (e.g., `yaw_acc`) paired with its corresponding value, separated by an `=` sign. Two different formats are available to report values: value only (e.g., `67.3`) or in the [base/value/max] format (e.g., `75/87.5/100`), which indicates a number within a base-max range. Values can be floating point numbers, booleans, or strings. These parameters represent the various interaction channels available to quadriplegic users.

(4) The *extra* field is designed to hold supplementary information pertinent to a particular sensor or wrapper that does not conform to the standard fields of the protocol. This may include specific control messages—such as calibration status—or error messages, as well as any parameters that are absent from the standard list. The NITHlibrary (refer to Sec. 3.3) aggregates the output from this segment; however, it does not offer standardized support for parsing the data in the *extra* field. Therefore, the utilization of this field is left at the discretion of the programmer.

A sensor can output any given number of parameters. A list of standard supported parameters is provided in Tab. 3, 4 and 5. Each parameter must be unique within the output line, and the order of the parameters is not significant.

An example of a valid NITH output line is:

```
$NITHheadAndBreathTracker-v0.1.5|OPR|
head_yaw=21.6&
head_pitch=50.3&
breath_pressure=[50/143.5/600]^
status=3;head tracker in calibration
```

The standard baud rate for NITHsensors which communicate through a serial port is set at 115200.

All the sensors and wrappers compatible with NITH *must* output their data according to this protocol, to ensure compatibility and interoperability across the entire system.

3.3 NITHlibrary

NITHlibrary is a *C♯* library that provides a set of classes and methods for interfacing with NITH-sensors and NITHwrappers. It is designed to be easy to use and to provide a simple and intuitive interface for developers to create new applications for quadriplegic users. The library is built on top of the .NET 8 framework, which should ensure cross-platform compatibility and ease of use.

The contents of the library will here be described in detail, including the main classes and their functions, the suggested workflow, and the main features provided. In order to provide a comprehensible overview, the presentation will follow the implementation of an example interactive application for quadriplegic users. Thus, the components will be presented in the same order as they would be used in its development. The example we will analyze is a simple application that uses the mouth aperture (represented by the *mouth_isOpen* parameter in table 4), detected by the *NITHwebcamWrapper* to perform mouse clicks: over a certain mouth aperture threshold, a click is performed. This example can be downloaded from its GitHub repository[3]. The example is based on

[3]NITHexample repository on GitHub: https://github.com/LIMUNIMI/NITHexample

Table 3. Standard interaction parameters (Part 1 of 3): Eyes. Abbreviations: NWW = NITHwebcamWrapper, NHT = NITHheadTracker, NBS = NITHbreathSensor, NTS = NITHteethPressureSensor. "(+P)" indicates that a preprocessor is required to extract the parameter. "(exp)" indicates experimental support, under development and testing.

Region	Parameter	Description	NITH Component
Eyes	eyeLeft_ape	Left eye aperture (continuous)	NWW
	eyeRight_ape	Right eye aperture (continuous)	NWW
	eyeLeft_isOpen	Left eye open status (boolean)	NWW (+P)
	eyeRight_isOpen	Right eye open status (boolean)	NWW (+P)
	eyeLeft_pos_x	Left eye position in X space	NWW
	eyeLeft_pos_y	Left eye position in Y space	NWW
	eyeLeft_pos_z	Left eye position in Z space	NWW
	eyeRight_pos_x	Right eye position in X space	NWW
	eyeRight_pos_y	Right eye position in Y space	NWW
	eyeRight_pos_z	Right eye position in Z space	NWW
	eyeLeft_brow_height	Left eyebrow height	NWW
	eyeRight_brow_height	Right eyebrow height	NWW
	eyeLeft_brow_phase	Left eyebrow phase (-1: down, 0: neutral, 1: up)	NWW (+P)
	eyeRight_brow_phase	Right eyebrow phase (-1: down, 0: neutral, 1: up)	NWW (+P)
	eyes_presence	Eyes presence	-
	gaze_x	Gaze coordinates on X-axis	NWW (exp)
	gaze_y	Gaze coordinates on Y-axis	NWW (exp)

the project *NITHtemplate*, which is discussed in Sec. 4.2. We advise to review that section in order to understand the structure of the example project.

Port receivers. To establish a connection with a NITH peripheral, the standard method within the NITH framework is to create a port receiver. Port receivers are contained within the `Ports` namespace and come in two types: `USBreceiver` and `UDPreceiver`, which are used to receive data from USB and UDP ports, respectively. Once the `Connect()` method is called, the port receiver starts receiving data from the specified port. It is also possible to specify the number of the port to connect to using an extension of the `Connect()` method. Both types of port receivers provide an `IsConnected()` method to check the connection status with the peripheral. They accept a list of `IPortListener` objects, which are capable of receiving the data from the port (which is in the form of string lines) and processing it.

The `NithModule` class is an `IPortListener`, so it can be connected to a port receiver to receive data from the peripheral. *In the example, a UDPreceiver is declared in the class Rack. It's set to receive data from the port 20100, the default for NITHwebcamWrapper.*

```
1    public static UDPreceiver UDPreceiver { get; set; }
```

Then the receiver is instanced in the Setup() method of the DefaultSetup class, which is called on application startup (upon MainWindow class loading)

Table 4. Standard interaction parameters (Part 2 of 3): Mouth. Refer to Table 3 for a full description of abbreviations.

Region	Parameter	Description	NITH Component
Mouth	voice_pitch	Voice pitch frequency	-
	voice_intensity	Voice intensity (volume)	-
	whistle_pitch	Whistle pitch (frequency)	-
	whistle_intensity	Whistle intensity (volume)	-
	breath_press	Breath pressure	NBS
	mouth_ape	Mouth aperture (continuous)	NWW
	mouth_isOpen	Mouth open status (boolean)	NWW (+P)
	teeth_press	Teeth pressure	NTS
	jaw_x, jaw_y, jaw_z	Jaw position on X, Y, Z axes	-
	tongue_free_x, tongue_free_y, tongue_free_z	Tongue position in free space	-
	tongue_palate_x, tongue_palate_y	Tongue position on the palate	-
	tongue_palate_pressure	Tongue pressure on the palate	-

Table 5. Standard interaction parameters (Part 3 of 3): Head. Refer to Table 3 for a full description of abbreviations.

Region	Parameter	Description	NITH Component
Head	head_presence	Head presence	NWW
	head_pos_yaw	Head rotation (yaw axis)	NHT, NWW
	head_pos_pitch	Head rotation (pitch axis)	NHT, NWW
	head_pos_roll	Head rotation (roll axis)	NHT, NWW
	head_acc_yaw	Head acceleration (yaw axis)	NHT, NWW (+P)
	head_acc_pitch	Head acceleration (pitch axis)	NHT, NWW (+P)
	head_acc_roll	Head acceleration (roll axis)	NHT, NWW (+P)
	neck_tension	Neck muscles tension	-

```
Rack.UDPreceiver = new UDPreceiver(port = 20100);
```

NithModule. The `NithModule` class is the core component of the NITH library, responsible for interfacing with NITHsensors and processing incoming data. It reads raw data strings from a NITH sensor and formats this data into `SensorData` objects, which are then dispatched to be handled by various behaviors. Indeed, most of the modules elements included in the library follow the

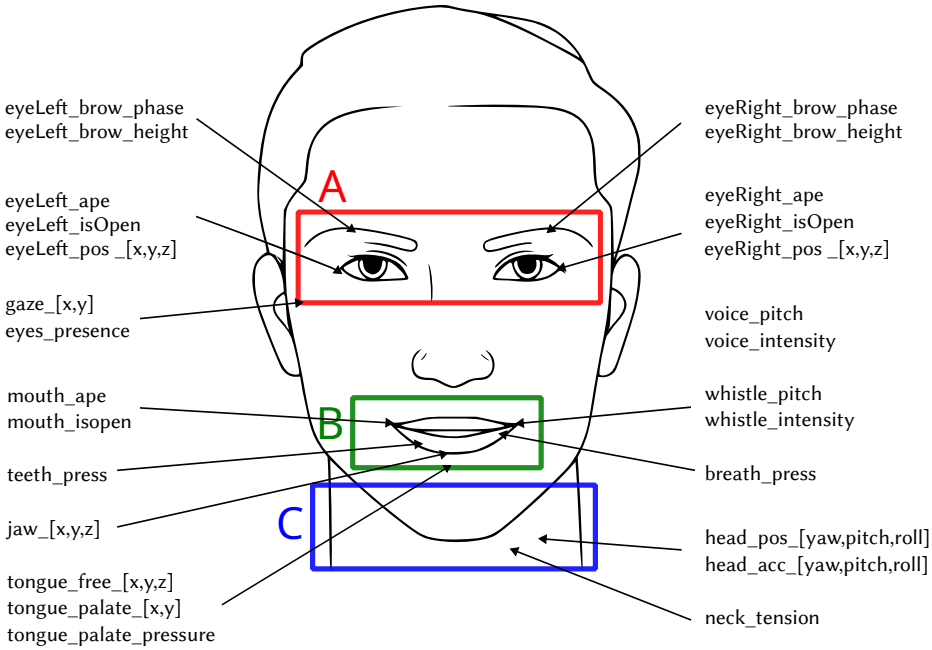

Fig. 2. Body regions for interaction channels and NITH parameters, following the grouping defined in Tab. 3, 4 and 5. Region colors are: (A) eyes, in red; (B) mouth, in green; (C) head, in blue.

Strategy design pattern, as defined in the "Gang of Four" book [33]. In the case of `NithModule`, the class implements two lists of behaviors: INithSensorBehavior and INithErrorBehavior. These are interfaces to define, respectively, behaviors associated with sensor data and error handling. The behaviors are implemented as classes that inherit from these interfaces. This allows for the dynamic selection of the associated behavior at runtime. The `NithModule` class also includes lists for expected sensor names, versions, and parameters, providing the ability to filter out input data that comes from, respectively, wrong sensor peripherals, wrong versions of the sensor peripherals, or data that does not contain the expected parameters. A specific error will be thrown in each of these cases, and the corresponding error behavior will be triggered. *In the example, we declare a NithModule in Rack*

```
1    // Make the NITH module
2    public static NithModule NithModule { get; set; }
```

Which is then instanced in the Setup() method of the DefaultSetup class, and connected to the port receiver as a listener

```
Rack.NithModule = new NithModule();
Rack.UDPreceiver.Listeners.Add(Rack.NithModule);
```

Sensor data. Incoming sensor data is parsed by the `NithModule` and subsequently packaged into `NithSensorData` objects. These objects encapsulate various attributes, including the following:

- `RawLine`: a raw data string that reflects the data received from the sensor prior to its parsing;
- `SensorName`: the name of the sensor responsible for generating the input;
- `Version`: the version of the sensor that produced the input;
- `StatusCode`: a status code provided by the sensor, which is categorized under the enumeration of `NithStatusCodes`;
- `Values`: a list of parameter values, which are represented as `NithParameterValue` objects;
- `ExtraData`: an additional string that may include supplementary information from the sensor.

Most importantly, these objects contain a list of `INithParameterValue` instances, which hold the various output arguments provided by the sensor. Furthermore, they also include important methods such as `GetParameterValue()` to retrieve the value of a specific parameter and `ContainsParameter()` to verify the presence of a particular parameter.

The `INithParameterValue` is a struct designed to encapsulate the name of the parameter, its corresponding value, and its lower and upper bounds. Furthermore, when both the value and the maximum are specified, it provides a normalized value ranging from 0 to 1, representing the ratio of the base value to the maximum value.

The `NithSensorData` object is then dispatched to the behaviors for processing.

In the example, we expect lines similar to the following (without line breaks), which will be parsed into a NithSensorData object.:

```
$NITHwebcamWrapper-0.1.0|OPR|
head_pos_pitch=16.62
&head_pos_yaw=-1.93
&head_pos_roll=3.62
&mouth_ape=0.14&eyeLeft_ape=0.3
&eyeRight_ape=0.32$
```

Sensor behaviors. Sensor behaviors are defined by the `INithSensorBehavior` interface, which plays a crucial role in managing the incoming sensor data. As previously mentioned, these behaviors are incorporated into the list of behaviors within the NithModule, enabling modifications at runtime.

A recommended workflow for structuring a sensor behavior involves the creation of a `List<NithParameters>` that contains the parameters the behavior expects to receive and is capable of managing. Within the `HandleData` method, it is essential to verify the presence of the expected parameter using an if statement that invokes the `ContainsParameter` function. This is subsequently followed by the extraction of the corresponding value through the `GetParameterValue` function.

The library features a meta-sensor-behavior `ANithBlinkEventBehavior`, which is designed to facilitate the detection of blinks. This mechanism allows for the definition of behavioral code that should be activated when blink events are detected from the left eye, the right eye, or both eyes simultaneously. It can differentiate on whether each eye is in an open or closed state. The underlying detection operates by monitoring the number of input lines (e.g. frames) in which the eyes are either open or closed, and it also offers the ability to set thresholds for the number of inputs that need to pass before a blink event is triggered.

More meta-behaviors should be implemented in future works, to facilitate the detection of other types of events.

In the example, we implemented a MouthClickBehavior class, inside the Behaviors folder and namespace. Such class inherits from the INithSensorBehavior interface. We add this behavior to the NithModule in the Setup() method of the Setup class.

```
Rack.NithModule.AddSensorBehavior(new MouthClickBehavior());
```

Error behaviors. Error behaviors are defined by the `INithErrorBehavior` interface and are specifically tasked with managing any errors that may arise during communication with the sensor. Like sensor behaviors, error behaviors are also included in a dedicated list within the NithModule, allowing for modifications to be made at runtime.

The `NithModule` can encounter a variety of errors as outlined in the `NithErrors` enumeration. These errors include `Connection`, which signifies a connection error; `OutputCompliance`, which occurs when the sensor output fails to meet the NITH communication protocol standard; `Name`, representing a scenario where the sensor name is absent from the list of supported sensor names by the `NithModule`; `Version`, indicating that the sensor version is not found on the list of supported sensor versions; `StatusCode`, which denotes an error status code (ERR) from the sensor; and `Parameters`, referring to the failure of the sensor to provide the necessary list of `NithParameters`.

