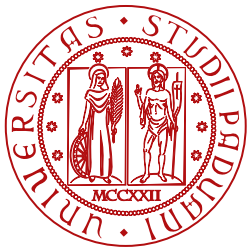

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DEPARTMENT OF INFORMATION ENGINEERING

BACHELOR DEGREE IN BIOMEDICAL ENGINEERING

Optimization of Array Designer: A MATLAB GUI for fNIRS Optode Placement

SUPERVISIOR
Prof. Brigadoi Sabrina, PhD

GRAD STUDENT
Campagnolo Mattia

ACADEMIC YEAR 2024-2025

GRADUATION DATE Jul. 22, 2025

Abstract

In the early 1990s, studies on the optical properties of light led to the development of functional near-infrared spectroscopy (fNIRS), a non-invasive neuroimaging technique that allows the monitoring of brain activity by measuring changes in blood oxygenation. Thanks to its portability, low cost and safe use, fNIRS has found wide application in the study of the brain in neonatology and paediatrics contexts, as well as to study brain activity during everyday activities, where other techniques are less feasible. The use of optodes placed on the scalp makes it possible to collect haemodynamic data without the need for invasive procedures. However, to date, no universally recognised standardisation system exists for optimal optodes positioning, which has led to the development of customisable software solutions—many of them based on programming environments such as MATLAB—aimed to optimize the array configuration based on the available equipment.

The work reported in this thesis aims to illustrate the changes and implementations made to the source code of *ROI Builder*, one of the two main modules of the *Array Designer* software, designed to support the design of optode arrays based on the definition of Regions of Interest (ROIs) on the cortical surface that the user would like to investigate. The changes introduced are intended to improve the tool's usability and flexibility, in order to facilitate the design of customised arrays according to experimental needs. In this context, the use of MATLAB's App Designer environment is also explored, with a focus on key concepts such as the creation of a graphical interface (GUI) and the implementation of callback functions—key elements for managing user interaction.

To support the technical analysis, the thesis also provides a theoretical overview of functional near-infrared spectroscopy, including the description of the forward modelling and the use of the MNI (Montreal Neurological Institute) coordinate system, which enables each point in cerebral space to be uniquely defined using a right-handed orthogonal axes.

Sommario

Nei primi anni ’90 del Novecento, gli studi sulle proprietà ottiche della luce hanno portato allo sviluppo della spettroscopia funzionale nel vicino infrarosso (fNIRS), una tecnica di neuroimaging non invasiva che consente di monitorare l’attività cerebrale attraverso la misurazione delle variazioni di ossigenazione del sangue. Grazie alla sua portabilità, al costo contenuto e alla sicurezza d’uso, la fNIRS ha trovato ampia applicazione nello studio del cervello in ambito neonatale e pediatrico e nello studio dell’attività cerebrale durante compiti quotidiani, dove altre tecniche risultano meno praticabili. L’utilizzo di optodi posizionati sullo scalpo rende possibile la raccolta di dati emodinamici senza la necessità di procedure invasive. Tuttavia, ad oggi non esiste un sistema di standardizzazione universalmente riconosciuto per il posizionamento ottimale degli optodi, il che ha portato allo sviluppo di soluzioni software personalizzabili, molte delle quali basate su ambienti di programmazione come MATLAB, e volte a ottimizzare la configurazione degli array in base alla strumentazione disponibile.

Il presente elaborato ha l’obiettivo di illustrare le modifiche e le implementazioni apportate al codice sorgente di *ROI Builder*, uno dei due moduli principali del software *Array Designer*, progettato per supportare lo sviluppo di array di optodi basati sulla definizione di Regioni di Interesse (ROIs) della superficie corticale che l’utilizzatore vorrebbe indagare. Le modifiche effettuate mirano a migliorare l’usabilità e la flessibilità dello strumento, con l’intento di facilitare la progettazione di array personalizzati in funzione delle esigenze sperimentali. In questo contesto, viene inoltre approfondito l’utilizzo dell’ambiente App Designer di MATLAB, chiarendo i concetti fondamentali legati alla creazione di un’interfaccia grafica (GUI) e all’implementazione delle funzioni di callback, elementi chiave per la gestione dell’interazione con l’utente.

A supporto dell’analisi tecnica, la tesi fornisce anche una trattazione teorica della spettroscopia funzionale nel vicino infrarosso, includendo la descrizione del problema diretto (*forward problem*) e l’impiego del sistema di coordinate MNI (Montreal Neurological Institute), che consente di definire in maniera univoca i punti nello spazio cerebrale attraverso una terna trirettangola di assi ortogonali.

Contents

Chapter 1

Introduction

In fields like medicine, psychology, neurology and bioengineering it is important to approach a problem from different perspectives such as treatment, diagnosis, prevention and its comprehension. Different technologies have been developed to maximize knowledge that gives both a global and a local point of view, each obtaining a unique kind of data and gathered under the term *neuroimaging techniques*. Neuroimaging is a branch of imaging science dedicated to invasive and non-invasive in vivo visualization of the structure, function, and metabolism of the central nervous system (CNS), with a particular focus on the brain [1].

Neuroimaging techniques have a rich history that follows advancements in both technology and our understanding of the brain. Early efforts to study brain function were primarily invasive and made on animals; human studies relied on observations in postmortem analyses. The first significant shift toward non-invasive techniques began in the 20th century, marked by the development of electroencephalography (EEG) in 1920s by Hans Berger's studies, which allowed researchers to record electrical activity from the scalp. The advent of computed tomography (CT) in the 1970s represented a leap in visualizing brain structure, followed closely by the development of magnetic resonance imaging (MRI) in the 1980s. These structural imaging methods were soon complemented by functional neuroimaging techniques such as positron emission tomography (PET) and functional magnetic resonance imaging (fMRI), allowing researchers to explore brain activity and connectivity in vivo. Recent decades have seen the emergence of more accessible and portable techniques, such as functional near-infrared spectroscopy (fNIRS), expanding the scope of neuroimaging beyond clinical environments to more dynamic everyday settings. Other recent but more invasive techniques are intracranial EEG or electrocorticography (ECoG)—which requires the scalp to be opened in order to place the elechtrodes directly onto the brain cortex—and functional ultrasound (fUS)—where ultrasounds are emitted through a cranial window. The integration of these techniques has led to the advancement of neuroscience, allowing a more comprehensive understanding of brain structure and function in both health and disease [2, 3].

Neuroimaging is commonly divided into two main categories: structural and functional (fig. 1.1). Structural techniques, such as CT and MRI, focus on the brain's anatomy, while functional techniques assess its activity. EEG and magnetoencephalography (MEG) are examples of functional methods that directly record neural signals, whereas PET, fMRI, and fNIRS infer brain activity indirectly by monitoring brain haemodynamics [4]. The synergy between structural and functional neuroimaging can enhance our understanding of the architecture and activity of the brain. Structural imaging methods, such as MRI, offer detailed views of brain anatomy, allowing the detection of structural abnormalities and insights into brain development. Functional imaging techniques, like fMRI and EEG, capture dynamic brain activity, revealing patterns of neural engagement during cognitive, sensory or motor tasks [5, 6]. Choosing among the neuroimaging techniques described above depends on the type of information needed, particularly in terms of the resolution each method provides (fig. 1.2). Spatial resolution refers to how precisely a technique can localize activity within a specific area of the brain. Temporal resolution, on the other hand, indicates how frequently measurements can be taken over time—a higher temporal resolution means shorter time intervals between images, allowing for more detailed tracking of fast neural dynamics.