Once these errors arise, they are communicated to all behaviors included in the error behaviors list, allowing them to handle the errors according to their respective implementations.

Additionally, the library offers an abstract class named `ANithErrorToStringBehavior`, which essentially serves as an error behavior designed to convert an error into a string format that is easily interpretable—thus facilitating visualization, such as on-screen display. This class also includes a virtual method, `ErrorActions`, which defines the appropriate actions to be taken with these error strings. This design follows the Template design pattern, allowing for a structured approach to error handling with flexibility in the implementation of specific actions based on the error context.

In the example, we add a simple ErrorToStringBehavior (which inherits from the abstract class ANithError-ToStringBehavior) to the NithModule in the Setup() method of the DefaultSetup class. This behavior will simply print the errors to a TextBlock in the MainWindow.

```
Rack.NithModule.ErrorBehaviors.Add(new ErrorToStringBehavior(Rack.NithModule,
↪  Rack.ConsoleTextToTextBlock));
```

Preprocessors. Preprocessors, defined by the interface `INithPreprocessor`, are specialized classes utilized for preprocessing data prior to its transmission to the behaviors. These classes modify the list of parameters within the `NithSensorData` object. They are incorporated in the list of preprocessors within the `NithModule`, and their execution follows the order in which they are added to this list. Preprocessors are instrumental in filtering, modifying, or executing any other necessary operation on the data before it is sent to the behaviors.

The library offers three sample preprocessors:

- `NithPreprocessor_HeadTrackerCalibrator` is a preprocessor designed to facilitate the calibration of head tracker data. It is specifically tailored for head trackers that detect head rotation across three axes. These movements are represented by the parameters `head_pos_yaw`, `head_pos_pitch`, and `head_pos_roll` (Tab. 5). This preprocessor enables users to establish a center position, allowing for the calibration of incoming data to that center through angular transformations.
- `NithPreprocessor_MAfilterParams` implements an exponentially decaying moving average filter for filtering each of the specified parameters. Parameters to be filtered are defined in a list provided during the instantiation of the preprocessor.
- `NithPreprocessor_WebcamWrapper` is specifically designed for use with the wrapper `NITHfacecamWrapper`. It performs several functions, including: (a) Calibration of eye and mouth apertures, taking into account the user's maximum and minimum aperture levels. Consequently, the aperture values for both eyes and mouth will be normalized to a range from 0 to 1, making these values accessible thereafter; (b)

Extraction of boolean values for eye and mouth apertures, indicating whether the aperture exceeds a specified threshold.

In the example, we add a NithPreprocessor_WebcamWrapper to the NithModule in the Setup() method of the DefaultSetup class. This preprocessor will provide calibration facilities for mouth aperture, taking into account the user's maximum and minimum aperture levels (which are continuously detected and updated). This data is used to provide an extra parameter to the NithSensorData object, which will contain the normalized mouth aperture value as well as the boolean value for the mouth aperture (open/closed).

```
// Add preprocessors
Rack.NithModule.Preprocessors.Add(new NithPreprocessor_WebcamWrapper());
```

We will also filter the mouth_ape parameter and feed the filtered value to a gauge which will provide feedback to the user on the detected mouth aperture level. We will add a NithPreprocessor_MAfilterParams to the NithModule in the Setup() method of the DefaultSetup class. This preprocessor will filter mouth_ape with a filter threshold of 0.5

```
List<NithParameters> ParamsToFilter = new List<NithParameters> {
↪  NithParameters.mouth_ape };
Rack.NithModule.Preprocessors.Add(new
↪  NithPreprocessor_MAfilterParams(ParamsToFilter, 0.5f));
```

Tools. The library contains a selection of additional tools that are organized within their respective namespaces.

The `Filters` namespace includes a variety of filters designed for the manipulation of input data. Among these are the Moving Average filters that can be applied to arrays, point coordinates (specifically, x and y), as well as numerical values.

The `Mappers` namespace comprises utility classes for performing transformations. Notably included are:

- A `SegmentMapper`, which is responsible for segmenting data into distinct ranges. This utility can be particularly useful for analyzing time series data or identifying specific events within a stream of data.
- An `AngleBaseChanger`, which provides functionality to adjust angle measurements returned by sensors such as gyroscopes. Specifically, it converts sensor-read angles to a new reference system ranging from [-180; +180]°, allowing for the adjustment of the center reference. This feature is instrumental for calibrating the gyroscope center, ensuring accurate angular readings.
- Several `VelocityCalculator` classes are also available, designed to extract the rate of change of a value over time based on positional information. For instance, they can derive head rotation speed from angular data. One class calculates velocity by simply taking the difference between the previous and current values, while another implements an exponentially decaying moving average.

The proposed workflow offers advantages in terms of sensor abstraction and data source flexibility: for example, it allows applications to operate independently of the specific sensor type employed, just recognizing the type of data input. The workflow accommodates various data sources, whether they stem from physical sensors connected via USB or software wrappers communicating over UDP: it is sufficient to connect the `NithModule` to the correct receiver. Wireless sensory peripherals, which could communicate with the computer via UDP, could be easily designed and integrated in the framework as a future development.

In the example, remapping is not necessary, but to provide an example we will remap the mouth_ape parameter (which is in the range $[0, 100]$*) and feed a visual gauge which accepts values in the range* $[0, 50]$*. In the HandleData method of the MouthClickBehavior class, we added the following code:*

```
    private SegmentMapper mapper = new SegmentMapper(0, 1, 0, 100); // Respectively:
    ↪ baseMin, baseMax, targetMin, targetMax

    public void HandleData(NithSensorData nithData)
    {
        // Check if the mouth aperture parameter is present in the data before trying
        ↪ to access it
        if (nithData.ContainsParameter(NithParameters.mouth_ape))
        {
            // Remap the mouth aperture value from the range [0, 100] to [0, 50]
            mouthAperture = double.Parse(nithData.GetParameterValue(NithParameters.m
            ↪ outh_ape).Value.Base);
            mouthAperture = mapper.Map(mouthAperture);

            // Sending the value to the mapping module for memorization prior to
            ↪ rendering
            Rack.MappingModule.MouthAperture = mouthAperture;
        }
    }
```

3.4 NITHemulation

NITHemulation is a small extension of the NITHlibrary that provides tools to emulate standard input methods, including mouse and keyboard functions. This extension has been separated from the main NITHlibrary due to its reliance on Windows-specific components, particularly the Windows Presentation Foundation (WPF) framework[4], which limits its use to WPF application development only. Calls to system functions through *.dll* libraries are also included, which further restricts the portability of this extension to Windows operating systems.

The core sections of NITHstdemu are organized within the `Mouse` and `Keyboard` namespaces.

Keyboard. The Keyboard section is implemented using the RawInputProcessor NuGet package[5]. This package allows for the handling of raw keyboard input, including detailed event handling such as key up and key down events, as well as support for multiple keyboards. The primary modules in this functionality are the `KeyboardModuleWPF` and `KeyboardModuleForms` classes. The former is tailored for WPF applications, whereas the latter is designed for Windows Forms applications. These modules operate similarly to the NithModule and other modules in the NITH framework by receiving input data and accepting a list of behaviors (specifically, `IKeyboardBehavior`) that process the data and map it to specific application behaviors. Additionally, the namespace includes the `LKeyPressStates`, `LVKeyNames`, and `VKeyCodes` enums, containing lists of all the keys that can be pressed, all the virtual key codes, and all the logical key press states (such as key down and key up), respectively.

Mouse. The implementation of the Mouse section includes both a `MouseSender` and a `MouseReceiverModule`. The `MouseSender` is a static class that provides methods to send various mouse events and emulate mouse functions, including setting cursor position, cursor visibility, simulating clicks, and moving the mouse wheel.

The `MouseReceiverModule` class serves as a module capable of storing associated `IMouseBehavior` objects to receive mouse input. The input polling rate can be finely adjusted using a timer with a configurable ratio. This module can receive mouse position data and extract current movement velocity through the `VelocityExtractor` class, as described in the Tools section of Sec. 3.3. It operates in two polling modes: Normal and FPS. In Normal mode, it outputs a `MouseModuleSample` object containing information such as

[4]WPF on .NET website: https://learn.microsoft.com/it-it/dotnet/desktop/wpf/overview/?view=netdesktop-9.0
[5]RawInputProcessor on NuGet: https://www.nuget.org/packages/RawInputProcessor

velocity, position, and movement direction. Conversely, FPS mode emulates mouse behavior in first-person shooter video games: during each polling cycle, velocity and movement directions are calculated while the cursor remains at the starting position. This mode is particularly useful in developing interactive applications where mouse movement velocity is the primary concern. A specific behavior, termed `NithSensorBehavior_GazeToMouse`, is included to straightforwardly translate gaze coordinates from a compatible sensor into mouse movements within the NITH protocol framework.

In our example, we will use the MouseSender to emulate mouse clicks. The following code is contained in the MouthClickBehavior class. The bool mouthOpen variable is declared outside the function.

```
bool mo = nithData.GetParameterValue(NithParameters.mouth_isOpen).Value.BaseAsBool;

// Use the mouth_isOpen parameter to send mouse clicks
if (mo != mouthOpen)
{
    mouthOpen = mo;
    if (mouthOpen)
    {
        MouseSender.SendMouseButtonEvent(MouseButtonFlags.LeftDown);
    }
    else
    {
        MouseSender.SendMouseButtonEvent(MouseButtonFlags.LeftUp);
    }
}
```

Console. The Console namespace includes a `ConsoleTextToTextBox` class, which enables the redirection of console output to a WPF `TextBlock` control. This functionality is particularly beneficial for redirecting sensor debug data and other pertinent information to the application interface.

In the example, we will use the ConsoleTextToTextBox class to redirect the console output to a black TextBlock in the MainWindow. It has thus been declared in the Rack class and instanced in the DefaultSetup.

3.5 NITHsensors

NITHsensors is a collection of DIY open-source hardware sensors integrated within the NITH framework. This suite includes various sensing peripherals designed to detect different interaction channels and parameters, as detailed in Tables 3, 4, 5. These peripherals are characterized by their user-friendly construction and operation.

The sensors can be connected to a computer via serial ports (USB) or through a local network utilizing the UDP protocol. The output data is formatted in accordance with the NITHprotocol (Sec. 3.2) and can be easily parsed using the tools and facilities provided by the NITHlibrary (Sec. 3.3).

The primary aim of this collection, along with the *NITHwrappers* described in Sec. 3.6, is to provide a wide array of interaction channels for quadriplegic users while ensuring the solutions are simple, affordable, and easy to reproduce and customize. This approach seeks to create a comprehensive set of tools tailored to meet diverse user needs and use cases.

This objective aligns with the observations made in Sec. 2, highlighting the necessity for customization and low-cost alternatives. To enhance economic accessibility, several principles have been adopted in the development of these peripherals, which utilize free and open-source licenses for software dissemination. The principles include:

- **Ease of construction:** The sensors can be replicated through Do-It-Yourself (DIY) methods, guided by straightforward instructions that do not require specialized electronic or manufacturing skills.
- **Use of open-source microprocessors:** The construction plans for the utilized microprocessor boards (such as Arduino Uno) are published under open-source licenses [8], enabling various manufacturers to replicate these boards and maintain low costs.

- **Use of readily available materials:** The materials employed, such as pipes, boxes, and cables, can be easily sourced from DIY stores, while transducers are accessible through e-commerce platforms.
- **Hardware reproducibility:** The construction projects and schematics of the peripherals are published under Creative Commons licenses, thus classifying them as Open-Source Hardware.
- **Open-Source control software:** The *NITHsensors* firmware is released under open-source licenses.
- **Simplicity:** The peripherals are designed to be straightforward to construct and operate, minimizing complexity in both hardware and software, in accordance with the KISS (Keep It Simple, Stupid) design principle[6]. This design philosophy facilitates swift replication and customization.

Open-Source Hardware (OSH) extends the principles of Free and Open-Source Software (FOSS) to physical devices, making their designs publicly available so that anyone can study, modify, manufacture, and distribute them [32, 70, 73]. While a single consensus on its definition is still emerging, OSH is fundamentally guided by principles of transparency, accessibility, and replicability [9, 12]. The Open-Source Hardware Association, for instance, outlines four key freedoms: to study and modify the design, and to manufacture and distribute the resulting product.

To facilitate this, thorough and open documentation is critical, as it fosters community-based development, which in turn promotes rapid innovation and reduces costs [12]. The potential of OSH is particularly evident in the advancement of scientific and medical hardware, where platforms like Arduino have become notable examples of its successful application in research, especially in developing countries [70, 73].

In the following section we will describe each peripheral which is now part of the collection, while the schematics and firmwares are available in the website of the project[7]. We will also provide a brief overview of the use of similar sensors in the literature, highlighting their applications and relevance in the context of accessibility and assistive technologies.

3.5.1 NITHbreathSensor. The `NITHbreathSensor` is able to detect breath pressure and air flow.

The construction of the NITHbreathSensor requires a small set of common electronic and hardware components. The core of the device consists of an Arduino-compatible microcontroller (e.g., Arduino Uno or Nano) and a low-pressure air sensor, such as the MPX 5010DP. These are connected using standard jumper wires. The physical interface is assembled from a flexible rubber tube that serves as the main air conduit, and a small section of PVC tubing with appropriate junctions to create a simple, interchangeable mouthpiece.

The materials needed to build a NITHbreathSensor are priced between €30 and €35[8], depending on the specific sensor model chosen and its availability in the market. Building instructions are available on the NITH project website[9].

The assembly process is simple, requiring only straightforward wiring of electrical components according to the schematic, as well as connecting the rubber tube to the pressure sensor and attaching a mouthpiece to the tube. No soldering or specialized skills are necessary. The mouthpiece is washable and interchangeable, while a cloth piece prevents saliva from damaging the electronics.

In its default setup, airflow is restricted, allowing users to breathe through their noses, enabling constant pressure application without techniques like circular breathing. For simulating wind instrument operation, creating an external air passage hole in the easily replaceable mouthpiece is simple.

The peripheral detects pressures from 0 to 10 KPa. The low-pressure sensor MPX-5010 DP, which has a sensitivity of 1 mV/mm and a response time of 1 ms. Empirical evaluations have shown a sampling rate of approximately 200Hz using similar units and sensors [20].

The Arduino firmware, which is accessible on the GitHub repository[10], is written in C++. It is possible to define, through preprocessor directives, the maximum pressure value that the sensor can detect. Additionally,

[6]KISS design principle on the Interaction Design Foundation website: https://www.interaction-design.org/literature/topics/keep-it-simple-stupid

[7]NITHsensors at Nicola Davanzo's personal website: https://neeqstock.notion.site/NITHsensors-and-NITHwrappers-56ab43db493a423f9e8823af04fa9c46

[8]Estimations were made on the Italian market in March 2025

[9]NITHbreathSensor at NITH project page: https://neeqstock.notion.site/NITHbreathSensor-5010DP-b23a43406b4d432d974a42bbe0f63695

[10]Repo: https://github.com/LIMUNIMI/NITHsensors-Firmwares

it allows for the configuration of a deadzone, which facilitates the exclusion of any pressure readings when the sensor is at rest. This feature addresses the potential issues arising from non-significant minimum pressure or inaccuracies in the sensor's construction.

A typical output from the sensor includes the parameter `breath_press` (value/max, with the deadzone accounted for).

Breath has a long usage history in the context of accessibility. Often reference is made to sip-and-puff sensors, which are often employed and find utility in diverse applications, such as wheelchair control [65]. To enhance interaction speed and efficiency, multi-level sip-and-puff sensors have been developed [34]. Moreover, innovations in demultiplexing have resulted in patented sip-and-puff devices that enable switching between different devices [92]. These devices have also been utilized for text entry on mobile phones [29, 31] and have been adapted to function as mouse devices for pointing and clicking [15, 27], including low-cost alternatives [39].

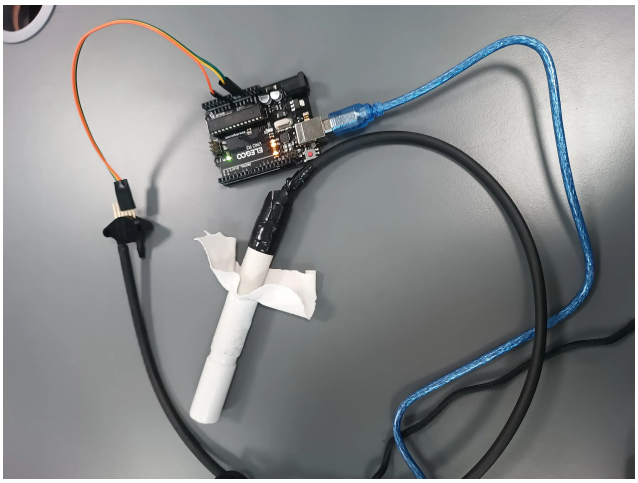

Fig. 3. A photo of a built sample of the NITHbreathPressureSensor.

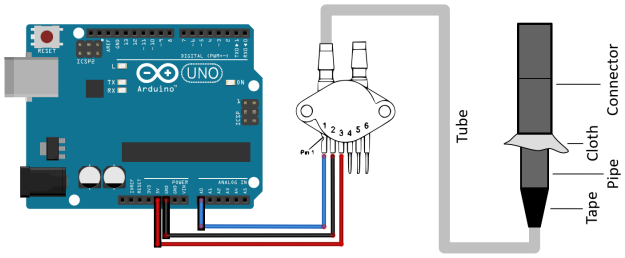

Fig. 4. Schematics for assembling the NITHbreathSensor using a MPX 5010DP low pressure sensor.

3.5.2 NITHheadTracker. The `NITHheadTracker` is a simple head tracking device that measures the orientation and angular acceleration of the head along three rotational axes: yaw, pitch, and roll. The estimated cost of the materials required for assembling the sensor is approximately €40[11], a cost that is predominantly influenced

[11]Estimations were made on the Italian market in March 2025

by the choice of breakout board. The NITHheadTracker can be effectively employed as a pointing device, including applications such as mouse emulation.

The materials necessary for assembly are widely accessible and comprise the following components:

The materials for the NITHheadTracker are readily available and low-cost. The device is built around an Arduino-compatible microcontroller and a 9-DoF Inertial Measurement Unit (IMU) breakout board, such as one based on the BNO055 sensor. Electrical connections are made with standard jumper wires. These components are then mounted on a simple wearable item, like a headset or a hairband, which serves as a stable base for the sensor.

The assembly process is relatively straightforward and does not necessitate soldering. Comprehensive building instructions are available on the NITH project website[12]. During assembly, both the Arduino and the BNO055 sensor, along with the breakout board, are mounted onto a wearable support, such as a headset or hairband and secured using insulating tape or strong duct tape.

The BNO055 is a 9-Degrees-of-Freedom (DoF) Inertial Measurement Unit (IMU) from Bosch Sensortec. Its key advantage is an integrated microcontroller that performs sensor fusion, combining data from its triaxial accelerometer, gyroscope, and magnetometer. This onboard processing is crucial as it yields absolute orientation data that is free from the rotational drift typically associated with gyroscopic sensors, ensuring stable tracking.

The output arguments can be categorized as follows:

- `head_pos_yaw`, `head_pos_pitch`, and `head_pos_roll`: These parameters represent the absolute head rotation positions measured by gyroscopic sensors. Thanks to the inclusion of a magnetometer, it is expected that these measurements will demonstrate no drift. Importantly, these outputs provide base values without imposing maximum limits.
- `head_acc_yaw`, `head_acc_pitch`, and `head_acc_roll`: These values indicate the head rotation acceleration, which is derived from the gyroscopic data. Similar to the previous set of parameters, these readings are reported as base values without a maximum threshold.

Moreover, the sensor outputs various calibration status flags (`cal_sys`, `cal_gyro`, `cal_acc`, and `cal_mag`) in the extra field: These represent the calibration status of the system, gyroscope, accelerometer, and magnetometer, respectively. In cases where the system is not fully calibrated, a slight movement of the device may enhance the calibration process. The calibration status is quantified on a scale ranging from 0 to 2.

In future developments, we plan to enhance the NITHsensors collection by introducing a wireless version of the sensor that communicates with computers using the UDP protocol. This can be achieved with a wireless Arduino module, such as the Arduino Uno R4 WiFi, which would eliminate the need for a USB cable, thereby improving overall usability and convenience.

A preliminary version of the NITHheadTracker has been tested against mouse, breath, and gaze pointing devices in Fitts' law-related tasks in a previous study [20]. The results indicated that while head tracking was not the fastest interaction method (as gaze and mouse pointing performed better), it demonstrated significant movement stability.

Although there has been no formal testing of this specific peripheral in the accessibility contest, multiple studies highlight the application of head tracking related tools in accessibility solutions. Previous systems, such as Touchless Head-Control [76] and AccessiMove [80], enable users to navigate computers using head movements, thus facilitating tasks like browsing and typing. In robotics, head tracking—used independently or alongside eye tracking—has been employed to control assistive robots, allowing individuals with severe motor impairments to engage in daily activities [48, 56, 90]. Additionally, some systems combine head tracking with speech recognition to enhance communication capabilities [54, 82]. The use of powered wheelchairs controlled by head gestures has become increasingly prevalent as an alternative to traditional joysticks [18, 35, 57, 95]. Furthermore, the recent rise of virtual reality technologies often involves head-mounted displays equipped with integrated head trackers, and the dynamics of head rotation have been thoroughly studied [83].

[12]NITHheadTracker on NITH project page: https://neeqstock.notion.site/NITHheadTracker-BNO055-eda9cb4d752c45869abd85d06a1d7e5d

Fig. 5. A photo of a built sample of the NITHheadTracker, mounted on a headband.

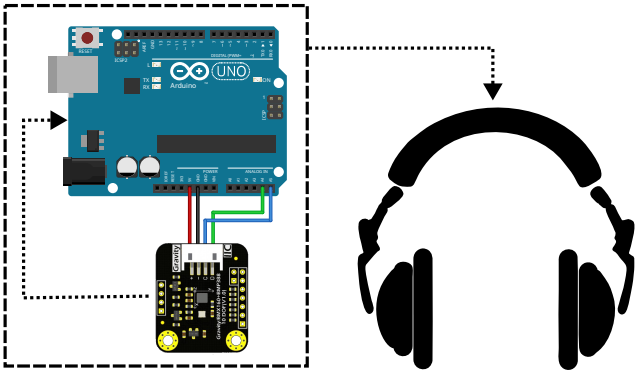

Fig. 6. Schematics for assembling the NITHheadTracker mounted on a pair of headphones, using a DFRobot BMX160 IMU sensor.

3.5.3 NITHbiteSensor. The `NITHbiteSensor` is a device specifically designed to detect dental pressure through the use of Force Sensing Resistors (FSR). These components enable precise measurement of the forces exerted by teeth. Priced at approximately €20-25, this sensor represents an affordable option for a wide range of users. The sensor is depicted in images 7 and 8.

In contrast to previously discussed sensors, the NITHbiteSensor necessitates experimentation to achieve optimal performance. It is essential to avoid excessively tightening the sensor, as this could distort the readings obtained from the FSR. A foundational understanding of the calibration process is also critical for ensuring accurate results. While soldering is not a requirement, it is advisable as it can enhance the device's reliability and functionality.

For its construction, the following materials are needed:

To construct the NITHbiteSensor, a few key components are necessary. The circuit is based on an Arduino-compatible microcontroller connected to a Force-Sensitive Resistor (FSR) and a 10 kΩ pull-down resistor. The FSR itself is mounted on a small, rigid substrate, such as a flat piece of wood, to create a biteable surface. Standard jumper wires are used for all electronic connections.

The procedure for assembling the sensor consists of replicating the electrical circuit as outlined in the assembly diagram. Subsequently, the sensor should be positioned on the wood or chosen material and secured with insulating tape. Finally, a groove must be created to accommodate the incisor teeth. Full building instructions are available on NITH project website[13].

The sensor has not yet undergone formal testing through scientific tools in any of its variants, in contrast to the previously mentioned devices. It is anticipated that testing could be conducted through an experiment similar to the one described in an already mentioned previous work [20].

Bite sensing has already found various applications in the field of assistive technologies, exemplified by several systems described in literature.

ClenchClick serves as a hands-free target selection method that integrates head movements with teeth clenching to confirm selections. Its performance, evaluated in augmented reality environments, has shown to surpass traditional hand gestures and dwell-time methods across several metrics, including workload, physical load, accuracy, and speed [86].

Some tooth-click control systems employ small clicks detected by accelerometers or microphones to facilitate cursor and button control. Studies indicate these systems outperform dwell-time control in speed and exhibit greater reliability compared to sip-and-puff control methods [87].

TeethTap, is a technique that can recognize up to 13 discrete tapping gestures. This system utilizes a wearable earpiece equipped with an inertial measurement unit (IMU) sensor and a contact microphone, achieving an impressive real-time classification accuracy of 90.9% even in noisy laboratory conditions [91].

Bitey is a discreet wearable device that captures input through tooth clicks. It employs a bone-conduction microphone to identify clicks from up to five pairs of teeth and has achieved a high accuracy of 94% in controlled settings [7].

These systems highlight the potential of bite sensing technology in enhancing assistive applications.

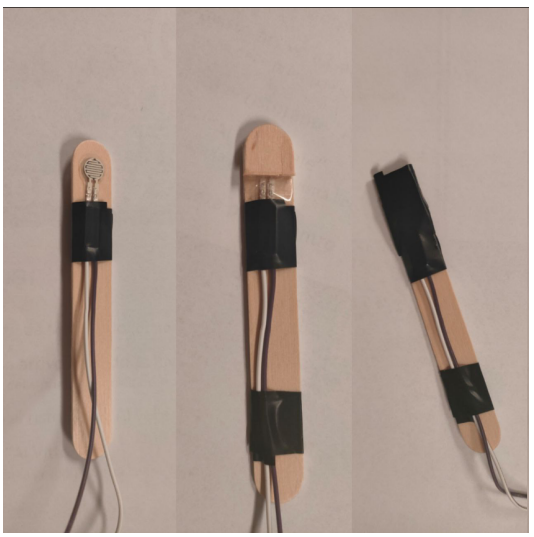

Fig. 7. A photo showing the construction stages of a NITHbiteSensor sample on a popsicle stick.

[13]NITH project page: https://neeqstock.notion.site/NITHbiteSensor-2kg-FSR-4

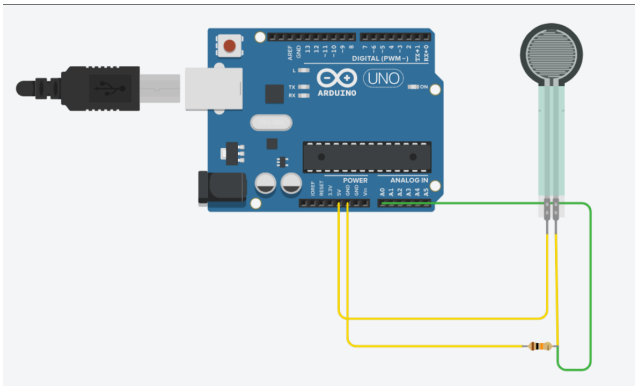

Fig. 8. Schematics for assembling the NITHbiteSensor using a Force Sensitive Resistor (FSR).

3.6 NITHwrappers

NITHwrappers are software components that encompass scripts, applications, and various other formats. Their primary function is to facilitate the interfacing of commercial sensors with the NITHlibrary, serving as software adapters. These read data from commercially available sensors and convert their output through the NITH communication protocol (described in Sec. 3.2), making it readable from NITHlibrary components.

The currently available wrappers are described below.

3.6.1 NITHwebcamWrapper. The `NITHwebcamWrapper` is a Python script that extracts facial movement features in real time by analyzing the video stream from a webcam. It estimates facial landmarks to track head pose as well as various facial characteristics.

This tool can be used with either a laptop's integrated webcam or any external webcam that frames the user's face adequately. The features currently extracted include:

- Head rotation (yaw and pitch)
- Opening of the left and right eyes
- Mouth opening

The detected features are relayed locally (over the IP address *127.0.0.1*) using the default UDP port 20100, following the standard NITH communication protocol (Sec. 3.2).

The frame rate of the detection is contingent on the specific webcam employed. The system relies on OpenCV for camera capture and calculations, while the extraction of landmarks happens through MediaPipe[14]'s *Face Mesh* model. MediaPipe is an open-source framework developed by Google that enables efficient and effective processing of perception tasks in computer vision, including face detection, hand tracking, and pose estimation. This model is initialized with defined detection and tracking confidence thresholds, which influence its sensitivity. The face mesh consists of 468 landmarks, providing a comprehensive representation of the facial structure.

Upon successful detection of facial landmarks, the program retrieves the coordinates for specific landmarks of interest. Key features calculated include head orientation (pose) and aperture ratios.

The *head orientation* is determined by applying a Perspective-n-Point (PnP) algorithm. From the 478 landmarks provided by the Mediapipe `Face Mesh` model, a specific subset of stable 3D points (e.g., nose tip, chin, eye corners) and their corresponding 2D projections on the image plane are selected. These point correspondences are fed into the `cv2.solvePnP` function, which estimates the head's rotation and translation vectors relative to the camera. The rotation vector is then converted into Euler angles (pitch, yaw) to quantify the head's pose. A separate calculation, based on the angle of the line connecting two landmarks on opposite sides of the face, is used to determine the head's roll.

[14]Google's MediaPipe repository on GitHub: https://github.com/google-ai-edge/mediapipe

For *blink detection*, the system calculates an Eye Aspect Ratio (EAR), a method validated in literature [102]. This ratio relates the height of the eye to its width. The algorithm identifies landmarks for the horizontal corners of the eye to measure its width, and landmarks on the upper and lower eyelids to measure its height. Specifically, the `get_eye_aperture_ratio_TWOSEGMENTSMETHOD` function computes the average of two vertical distances between the eyelids and divides it by the horizontal distance between the eye corners. This ratio is robust to changes in head position and distance from the camera. A blink is detected when this ratio momentarily drops below a defined threshold, which is handled by the `NithPreprocessor_WebcamWrapper` to provide boolean open/closed states.

Output is structured according to the NITH communication protocol (Sec. 3.2) and include the following parameters:

- `head_pos_pitch`, `head_pos_yaw`, `head_pos_roll` describe the position of the head, expressed in degrees; however, precision may vary.
- `eyeLeft_ape`, `eyeRight_ape` represent the eye aperture ratios. Initially, only the base values are available, but calibration through the preprocessor will also enable the retrieval of maximum values.
- `mouth_ape` signifies the mouth aperture ratio. Similar to eye ratios, the system starts with base values, with maximum values accessible post-calibration via the preprocessor.

At the moment of writing, an experimental implementation of gaze tracking is also being tested, through the open-source library *EyeTrax*[15].

The `NITHwebcamWrapper` does not, by default, provide certain values, including binary blink states (such as *eyeLeft_blink*) and calibration thresholds for eye and mouth aperture. However, the NITHlibrary features a `NithModule` preprocessor, named `NithPreprocessor_FaceCam`, which can compute these values on the client side. This preprocessor enables the client application to offer functions for calibrating the maximum and minimum aperture values of both the eyes and mouth. As a result, both the base and maximum values are made available in the output parameters.

There are two available calibration modes:

- `AutomaticContinuous`: Calibration occurs automatically, with maximum and minimum values continually updated based on incoming data, utilizing recorded aperture values as a reference.
- `Manual`: Calibration is carried out manually, requiring to define manually the maximum and minimum aperture values.

In manual calibration, a method is present for setting either the minimum or maximum thresholds to the current aperture state of the mouth or eyes. By using the preprocessor, the parameters `eyeLeft_ape`, `eyeRight_ape`, and `mouth_ape` are normalized to a range from 0 to 1, making them accessible for further processing. Additionally, the preprocessor provides the following parameters:

- `mouth_isOpen`: A boolean value that indicates whether the mouth is closed or (even slightly) open, adjusting according to the calibration mode.
- `eyeLeft_isOpen` and `eyeRight_isOpen`: Boolean values that indicate whether the respective eyes are open, also adapting based on the calibration method.

Facial gesture recognition through cameras has emerged as a promising approach in the accessibility field. Various systems utilize face tracking methods to monitor specific facial features, such as eye movements, blinks, and head positions, enabling tasks like cursor control and command execution [96]. Recent advancements by Jaiwal et al. [51] illustrate how users can navigate and interact with on-screen elements using facial gestures, including mouth opening and head rotation. The NITHmouseController, detailed in Sec. 5.1, exemplifies a mouse control system that operates solely through webcam-based face tracking via NITHwebcamWrapper.

Facial gesture recognition using cameras is a promising method in accessibility. Various systems apply face tracking techniques to monitor facial features like eye movements, blinks, and head positions, facilitating tasks such as cursor control and command execution [96]. Recent advancements by Jaiwal et al. [51] demonstrate how users navigate and interact with on-screen elements via facial gestures, such as mouth opening and head rotation. The introduction of convolutional neural networks (CNNs) has significantly improved the accuracy of these systems. For example, ResNet18 models have been effectively applied to detect facial landmarks and

[15]EyeTrax GitHub repository: https://github.com/ck-zhang/EyeTrax

translate them into cursor movements [77]. Head and nose tracking methods have also been investigated as alternatives for computer interaction, benefiting users with limited facial mobility. Notably, applications utilizing the nose tip's position for cursor control have been developed [55], along with systems that enhance user flexibility through head movements [5]. Moreover, the algorithm employed in NITHwebcamWrapper for blink detection, based on Eye Aspect Ratio (EAR) extraction, has been validated in other contexts [102]. The literature provides various methods for eye blink detection [62, 88], indicating that the current method could be refined or replaced with more advanced techniques.

NITHwebcamWrapper is available for download on the respective GitHub repository[16]

3.6.2 NITHbeamWrapper. NITHbeamWrapper serves as a wrapper for the Beam Eye Tracker[17], a software that transforms a standard webcam into an eye tracking sensor, through machine learning based techniques. This tool operates as a Python script that runs in the background while the Beam software is active.

The Beam Eye Tracker can provide essential real-time data, such as the user's gaze point on the screen and the user's head pose. The head pose expresses the user's rotational position across three axes: yaw, pitch, and roll.

The NITHbeamWrapper's workload includes reading data transmitted from Beam and converting it into the NITH communication protocol. The output parameters are, in accordance with NITHprotocol (Sec. 3.2) specifications:

- `head_pos_pitch`, `head_pos_yaw`, `head_pos_roll`: These values capture the rotational data of the user's head.
- `gaze_x`, `gaze_y`: These values denote the user's gaze position on the screen.
- `head_presence`: A boolean parameter indicating whether the user's head rotation is detectable, thus confirming if the user's head is within the webcam's field of view.
- `gaze_presence`: A boolean parameter that signifies whether the user's gaze is detectable, affirming the presence of the user's eyes within the webcam's field of view.

The application of eye tracking technology for accessibility purposes among individuals with motor disabilities has been thoroughly examined and documented in previous studies [3, 22, 23, 46].

While dedicated eye tracking hardware may provide superior performance regarding framerate, precision, and accuracy, webcam-based eye tracking systems are recently emerging as more cost-effective alternatives, yielding promising outcomes [17, 50, 93].

4 Additional software

4.1 NITHtester

NITHtester is a graphical user interface (GUI) application developed to monitor input. Its primary objective is to facilitate sensor development and testing by providing tools for users to visualize sensor data, check status codes, and identify any errors or issues effectively. It can be used for example to ensure that a just built sensor peripheral is working correctly.

This application can receive data from both *NITHsensors* and *NITHwrappers*, given their compliance with the NITH standard, and presents the information in an accessible format. The user interface includes several indicators and gauges that display parsed sensor data derived from raw input strings. Notable features include:

- Sensor name, version, status code, and additional data fields.
- A detailed list of arguments and their corresponding values, which are comfortably arranged to include base, maximum, and normalized values
- A compilation of error messages encountered during operation.

For enhanced data visualization, *NITHtester* offers graphical tools such as bindable gauges that users can configure to any argument, allowing customization of minimum, maximum, and offset parameters. Users can also adjust the moving average smoothing filter's alpha value (on a base of 100). Furthermore, a head tracker testing indicator presents two circles that represent head positions along the yaw, pitch, and roll axes. These

[16]NITHwebcamWrapper GitHub repository: https://github.com/LIMUNIMI/NITHwebcamWrapper
[17]Beam Eye Tracker website: https://beam.eyeware.tech/

indicators can be bound to position or acceleration values, with the ability to calibrate, multiply by custom factors, and smooth through a moving average filter.

An illustrative example of the *NITHtester* interface is provided in Fig. 9.

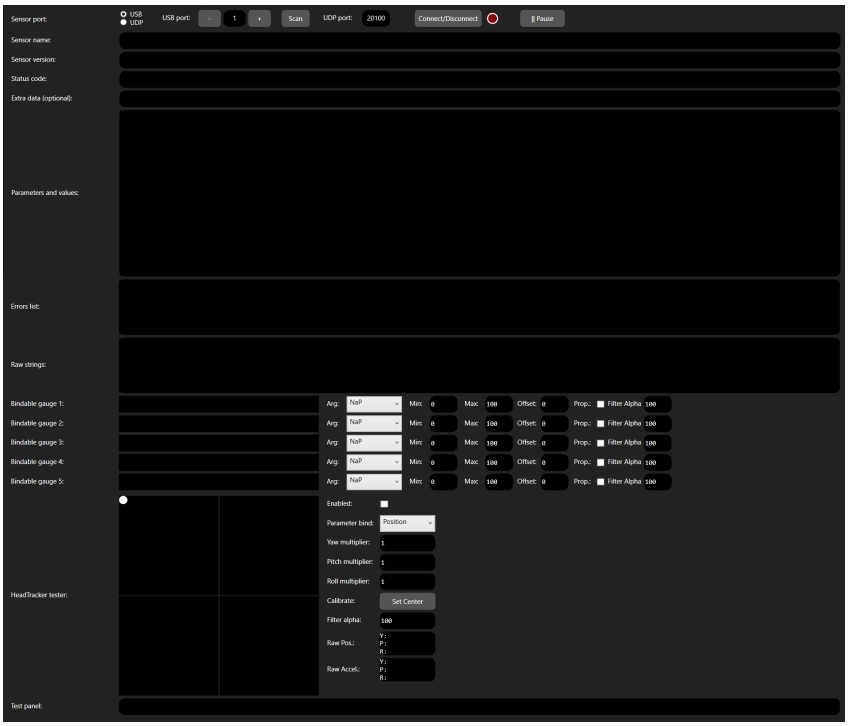

Fig. 9. A screenshot of the NITHtester interface, showing the various fields and gauges available for sensors testing.

Being developed using the *NITHlibrary* itself, *NITHtester* can be seen as a testing facility for the library's functionalities.

4.2 NITHtemplate

NITHtemplate serves as a blueprint Visual Studio C# project for the development of NITH applications. Developers can copy and edit this as a starting point for their own projects.

Adhering to the development philosophy of NITH, its structure is organized around several key classes:

- The `Rack` class is a singleton class, which can be referenced from all code, that serves as a scaffold. It can contain references to all modules and other static objects, akin to a rack of synthesizers and musical equipment, facilitating seamless connections among components. Since it's a statc class, it will be instanced at application startup, and should have better performance compared, for example to a Singleton[18], since it's stored on high frequency heap memory and its members are bonded on compile time. Moreover, such members will not be garbage collected, ensuring that they are always available through application runtime.
- The `DefaultSetup` class is tasked with initializing and configuring essential modules and their connections. It manages the creation of various components, including mapping, rendering, and communication receivers, while also handling cleanup via the `Dispose` method. It is recommended to use this class to instantiate all modules within the `Rack` and manage their interactions.

[18]Singleton design pattern on Wikipedia: https://en.wikipedia.org/wiki/Singleton_pattern

- The `MappingModule` class implements the interaction strategies for the application, containing shared variables and methods for data processing. For instance, when integrating data from a head tracker and a breath sensor, the `MappingModule` can gather data from both sensors and define the methods for their interaction.
- The `RenderingModule` class oversees the management of graphical rendering within the application (e.g. dynamic GUI elements and meters). It employs a `DispatcherTimer` to repeatedly trigger rendering operations at predetermined intervals. The `DispatcherUpdate` method can be customized to include the specific rendering logic necessary for the application's functionality.

The template comprises minimal code to illustrate the declaration and connection of various modules. NITHtemplates contains references and depends from NITHlibrary and NITHdmis libraries.

5 Applications and case study

We envision several ways in which the NITH framework could be used to develop applications for hands-free interaction:

- The primary application for which this set of tools were developed is to enable the creation of applications for computer control aimed at individuals with tetraplegia, thus replacing standard input methods or creating new interaction approaches.
- Another avenue involves the creation of Accessible Digital Musical Instruments (ADMIs) and musical interfaces that require precise and rapid control. This concept is explored in the work of Davanzo and Avanzini [22], which discusses skill-based hands-free ADMIs that, like an acoustic instrument, provide immediate feedback and necessitate the development of expertise for effective performance, allowing virtuosism.
- Additionally, applications for Augmentative and Alternative Communication (AAC) could be developed. AAC encompasses various communication methods beyond verbal speech, enabling individuals who experience difficulties with language skills to convey their thoughts effectively. The term "augmentative" refers to supplementary methods that enhance existing speech, while "alternative" refers to modes used in place of speech. Some individuals utilize AAC throughout their lives, while others may rely on it temporarily, such as during recovery from surgery that impairs speech capabilities [4].
- Furthermore, applications enabling communication with Internet of Things (IoT) devices could be developed, allowing individuals with mobility impairments to control household objects. The open-source nature of the NITH framework could facilitate centralized management of these devices through a single application on a personal computer, thereby streamlining the interaction process and contributing to the individuals' autonomy.

In order to support this vision, in the following sections we will consider and propose two case studies.

5.1 Case study: NITHmouseController

During the development of the framework, we met an individual with tetraplegia. We aimed to create an interaction solution tailored to his needs using the framework, which we will present as a case study to demonstrate its application. In the following section we will focus on the design process, rather than on the evaluation of the effectiveness of such solution.

The user, referred to as X to protect his privacy, is a 32-year-old male who sustained a spinal cord injury from a traffic accident, specifically affecting the upper cervical vertebrae and resulting in incomplete tetraplegia. Although X retains partial, imprecise control of one arm, he lacks hand mobility. He frequently uses a computer; however, he does not possess sufficient control to maneuver the mouse or type on the keyboard.

Additionally, X's injury has led to optic nerve damage, impacting his vision. He cannot see simultaneously with both eyes. He reports the ability to selectively perceive with either the right or left eye, but not both at once. He reports an almost complete degree of freedom in head movement and can control the muscles of his face.

Previously, X attempted to utilize an eye tracker for pointing; however, he found the precision and accuracy of such devices to be inadequate. This challenge may stem from the independent movement of his eyes when

focusing with one eye. We corroborated this evidence through testing with him a standard usage of the "Tobii Eye Tracker 5[19]," observing together the same difficulties that he reported.

Consequently, X relies on voice commands for computer operation, using "Nuance Dragon[20]." This allows him to dictate text, effectively substituting for traditional keyboard input, although it presents complications when typing uncommon symbols or terms not included in the application's dictionary.

For mouse control, he employs MouseGrid, a feature of Nuance that is also available in recent iterations of Windows[21]. MouseGrid functions by segmenting the screen into nine numbered sectors. By pronouncing the number corresponding to a sector, the application isolates it and recursively divides it into nine smaller regions until it reaches a sufficiently small area for accurate selection. At this point, a mouse click is executed at the designated location. X conveys that while the system works effectively, it is correspondingly very slow and often frustrating for a variety of tasks.

His peculiar situation motivated us to develop a customized interaction solution using the NITH framework. The resulting application, *NITHmouseController*, was developed to allow precise mouse control without relying on eye tracking. We present this application as a proof of concept for our design philosophy.

The `NITHmouseController` makes use of the `NITHlibrary` to facilitate the control of the mouse pointer via movements of the head, mouth, and eyelids. The position of the mouse cursor is governed by the rotation of the head around the yaw and pitch axes. This positional information can be obtained from the hardware head tracker, `NITHheadTracker`, or alternatively from a webcam through the `NITHfacecamWrapper`. While the hardware head tracker provides enhanced precision and stability, the webcam based solution serves as a viable alternative in situations where the hardware tracker is unavailable.

While using NITHmouseController, mouse clicking can be achieved in two ways.

- Through blink detection via `NITHfacecamWrapper`. A blink with the left eye corresponds to a left click, while a blink with the right eye corresponds to a right click. When the eye remains closed, the corresponding mouse button is kept pressed.
- Alternatively, mouse clicks can be managed using the mouth, again through `NITHfacecamWrapper`. The rotation around the roll axis determines the type of click, either left or right. Tilting the head to the left or right effectively selects whether the next click will be a left or right click, with the actual click executed by opening the mouth. While the mouth is open, the mouse button remains pressed.

The system incorporates a calibratable moving average filter designed to stabilize cursor movement. This feature is particularly significant for sensors that may be susceptible to noise or jitter. Additionally, it may prove beneficial for users who may struggle to maintain a steady head position, such as individuals experiencing mild muscle spasms. The user can adjust the filter's sensitivity: higher values provide more stability but may slow down the cursor, while lower values offer faster cursor movement but may be more prone to noise.

The calibration of the mouse relative to the head position occurs through two main steps: centering, where the user is instructed to keep their head in a central position while facing the screen, and the adjustment of the proportionality between head movement and mouse movement.

Furthermore, it is necessary to calibrate the opening of the mouth and eyes for click recognition. The user is required to calibrate the values for when the mouth and eyes are closed, as well as when they are fully open. Detection of aperture states (open, close) happens through the application of a double threshold on the aperture value.

The application has not yet undergone formal testing. Usability assessment questionnaires may be administered, alongside performance tests such as Fitts' Law based evaluations, such as those formalized in the standard ISO/TS 9241-411:2012[22]. These involve proposing targets for the cursor to reach and clicking, and comparing these results with the use of a mouse by able-bodied users.

[19]Tobii Eye Tracker 5 specifications on Tobii website: https://help.tobii.com/hc/en-us/articles/360012483818-Specifications-for-Eye-Tracker-5

[20]Nuance Dragon on Nuance website:https://www.nuance.com/dragon.html

[21]"Use the mouse with voice", on Microsoft support website: https://support.microsoft.com/en-us/topic/use-the-mouse-with-voice-06cfd350-e07b-4154-be93-792fbe7d34c3

[22]ISO/TS 9241-411:2012 on ISO website: https://www.iso.org/standard/54106.html

We invite developers to envision the source code[23] of this application, as a reference to study NITH framework, to develop similar solutions or to customize the interaction strategy.

Fig. 10. A screenshot of the NITHmouseController interface. On top, three console outputs for the two sensors and general information. On the bottom left, an indicator for head rotation, with two dots marking the yaw/pitch axes (white) and the pitch/roll axes (green). On the bottom right, various buttons and indicators for the mouth and eye states, as well as the calibration buttons.

5.2 Case study: Tailoring musical interaction

In its initial phases, the NITH framework was developed while working on the development to create Assistive Digital Musical Instruments (ADMIs) for individuals with tetraplegia, serving as a foundation for various of them, including *Netychords* [25]. Netychords employs the NITHlibrary, NITHdmis, and NITHemulation libraries in its source code.

The initial Netychords prototype operated using gaze pointing for chord selection and head rotation for chords strumming, enabling control over intensity and onset tempo.

The developers introduced it to a 7-year-old boy with severe dyskinetic cerebral palsy (CP). His condition resulted in significant involuntary movements and upper limb restrictions, alongside hypotonia in his lower limbs. These challenges necessitated wheelchair and walker use, as well as assistance with daily activities.

During initial meetings, developers found that the child's motor limitations hindered effective use of Netychords. While gaze pointing worked properly and was already used by the child in his daily activities, the precision needed for head movements to strum chords proved difficult, leading to frustration and disengagement. This situation required significant adaptations for effective use.

The child and his family had a limited time available for their short stay. Over the next few days, leveraging the modularity of the NITH framework, the development team implemented tailored solutions for him:

- A blink-based strumming mechanism enabled the child to strum by blinking both eyes simultaneously. Although it posed challenges in distinguishing voluntary blinks from involuntary ones, it provided an alternative method for triggering chords.
- An Auto-Strumming feature became particularly popular, enabling automatic chord playback upon gazing at the corresponding key, thus reducing the need for precise head movements.

[23] NITHmouseController source code on GitHub: https://github.com/LIMUNIMI/NITHmouseController

- Later versions of Netychords incorporated a mouth and blink-based interaction method, which is not documented in the original source paper, where mouth aperture controls sound intensity and blinking triggers chord onset.

In follow-up meetings, the child enjoyed the new features, successfully playing and improvising with his violin teacher, bringing joy to him and his family.

The results were evident when the child joined the Arts Education Program in Portugal, leading to refinements based on educator feedback, including customizable presets for different musical styles and a streamlined interface.

These adaptations significantly impacted the child's experience, enabling participation in music education and collaborative performances, fostering a sense of community and belonging.

6 Discussion

6.1 Rationale behind NITH

Here we will discuss some important points of the rationale behind its design, in light of the challenges and issues identified in the Background (Sec. 2).

Our framework emphasizes simplicity and modularity, enabling straightforward modifications and facilitating the construction of new devices as needed. Tailoring an application for an individual involves assessing which of them are available based on their specific capabilities. For example, a user may find eye tracking more convenient than head movement tracking, especially if they experience uncontrollable head spasms.

The framework addresses potential limitations associated with restricted access to do-it-yourself (DIY) facilities [11], as all devices within it can be assembled at home using minimal and household equipment. This approach aligns with the principles articulated by Bohre et al., particularly the promotion of accessible household materials for repurposing in DIY assistive technology (AT) devices, as well as the emphasis on modular designs that enhance user personalization and functionality.

The framework addresses the aforementioned World Health Organization's (WHO) *4A's* of Assistive Technology in the following way:

- **Availability**: Users are empowered to construct devices independently, circumventing the delays commonly associated with mass-produced assistive technologies.
- **Accessibility**: The framework's capacity for customization enables adaptations that specifically cater to individual requirements, thereby increasing the effectiveness of these technologies.
- **Appropriateness**: Devices are crafted in alignment with user specifications to meet diverse needs.
- **Affordability**: These devices can be replicated using simple materials at minimal cost, providing an accessible solution for individuals with limited financial resources. The problem is also evidenced by Orejuela and Zapata [71].

The concept of rapid prototyping is emphasized in the work of Hurst et al. [47], highlighting it as a swift and cost-effective method for individuals exploring and personalizing assistive devices. Our framework is positioned to facilitate this rapid prototyping process effectively by offering a common communication protocol and a dedicated testing platform.

Some of the principles discussed could prove effective only if the framework is widely adopted, benefiting from network effects. Consequently, future efforts should focus on disseminating the framework through various channels, such as online communities and workshops, which have demonstrated effectiveness in previous studies. An advancement in the project's trajectory may involve creating an online platform for sharing resources, including alternative mappings, novel sensors, and innovative designs, thereby fostering community collaboration. Ultimately, the framework could be a step forward into the direction delineated by [24], with the creation of a community based open-source framework for DIY-AT devices.

To achieve the realization of user-tailored mapping on large scales, we envision two strategies, which were already delineated in a previous work [24]:

- **Self-Configurability:** including a range of mappings directly inside a packaged tool, which can be selected (e.g. from options menus) without the need of expert assistance. For example, an application for mouse emulation can already include compatibility with multiple sensors and interaction strategies.

Alternatively, these mappings could be generated automatically from connected compatible sensors, through generative methods (e.g. machine learning) or other forms of automation;
- **Developer Intervention:** direct intervention from a developer specialized in accessible interaction, which meets the users and develops a tailored solution for them.

These strategies could be incorporated into a community-based open-source framework, where mappings developed for specific users could then be shared with the public.

Fig. 11 illustrates this proposed (and hypothetical) user-centered and community-driven workflow. The process begins with a user-centric **Case Study** to identify available interaction channels. In the **Tools gathering** and **Mapping decision** phases, a developer first consults existing framework resources or **Community** materials to find suitable peripherals and interaction mappings. If a required peripheral or mapping is not present (the 'No' or 'New' paths), the framework encourages its creation. This new device is then pre-tested (using tools like **NITHtester**) and **Shared** back into the community, creating a virtuous cycle of growth. The core **Application development** leverages the **NITHtemplate** and the modular libraries (**NITHlibrary**, **NITHemulation**, **NITHdmis**) tailored to the application type. The resulting solution undergoes iterative **Pre-testing** and refinement with the target user. Finally, long-term use of the application generates valuable feedback, which enriches the community's collective knowledge base and supports future development.

In the following sections, we will discuss some of the problems and limitations of the NITH framework, both in terms of hardware and software components, as well as the NITHlibrary itself.

6.2 Hardware Problems and Limitations

The following discussion highlights some issues and limitations related to the hardware component of the framework, which comprises the NITHsensors peripherals and NITHwrappers.

Difficulty in recreating DIY-AT solutions for some interaction channels. Some peripherals, such as gaze-tracking devices, pose significant challenges for realization through DIY practices. This technology necessitates a high degree of precision, making it difficult to construct a device that matches the precision and accuracy of commercial products. While software solutions exist that can leverage a webcam—like Beam Eye Tracker, for which we developed a NITH wrapper mentioned in Sec. 3.6.1, and some other recent solutions [52]—, and various advancements have been made in the DIY field, [38, 60, 78, 84] the performance of these solutions often still falls short in comparison to that of commercial offerings. Gaze-tracking is particularly noteworthy as it serves as one of the most promising technologies for enabling accessible applications for individuals with tetraplegia. As discussed by Davanzo and Avanzini [19], gaze remains one of the interaction channels preserved even under extreme conditions, such as in cases of locked-in syndrome and advanced stages of amyotrophic lateral sclerosis.

Moreover, the inclusion of open-source wrappers for certain mass-market sensors is often impractical, a limitation that is likely to continue. For instance, in the case of eye tracking, although low-cost solutions such as the Tobii gaming series, including the Tobii Eye-Tracker 5, are available[24], the associated libraries and SDK are not released under open-source licenses, which restricts their distribution.

Acknowledgment of Testing Limitations. While the peripherals we developed demonstrate promising functionality, it is crucial to acknowledge the lack of formal testing, which encompasses thorough measurements and assessments. This limitation is not unique to our research; similar concerns have been noted in the literature by other researchers within the DIY-AT domain [6].

While technical specifications for various sensors are accessible—mainly outlined in the components' datasheets that detail precision, accuracy, and other performance metrics—it is crucial to evaluate the ability of these devices to solve real-world problems. Assessments should involve users with disabilities and utilize methods like case studies and usability evaluations. These evaluations are essential for identifying important factors such as comfort, durability, and the overall user experience, including considerations like the comfort of use, sturdiness, and potential annoyances (e.g. related to cable presence). In particular, enhancing the sturdiness of these devices is crucial for some use cases (e.g. users with involuntary/spastic movements).

[24]Tobii Eye-Tracker 5 on Tobii website: https://gaming.tobii.com/product/eye-tracker-5/

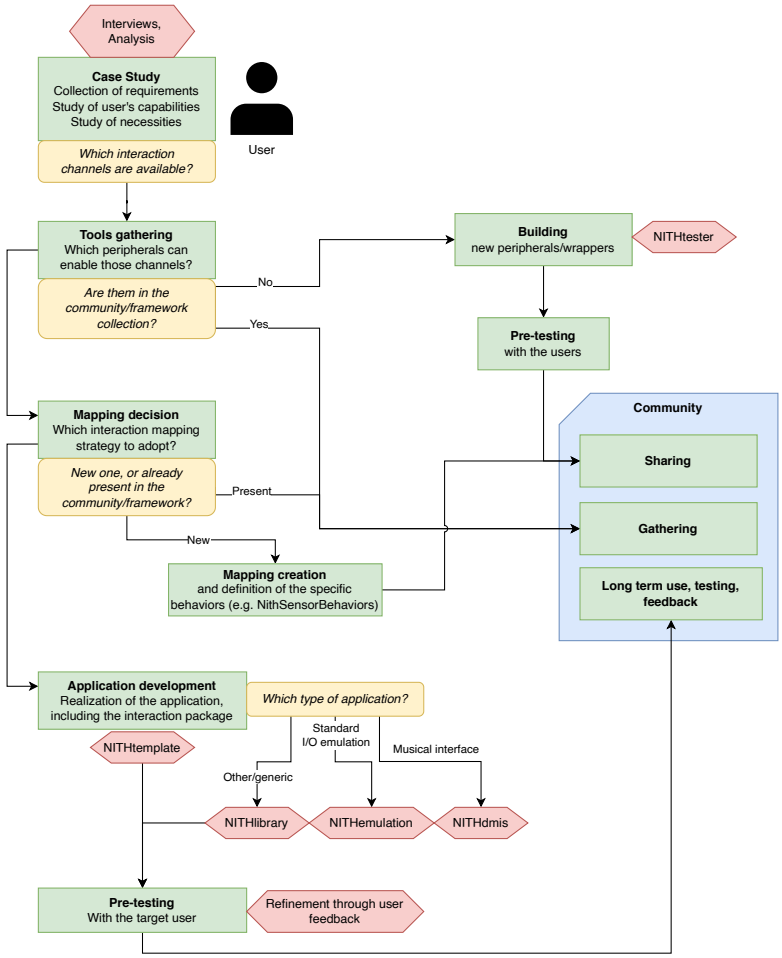

Fig. 11. The proposed workflow for developing accessible technologies with the NITH framework. In the diagram, green rectangles represent the various steps of the process, red shapes indicate specific tools, yellow rounded rectangles denote decision points, and the blue figure encompasses the community-related elements.

Providing protective casing designs can significantly improve the resilience of DIY solutions, making them more suitable for daily use by individuals with disabilities, and should be included as a future work.

We regard this work as a critical intermediary step between prototype development and the transition to a final product. Future research within our framework must concentrate on these essential areas to enhance the efficacy and user satisfaction of the peripherals.

Incomplete coverage of interaction channels. While the NITH framework provides comprehensive support for several key interaction channels, it currently does not encompass all possible modalities that could benefit quadriplegic users, as can be seen in Tables 3, 4, 5. For instance, channels such as tongue movement and voice commands are still not currently addressed.

6.3 Software and design Problems and Limitations

Expert intervention. The proposed framework aims to accelerate the rapid prototyping of new low-cost accessibility solutions for individuals with tetraplegia. However, the implementation of these systems using the NITH framework still necessitates expertise in programming. Although the system is designed specifically to eliminate the overhead associated with designing compatible solutions, customization is not performed by the tetraplegic user or their family members. Instead, it relies on an IT expert who must program the solution on their behalf. Nevertheless, we anticipate that the extreme ease of use of the library will provide an incentive for the development of innovative solutions in a straightforward and efficient manner.

Porting to other languages. Currently, NITH libraries are written in C# using .NET 8, making them cross-compilable for different operating systems. However, we acknowledge that porting the library to other languages, such as Python or C++, would enhance compatibility with a broader range of platforms and applications.

Support for Multiple Instances of the Same Channels. A planned update to the protocol aims to enable support for multiple instances of the same interaction parameter within a single sensor. Currently, each sensor allows for the declaration of only one type of parameter, such as `breath_pressure`. However, it is conceivable to have a single device with two mouthpieces connected to separate sensors (as could be, for example, in an hypothetical musical instrument where the musician can switch among two mouthpieces). To accommodate these specific edge cases, we have created the "extra" field in the protocol. Yet, a more elegant solution would be to permit the declaration of an array of parameters of the same type. A future update could address this issue while remaining backward compatible, for instance, by introducing an optional index in square brackets for the parameter, such as `breath_press[1]`.

7 Conclusions

In this work, we introduce the NITH framework, a comprehensive and modular ecosystem designed to simplify the development of accessible interaction systems for individuals with severe motor impairments which hinder the use of hands, such as quadriplegia. By incorporating open-source libraries, do-it-yourself (DIY) sensors, commercial wrappers, and development tools, the framework aims to facilitate rapid prototyping and customization of new software applications in assistive technology.

The architecture of the framework is founded on several key components. First, it identifies interaction channels suitable for the target users, based on a wide range of movements and signals that individuals can employ using body parts from the neck upwards. Second, the NITHsensors—a collection of straightforward and customizable DIY sensor peripherals—alongside NITHwrappers, which are software "adapters" for commercially available sensor peripherals, effectively capture these movements and signals. Third, the included libraries offer a versatile and extendable means to parse sensor data according to a standardized protocol, allowing for data manipulation through filtering, calibration, and various transformation facilities. This data can be dispatched to user-defined behaviors and can emulate standard computer input methods. Fourth, the framework provides templates, examples, and testing facilities to aid development. Finally, the entire framework—comprising both software and hardware components—is completely open-source, empowering the community to contribute to its development and enabling the creation of new, economically feasible solutions for end users.

Our discussion of existing assistive technologies highlights challenges in their adoption, including systems based on gaze, sip-and-puff devices, and head tracking solutions. This underscores the necessity for user-centered and adaptable designs. The literature review and case studies presented in this paper further emphasize the difficulties faced by individuals with tetraplegia, particularly their need for solutions capable of evolving alongside their changing abilities. Often, this community resorts to customization to meet their specific needs. In addressing these issues, the NITH framework prioritizes modularity and customizability, enabling interaction channels to be combined, recalibrated, and remapped according to individual requirements, thus allowing for both static and dynamically generated mappings.

The case studies discussed—such as the NITH mouse controller—demonstrate the versatility of the framework. They illustrate its potential to replace traditional input devices and empower users to undertake complex tasks, including computer control and even playing music via alternative channels such as head movements,

facial gestures, and mouth aperture. Some preliminary implementations are reportedly in use by target users [24], providing feedback and highlighting areas that require further improvement.

We have idenfified some limitations of the NITH framework, which can offer insights for future work. Some interaction channels, notably precise eye tracking and tongue movement detection, are not fully supported by DIY NITHsensors, although low-cost eye tracking solutions are available through NITHwrappers. While the current implementation in C# using .NET 8 achieves cross-platform compatibility, expanded support for languages like Python or C++ could enhance its accessibility. Furthermore, NITHsensors and NITHwrappers do not yet encompass all the interaction channels listed in Tables 3, 4, and 5, and some have not been thoroughly tested with end users. Cross-interactions between channels, which could enable simultaneous use, remain to be evaluated through experimentation. Further integration with commercial sensors may also be limited by proprietary SDKs and licensing constraints. Comprehensive validation through user studies is essential to ensure the system addresses real-world usability, comfort, and reliability. A detailed comparison of each interaction channel and their performance across different applications—similar to the analysis conducted in a previous work on breath pressure, head rotation, and gaze tracking [20]—should be pursued to provide a clearer understanding of the advantages and disadvantages of each channel.

In summary, we propose the NITH framework as a practical toolkit for designing assistive technologies that embrace openness, modularity, and rapid prototyping. It can estabilish a foundation for a community-driven approach to the development of accessible human-computer interfaces, encouraging collaboration between developers and end users to refine and expand interaction paradigms. By reducing development barriers and promoting innovation, the framework aims to bridge the gap between mainstream and specialized technologies, and to pave the way for a future in which interfaces are tailored to the unique needs of each individual. We envision that ongoing research, iterative enhancements, and active community involvement will drive its continued evolution, ultimately contributing to a more inclusive and accessible technological landscape.

Acknowledgments

Various AI Large Language Models (LLMs) have been employed for the revision of this text (including rephrasing, grammar and spelling checks). These were not involved in co-authorship of ideas and concepts, which are human-made, nor for text generation from scratch.

References

[1] L. Aflatoony and S. Kolarić. 2022. One Size Doesn't Fit All: On the Adaptable Universal Design of Assistive Technologies. *Proceedings of the Design Society* 2 (May 2022), 1209–1220. doi:10.1017/pds.2022.123

[2] Mindy Lipson Aisen, Danielle Kerkovich, Joelle Mast, Sara Mulroy, Tishya AL Wren, Robert M. Kay, and Susan A. Rethlefsen. 2011. Cerebral Palsy: Clinical Care and Neurological Rehabilitation. *The Lancet Neurology* 10, 9 (Sept. 2011), 844–852. doi:10.1016/S1474-4422(11)70176-4

[3] Amer Al-Rahayfeh and Miad Faezipour. 2015. Eye Tracking and Head Movements Detection to Assist People with Disabilities: Unveiled. In *Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering*, Tarek Sobh and Khaled Elleithy (Eds.). Springer International Publishing, Cham, 209–213. doi:10.1007/978-3-319-06773-5_28

[4] American Speech-Language-Hearing Association. last visited on 2025. Augmentative and Alternative Communication (AAC). https://www.asha.org/public/speech/disorders/AAC/.

[5] Rui Azevedo Antunes, Luís Brito Palma, Fernando V. Coito, Hermínio Duarteramos, and Paulo Gil. 2016. Intelligent Human-Computer Interface for Improving Pointing Device Usability and Performance. In *2016 12th IEEE International Conference on Control and Automation (ICCA)*. IEEE, Kathmandu, Nepal, 714–719. doi:10.1109/ICCA.2016.7505363

[6] Jonathan Álvarez Ariza and Joshua M. Pearce. 2022. Low-Cost Assistive Technologies for Disabled People Using Open-Source Hardware and Software: A Systematic Literature Review. *IEEE Access* 10 (2022), 124894–124927. doi:10.1109/ACCESS.2022.3221449

[7] Daniel Ashbrook, Carlos Tejada, Dhwanit Mehta, Anthony Jiminez, Goudam Muralitharam, Sangeeta Gajendra, and Ross Tallents. 2016. Bitey: An Exploration of Tooth Click Gestures for Hands-Free User Interface Control. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '16)*. Association for Computing Machinery, New York, NY, USA, 158–169. doi:10.1145/2935334.2935389

[8] Open Source Hardware Association. 2021. Open Source Hardware - Definition. https://www.oshwa.org/definition/.

[9] Kerstin Balka, Christina Raasch, and Cornelius Herstatt. 2014. The Effect of Selective Openness on Value Creation in User Innovation Communities. *Journal of Product Innovation Management* 31, 2 (2014), 392–407. doi:10.1111/jpim.12102

[10] Brianna Blaser and Richard E. Ladner. 2020. Why Is Data on Disability so Hard to Collect and Understand?. In *2020 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)*, Vol. 1. IEEE, Portland, OR, USA, 1–8. doi:10.1109/RESPECT49803.2020.9272466

[11] Yash Bohre, Purba Joshi, and Rowan Page. 2023. A Review of the Potential and Path to the Large-Scale Adaptation of DIY in Assistive Technology. In *Design in the Era of Industry 4.0, Volume 1*, Amaresh Chakrabarti and Vishal Singh (Eds.). Springer Nature, Singapore, 1067–1079. doi:10.1007/978-981-99-0293-4_86

[12] Jérémy Bonvoisin, Robert Mies, Jean-François Boujut, and Rainer Stark. 2017. What Is the "Source" of Open Source Hardware? *Journal of Open Hardware* 1, 1 (2017), 1–18. doi:10.14279/depositonce-6535

[13] Albert Bosch, E. Shannon Stauffer, and Vernon L. Nickel. 1971. Incomplete Traumatic Quadriplegia: A Ten-Year Review. *JAMA* 216, 3 (April 1971), 473–478. doi:10.1001/jama.1971.03180290049006

[14] Leah Bryan, Wendy Kaye, Vinicius Antao, Paul Mehta, Oleg Muravov, and D. Kevin Horton. 2016. Preliminary Results of National Amyotrophic Lateral Sclerosis (ALS) Registry Risk Factor Survey Data. *PLOS ONE* 11, 4 (April 2016), e0153683. doi:10.1371/journal.pone.0153683

[15] Erick M. Campos, Alexandre A. Freitas, Renan F. Cunha, Cassio T. Batista, Bianchi S. Meiguins, and Nelson C. Sampaio Neto. 2018. A Non-Conventional Interaction on Computational Systems Based on Mouth Puffing. In *Simpósio Brasileiro Sobre Fatores Humanos Em Sistemas Computacionais (IHC)*. SBC, Universidade Federal do Pará, Belém, Brasil, 2. doi:10.5753/ihc.2018.4185

[16] Christine Cans. 2012. Description of Children with Cerebral Palsy: Steps for the Future. *Developmental Medicine & Child Neurology* 54, 8 (2012), 679–679. doi:10.1111/j.1469-8749.2012.04336.x

[17] Cheng-Hui Chang, Jason C. Hung, and Jia-Wei Chang. 2024. Exploring the Potential of Webcam-Based Eye-Tracking for Traditional Eye-Tracking Analysis. In *Frontier Computing on Industrial Applications Volume 4*, Jason C. Hung, Neil Yen, and Jia-Wei Chang (Eds.). Springer Nature, Singapore, 313–316. doi:10.1007/978-981-99-9342-0_33

[18] Sotirios Chatzidimitriadis, Saber Mirzaee Bafti, and Konstantinos Sirlantzis. 2023. Non-Intrusive Head Movement Control for Powered Wheelchairs: A Vision-Based Approach. *IEEE Access* 11 (2023), 65663–65674. doi:10.1109/ACCESS.2023.3275529

[19] Nicola Davanzo and Federico Avanzini. 2020. A Dimension Space for the Evaluation of Accessible Digital Musical Instruments. In *Proc. 20th Int. Conf. on New Interfaces for Musical Expression (NIME '20) (NIME '20)*. Online repository, Royal Birmingham Conservatoire, 7. doi:10.5281/ZENODO.4813326

[20] Nicola Davanzo and Federico Avanzini. 2020. Experimental Evaluation of Three Interaction Channels for Accessible Digital Musical Instruments. In *Proc. '20 Int. Conf. on Computers Helping People With Special Needs*. Springer, Cham, Online Conf., 437–445.

[21] Nicola Davanzo and Federico Avanzini. 2020. Hands-Free Accessible Digital Musical Instruments: Conceptual Framework, Challenges, and Perspectives. *IEEE Access* 8 (2020), 163975–163995. doi:10.1109/ACCESS.2020.3019978

[22] Nicola Davanzo and Federico Avanzini. 2020. Hands-Free Accessible Digital Musical Instruments: Conceptual Framework, Challenges, and Perspectives. *IEEE Access* 8 (2020), 163975–163995. doi:10.1109/ACCESS.2020.3019978

[23] Nicola Davanzo and Federico Avanzini. 2022. Design Concepts for Gaze-Based Digital Musical Instruments. In *Proceedings of the 2022 Sound and Music Computing Conference*. Zenodo, Saint-Etiénne, France, 477–483.

[24] Nicola Davanzo, Federico Avanzini, Luca A. Ludovico, Davys Moreno, António Moreira, Oksana Tymoshchuk, Júlia Azevedo, and Carlos Marques. 2023. A Case Study on Netychords: Crafting Accessible Digital Musical Instrument Interaction for a Special Needs Scenario. In *Computer-Human Interaction Research and Applications*, Hugo Plácido Da Silva and Pietro Cipresso (Eds.). Vol. 1996. Springer Nature Switzerland, Cham, 353–372.

[25] Nicola Davanzo, Matteo De Filippis, and Federico Avanzini. 2021. Netychords: An Accessible Digital Musical Instrument for Playing Chords Using Gaze and Head Movements. In *In Proc. '21 Int. Conf. on Computer- Human Interaction Research and Applications (CHIRA '21)*. SciTePress, Online conf., 8.

[26] Ton Fang, Ahmad Al Khleifat, Daniel R. Stahl, Claudia Lazo La Torre, Caroline Murphy, Uk-Mnd LicalS, Carolyn Young, Pamela J. Shaw, P. Nigel Leigh, and Ammar Al-Chalabi. 2017. Comparison of the King's and MiToS Staging Systems for ALS. *Amyotrophic Lateral Sclerosis & Frontotemporal Degeneration* 18, 3-4 (Jan. 2017), 227. doi:10.1080/21678421.2016.1265565

[27] Michael G. Farnet, Harold R. McWilliams, and John J. Stutz. 2008. Sip and Puff Mouse.

[28] Fausto O. Medola, Jamille Lanutti, Claudia G. Bentim, Adrieli Sardella, Ana Elisa Franchinni, and Luis C. Paschoarelli. 2015. Experiences, Problems and Solutions in Computer Usage by Subjects with Tetraplegia. In *SciSpace - Paper*. Springer, Cham, Las Vegas, NV, USA, 131–137. doi:10.1007/978-3-319-20898-5_13

[29] Jackson Feijó Filho, Wilson Prata, and Thiago Valle. 2013. Pufftext: A Puff Controlled Software-Based Hands-Free Spin Keyboard for Mobile Phones. In *Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services (MobileHCI '13)*. Association for Computing Machinery, New York, NY, USA, 468–471. doi:10.1145/2493190.2494661

[30] Wenxin Feng, Mehrnoosh Sameki, and Margrit Betke. 2018. Exploration of Assistive Technologies Used by People with Quadriplegia Caused by Degenerative Neurological Diseases. *International Journal of Human–Computer Interaction* 34, 9 (Sept. 2018), 834–844. doi:10.1080/10447318.2017.1395572

[31] Jackson Feijó Filho, Thiago Valle, and Wilson Prata. 2013. PuffText: A Voiceless and Touchless Text Entry Solution for Mobile Phones. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '13)*. Association for Computing Machinery, New York, NY, USA, 1–2. doi:10.1145/2513383.2513424

[32] Daniel K. Fisher and Peter J. Gould. 2012. Open-Source Hardware Is a Low-Cost Alternative for Scientific Instrumentation and Research. *Modern Instrumentation* 1, 2 (April 2012), 8–20. doi:10.4236/mi.2012.12002

[33] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, Boston, USA.

[34] Christer Gerdtman and Maria Lind{\'{e}}n. 2010. Six-Button Click Interface for a Disabled User by an Adjustable Multi-level Sip-and-Puff Switch. In *Proceedings of the Proceedings of {SIGRAD} 2010: Content Aggregation and Visualization (Link{\"{o}}ping Electronic Conference Proceedings, Vol. 52)*. Link{\"{o}}ping University Electronic Press, V\"aster\aa s, Sweden, 59–63.

[35] Daniel Gomes, F. Fernandes, Eduardo Castro, and Gabriel Pires. 2019. Head-Movement Interface for Wheelchair Driving Based on Inertial Sensors. *2019 IEEE 6th Portuguese Meeting on Bioengineering (ENBENG)* 1 (2019), 1–4. doi:10.1109/ENBENG.2019.8692475

[36] Naomi Goodman, Alan M. Jette, Bethlyn Houlihan, and Steve Williams. 2008. Computer and Internet Use by Persons after Traumatic Spinal Cord Injury. *Archives of Physical Medicine and Rehabilitation* 89, 8 (Aug. 2008), 1492–1498. doi:10.1016/j.apmr.2007.12.038

[37] H. Kerr Graham, Peter Rosenbaum, Nigel Paneth, Bernard Dan, Jean-Pierre Lin, Diane L. Damiano, Jules G. Becher, Deborah Gaebler-Spira, Allan Colver, Dinah S. Reddihough, Kylie E. Crompton, and Richard L. Lieber. 2016. Cerebral Palsy. *Nature Reviews Disease Primers* 2, 1 (Jan. 2016), 15082. doi:10.1038/nrdp.2015.82

[38] Simone Guasconi, Marco Porta, Cristiano Resta, and Carlo Rottenbacher. 2017. A Low-Cost Implementation of an Eye Tracking System for Driver's Gaze Analysis. In *Proceedings of the 10th International Conference on Human System Interactions*. {IEEE}, Paris, France, 264–269. doi:10.1109/HSI.2017.8005043

[39] Junzhe Guo. 2024. Oralpunk: High-efficiency and Low-cost Oral Controller. *Science and Technology of Engineering, Chemistry and Environmental Protection* 1, 10 (Dec. 2024), 1–5. doi:10.61173/ffyhye33

[40] Foad Hamidi, Tsion Kidane, Patrick Mbullo Owuor, Michaela Hynie, and Melanie Baljko. 2023. Supporting Social Inclusion with DIY-ATs: Perspectives of Kenyan Caregivers of Children with Cognitive Disabilities. *ACM Trans. Access. Comput.* 16, 3 (Sept. 2023), 20:1–20:27. doi:10.1145/3616378

[41] Foad Hamidi, Patrick Mbullo, Deurence Onyango, Michaela Hynie, Susan McGrath, and Melanie Baljko. 2018. Participatory Design of DIY Digital Assistive Technology in Western Kenya. In *Proceedings of the Second African Conference for Human Computer Interaction: Thriving Communities (AfriCHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–11. doi:10.1145/3283458.3283478

[42] Liwen He, Yifan Li, Mingming Fan, Liang He, and Yuhang Zhao. 2023. A Multi-modal Toolkit to Support DIY Assistive Technology Creation for Blind and Low Vision People. In *Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23 Adjunct)*. Association for Computing Machinery, New York, NY, USA, 1–3. doi:10.1145/3586182.3616646

[43] Michael Heron, Vicki L. Hanson, and Ian Ricketts. 2013. Open Source and Accessibility: Advantages and Limitations. *Journal of Interaction Science* 1, 1 (May 2013), 2. doi:10.1186/2194-0827-1-2

[44] Jaylin Herskovitz. 2024. DIY Assistive Software: End-User Programming for Personalized Assistive Technology. *SIGACCESS Access. Comput.* 1, 137 (March 2024), 1. doi:10.1145/3654768.3654772

[45] Jaylin Herskovitz, Andi Xu, Rahaf Alharbi, and Anhong Guo. 2023. Hacking, Switching, Combining: Understanding and Supporting DIY Assistive Technology Design by Blind People. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–17. doi:10.1145/3544548.3581249

[46] Lida Huang, Chaomei Xu, Thomas Westin, Jerome Dupire, Florian Le Lièvre, and Xueting Shi. 2022. A Study of the Challenges of Eye Tracking Systems and Gaze Interaction for Individuals with Motor Disabilities. In *HCI International 2022 – Late Breaking Papers: HCI for Health, Well-being, Universal Access and Healthy Aging*, Vincent G. Duffy, Qin Gao, Jia Zhou, Margherita Antona, and Constantine Stephanidis (Eds.). Springer Nature Switzerland, Cham, 396–411. doi:10.1007/978-3-031-17902-0_28

[47] Amy Hurst and Jasmine Tobias. 2011. Empowering Individuals with Do-It-Yourself Assistive Technology. In *The Proceedings of the 13th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '11)*. Association for Computing Machinery, New York, NY, USA, 11–18. doi:10.1145/2049536.2049541

[48] Anja Jackowski, Marion Gebhard, and Roland Thietje. 2018. Head Motion and Head Gesture-Based Robot Control: A Usability Study. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 26, 1 (Jan. 2018), 161–170.

doi:10.1109/TNSRE.2017.2765362

[49] Mohd Rizwan Jafar and D. S. Nagesh. 2023. Literature Review on Assistive Devices Available for Quadriplegic People: Indian Context. *Disability and Rehabilitation: Assistive Technology* 18, 6 (Aug. 2023), 929–941. doi:10.1080/17483107.2021.1938708

[50] Tanmay Jain, Samiksha Bhatia, Chandan Sarkar, Priyanka Jain, and N. K. Jain. 2024. Real-Time Webcam-Based Eye Tracking for Gaze Estimation: Applications and Innovations. In *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, Vol. 1. IEEE, Mandi, Himachal Pradesh, India., 1–7. doi:10.1109/ICCCNT61001.2024.10724037

[51] Palak Jaiswal, Mansi Dhakite, Chetan Dhule, Nirmal Mungale, Sampada Wazalwar, and Atul R. Deshmukh. 17-19, May 2023. Smart AI Based Eye Gesture Control System. In *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, Madurai, India, 1873–1876. doi:10.1109/ICICCS56967.2023.10142921

[52] Tobiasz Kaduk, Caspar Goeke, Holger Finger, and Peter König. 2024. Webcam Eye Tracking Close to Laboratory Standards: Comparing a New Webcam-Based System and the EyeLink 1000. *Behavior Research Methods* 56, 5 (Aug. 2024), 5002–5022. doi:10.3758/s13428-023-02237-8

[53] Yi Kang, Han Ding, Hengxing Zhou, Zhijian Wei, Lu Liu, Dayu Pan, and Shiqing Feng. 2017. Epidemiology of Worldwide Spinal Cord Injury: A Literature Review. *Journal of Neurorestoratology* 2018, 6 (Dec. 2017), 1–9. doi:10.2147/JN.S143236

[54] Alexey A. Karpov, Andrey L. Ronzhin, Alexander I. Nechaev, and Svetlana E. Chernakova. 2004. Assistive Multimodal System Based on Speech Recognition and Head Tracking. In *Proceedings of the 13th European Signal Processing Conference*, Vol. 1. IEEE, Antalya, Turkey, 521–530.

[55] Shadman Sakib Khan, Md. Samiul Haque Sunny, M. Shifat Hossain, Eklas Hossain, and Mohiuddin Ahmad. 2017. Nose Tracking Cursor Control for the People with Disabilities: An Improved HCI. In *2017 3rd International Conference on Electrical Information and Communication Technology (EICT)*, Vol. 1. IEEE, Khulna, Bangladesh, 1–5. doi:10.1109/EICT.2017.8275178

[56] Maria Kyrarini, Quan Zheng, Muhammad Abdul Haseeb, and Axel Gräser. 2019. Robot Learning of Assistive Manipulation Tasks by Demonstration via Head Gesture-based Interface. In *2019 IEEE 16th International Conference on Rehabilitation Robotics (ICORR)*. IEEE, Toronto, ON, Canada, 1139–1146. doi:10.1109/ICORR.2019.8779379

[57] J. Leaman and H. M. La. 2017. A Comprehensive Review of Smart Wheelchairs: Past, Present, and Future. *IEEE Transactions on Human-Machine Systems* 47, 4 (Aug. 2017), 486–499. doi:10.1109/THMS.2017.2706727

[58] Annette Majnemer, Michael Shevell, Nicholas Hall, Chantal Poulin, and Mary Law. 2010. Developmental and Functional Abilities in Children With Cerebral Palsy as Related to Pattern and Level of Motor Function. *Journal of Child Neurology* 25, 10 (Oct. 2010), 1236–1241. doi:10.1177/0883073810363175

[59] David W. K. Man and Mei-Sheung Louisa Wong. 2007. Evaluation of Computer-Access Solutions for Students With Quadriplegic Athetoid Cerebral Palsy. *The American Journal of Occupational Therapy* 61, 3 (May 2007), 355–364. doi:10.5014/ajot.61.3.355

[60] Radosław Mantiuk, Michał Kowalik, Adam Nowosielski, and Bartosz Bazyluk. 2012. Do-It-Yourself Eye Tracker: Low-Cost Pupil-Based Eye Tracker for Computer Graphics Applications. In *Advances in Multimedia Modeling*, Klaus Schoeffmann, Bernard Merialdo, Alexander G. Hauptmann, Chong-Wah Ngo, Yiannis Andreopoulos, and Christian Breiteneder (Eds.). Springer, Berlin, Heidelberg, 115–125. doi:10.1007/978-3-642-27355-1_13

[61] Gillian Mayman, Marisa Perera, Michelle A. Meade, Joanna Jennie, and Eric Maslowski. 2017. Electronic Device Use by Individuals with Traumatic Spinal Cord Injury. *The Journal of Spinal Cord Medicine* 40, 4 (July 2017), 449–455. doi:10.1080/10790268.2016.1248525

[62] Paulo Augusto de Lima Medeiros, Gabriel Vinícius Souza da Silva, Felipe Ricardo dos Santos Fernandes, Ignacio Sánchez-Gendriz, Hertz Wilton Castro Lins, Daniele Montenegro da Silva Barros, Danilo Alves Pinto Nagem, and Ricardo Alexsandro de Medeiros Valentim. 2022. Efficient Machine Learning Approach for Volunteer Eye-Blink Detection in Real-Time Using Webcam. *Expert Systems with Applications* 188 (Feb. 2022), 116073. doi:10.1016/j.eswa.2021.116073

[63] Ling-Fu Meng, Tieng-Yu Li, Chi-Nung Chu, Ming-Chung Chen, Sophie Chien-Huey Chang, Arr-Mien Chou, Tony Yang, Chih Chen Hui, Ku Ai Chiao, Yun Lung Lin, Pei-ting Weng, Yu-chen Shih, Tsung-ying Lu, and Nai-hsien Yeh. 2004. Applications of Computer Access Approach to Persons with Quadriplegics. In *Computers Helping People with Special Needs*, Klaus Miesenberger, Joachim Klaus, Wolfgang L. Zagler, and Dominique Burger (Eds.). Springer, Berlin, Heidelberg, 857–864.

[64] Tobias Mettler, Stephan Daurer, Michael A. Bächle, and Andreas Judt. 2023. Do-It-Yourself as a Means for Making Assistive Technology Accessible to Elderly People: Evidence from the ICARE Project. *Information Systems Journal* 33, 1 (2023), 56–75. doi:10.1111/isj.12352

[65] I. Mougharbel, R. El-Hajj, H. Ghamlouch, and E. Monacelli. 2013. Comparative Study on Different Adaptation Approaches Concerning a Sip and Puff Controller for a Powered Wheelchair. In *Proc. of the 2013 Science and Information*

Conf., Vol. 1. IEEE, London, UK, 597–603.

[66] T. L. Munsat, P. L. Andres, L. Finison, T. Conlon, and L. Thibodeau. 1988. The Natural History of Motoneuron Loss in Amyotrophic Lateral Sclerosis. *Neurology* 38, 3 (March 1988), 409–413. doi:10.1212/wnl.38.3.409

[67] R. L. Naeye, E. C. Peters, M. Bartholomew, and R. Landis. 1990. Origins of Cerebral Palsy:. *Obstetric Anesthesia Digest* 10, 2 (July 1990), 58. doi:10.1097/00132582-199007000-00011

[68] National Institute of Neurological Disorders and Stroke. 2025. Amyotrophic Lateral Sclerosis (ALS) | National Institute of Neurological Disorders and Stroke. https://www.ninds.nih.gov/health-information/disorders/amyotrophic-lateral-sclerosis-als.

[69] National SCI Statistical Center. 2019. *Spinal Cord Injury Facts and Figures at a Glance.* Technical Report. National Spinal Cord Injury Statistical Center, University of Alabama at Birmingham.

[70] Gerrit Niezen, Parisa Eslambolchilar, and Harold Thimbleby. 2016. Open-Source Hardware for Medical Devices. *BMJ Innovations* 2, 2 (April 2016), 78–83. doi:10.1136/bmjinnov-2015-000080

[71] Juan F. Orejuela-Zapata, Sarita Rodríguez, and Gonzalo Llano Ramírez. 2019. Self-Help Devices for Quadriplegic Population: A Systematic Literature Review. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 27, 4 (April 2019), 692–701. doi:10.1109/TNSRE.2019.2901399

[72] Dilip R. Patel, Mekala Neelakantan, Karan Pandher, and Joav Merrick. 2020. Cerebral Palsy in Children: A Clinical Overview. *Translational Pediatrics* 9, S1 (Feb. 2020), S125–S135. doi:10.21037/tp.2020.01.01

[73] Joshua M. Pearce. 2012. Building Research Equipment with Free, Open-Source Hardware. *Science* 337, 6100 (Sept. 2012), 1303–1304. doi:10.1126/science.1228183

[74] Betts Peters, Kerth O'Brien, and Melanie Fried-Oken. 2024. A Recent Survey of Augmentative and Alternative Communication Use and Service Delivery Experiences of People with Amyotrophic Lateral Sclerosis in the United States. *Disability and Rehabilitation. Assistive Technology* 19, 4 (May 2024), 1121–1134. doi:10.1080/17483107.2022.2149866

[75] Zhen Qian, Yuancun Li, Zhiqiang Guan, Pi Guo, Ke Zheng, Yali Du, Shengjie Yin, Binyao Chen, Hongxi Wang, Jiao Jiang, Kunliang Qiu, and Mingzhi Zhang. 2023. Global, Regional, and National Burden of Multiple Sclerosis from 1990 to 2019: Findings of Global Burden of Disease Study 2019. *Frontiers in Public Health* 11 (Feb. 2023), 1073278. doi:10.3389/fpubh.2023.1073278

[76] Wahyu Rahmaniar, Alfian Ma'Arif, and Ting-Lan Lin. 2022. Touchless Head-Control (THC): Head Gesture Recognition for Cursor and Orientation Control. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 30 (2022), 1817–1828. doi:10.1109/TNSRE.2022.3187472

[77] Akula Kumar Raja, Chidakash Sugandhi, Gorantla Nymish, Nama Sai Havish, and Manazhy Rashmi. 2023. Convolutional Neural Network Based Virtual Mouse. In *2023 4th International Conference for Emerging Technology (INCET)*. IEEE, Belgaum, India, 1–7. doi:10.1109/INCET57972.2023.10169942

[78] Mario Rojas, Jorge A. Lopez, Frida S. Sosa, Christian R. Velasco, Pedro Ponce, David C. Balderas, and Arturo Molina. 2023. Design of a Low-Cost Prototype with 3D Printed Glasses for Real Time Eye-Tracking System. In *2023 10th International Conference on Electrical and Electronics Engineering (ICEEE)*. IEEE, Istanbul, Turkey, 52–56. doi:10.1109/ICEEE59925.2023.00017

[79] M. Rosen and J. C. Dickinson. 1992. The Incidence of Cerebral Palsy. *American journal of obstetrics and gynecology* 167 2 (1992), 417–23. doi:10.1016/S0002-9378(11)91422-7

[80] Michael Ruocco, Jack Duggan, Chetan Jaiswal, Brian O'Neill, and Karen Majeski. 2024. Leveraging {AI} Face-Tracking and Gesture Recognition for Hands-Free Computing: Bridging the Gap for Users with Physical Disabilities. In *Proceedings of the {IEEE} Global Humanitarian Technology Conference*, Vol. 1. {IEEE}, Radnor, PA, USA, 232–239. doi:10.1109/GHTC62424.2024.10771580

[81] M. Sadowska, B. Sarecka-Hujar, and I. Kopyta. 2020. Cerebral Palsy: Current Opinions on Definition, Epidemiology, Risk Factors, Classification and Treatment Options. *Neuropsychiatric Disease and Treatment* 16 (2020), 1505–1518. doi:10.2147/NDT.S235165

[82] Nazmus Sahadat, Nordine Sebkhi, and Maysam ghovanloo. 2018. Simultaneous Multimodal Access to Wheelchair and Computer for People with Tetraplegia. In *Proceedings of the 20th ACM International Conference on Multimodal Interaction (ICMI '18)*. Association for Computing Machinery, New York, NY, USA, 393–399. doi:10.1145/3242969.3242980

[83] Sarig-Bahat. 2003. Evidence for Exercise Therapy in Mechanical Neck Disorders. *Manual Therapy* 8, 1 (Feb. 2003), 10–20. pmid:12586557

[84] Valentin Schwind, Norman Pohl, and Patrick Bader. 2015. Accuracy of a Low-Cost 3D-printed Head-Mounted Eye Tracker. In *Mensch Und Computer 2015 – Tagungsband*, Martin Pielot, Sarah Diefenbach, and Niels Henze (Eds.). De Gruyter, Radnor, PA, USA, 259–262. doi:10.1515/9783110443929-028

[85] Andrew Sears, Mark Young, and Jinjuan Feng. 2008. Physical Disabilities and Computing Technologies: An Analysis of Impairments. In *The Human-Computer Interaction Handbook* (2 ed.), Andrew Sears and Julie A. Jacko (Eds.). CRC

Press, United States, Chapter 42, 829–852.
[86] Xiyuan Shen, Yukang Yan, Chun Yu, and Yuanchun Shi. 2022. ClenchClick: Hands-Free Target Selection Method Leveraging Teeth-Clench for Augmented Reality. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 6, 3 (Sept. 2022), 139:1–139:26. doi:10.1145/3550327
[87] Tyler Simpson ^*, Colin Broughton, Michel J. A. Gauthier, and Arthur Prochazka. 2008. Tooth-Click Control of a Hands-Free Computer Interface. *IEEE Transactions on Biomedical Engineering* 55, 8 (Aug. 2008), 2050–2056. doi:10.1109/TBME.2008.921161
[88] Hari Singh and Jaswinder Singh. 2018. Real-Time Eye Blink and Wink Detection for Object Selection in HCI Systems. *Journal on Multimodal User Interfaces* 12, 1 (March 2018), 55–65. doi:10.1007/s12193-018-0261-7
[89] Eimear Smith and Mark Delargy. 2005. Locked-in Syndrome. *BMJ* 330, 7488 (Feb. 2005), 406–409. doi:10.1136/bmj.330.7488.406
[90] Sarah Stalljann, Lukas Wöhle, Jeroen Schäfer, and Marion Gebhard. 2020. Performance Analysis of a Head and Eye Motion-Based Control Interface for Assistive Robots. *Sensors* 20, 24 (Jan. 2020), 7162. doi:10.3390/s20247162
[91] Wei Sun, Franklin Mingzhe Li, Benjamin Steeper, Songlin Xu, Feng Tian, and Cheng Zhang. 2021. TeethTap: Recognizing Discrete Teeth Gestures Using Motion and Acoustic Sensing on an Earpiece. In *Proceedings of the 26th International Conference on Intelligent User Interfaces (IUI '21)*. Association for Computing Machinery, New York, NY, USA, 161–169. doi:10.1145/3397481.3450645
[92] Ying Sun, Kaylen J. Haley, Kerri-Anne Lachance, and Kerri Pinnock. 2004. Pneumatic Demultiplexer for Controlling Multiple Assistive Technology Devices.
[93] Aryaman Taore. 2023. Webcam Eye Tracking. doi:10.31219/osf.io/eqtak
[94] Jim Tobias. 2003. Information Technology and Universal Design: An Agenda for Accessible Technology. *Journal of Visual Impairment & Blindness* 97, 10 (Oct. 2003), 592–601. doi:10.1177/0145482X0309701004
[95] Fitri Utaminingrum, Yuita Arum Sari, Putra Pandu Adikara, Dahnial Syauqy, and Sigit Adinugroho. 2018. Hybrid Head Tracking for Wheelchair Control Using Haar Cascade Classifier and KCF Tracker. *TELKOMNIKA (Telecommunication Computing Electronics and Control)* 16, 4 (Feb. 2018), 1616–1624. doi:10.12928/telkomnika.v16i4.6595
[96] Norman H. Villaroman, Dale C. Rowe, and Richard G. Helps. 2013. Design and Evaluation of Face Tracking User Interfaces for Accessibility. In *Proceedings of the 2nd Annual Conference on Research in Information Technology (RIIT '13)*. Association for Computing Machinery, New York, NY, USA, 65–70. doi:10.1145/2512209.2512218
[97] Aylin Wagner, Cora Schweizer, Elias Ronca, and Armin Gemperli. 2023. The Most Important Assistive Devices for Persons with Spinal Cord Injury in Switzerland: A Cross-Sectional Study. *Disabilities* 3, 3 (Sept. 2023), 367–378. doi:10.3390/disabilities3030024
[98] Christopher D. Witiw and Michael G. Fehlings. 2015. Acute Spinal Cord Injury. *Clinical Spine Surgery* 28, 6 (July 2015), 202. doi:10.1097/BSD.0000000000000287
[99] World Health Organization. 2013. *Spinal Cord Injury Report.* Technical Report. World Health Organization.
[100] Lu Xu, Tanxin Liu, Lili Liu, Xiaoying Yao, Lu Chen, Dongsheng Fan, Siyan Zhan, and Shengfeng Wang. 2020. Global Variation in Prevalence and Incidence of Amyotrophic Lateral Sclerosis: A Systematic Review and Meta-Analysis. *Journal of Neurology* 267, 4 (April 2020), 944–953. doi:10.1007/s00415-019-09652-y
[101] S Yang, J Xia, J Gao, and L Wang. 2021. Increasing Prevalence of Cerebral Palsy among Children and Adolescents in China 1988–2020: A Systematic Review and Meta-Analysis. *Journal of Rehabilitation Medicine* 53, 5 (2021), 10. doi:10.2340/16501977-2841
[102] Lu Youwei. 2023. Real-Time Eye Blink Detection Using General Cameras: A Facial Landmarks Approach. *International Science Journal of Engineering & Agriculture* 2, 5 (Oct. 2023), 1–8. doi:10.46299/j.isjea.20230205.01

Received TBD; revised TBD; accepted TBD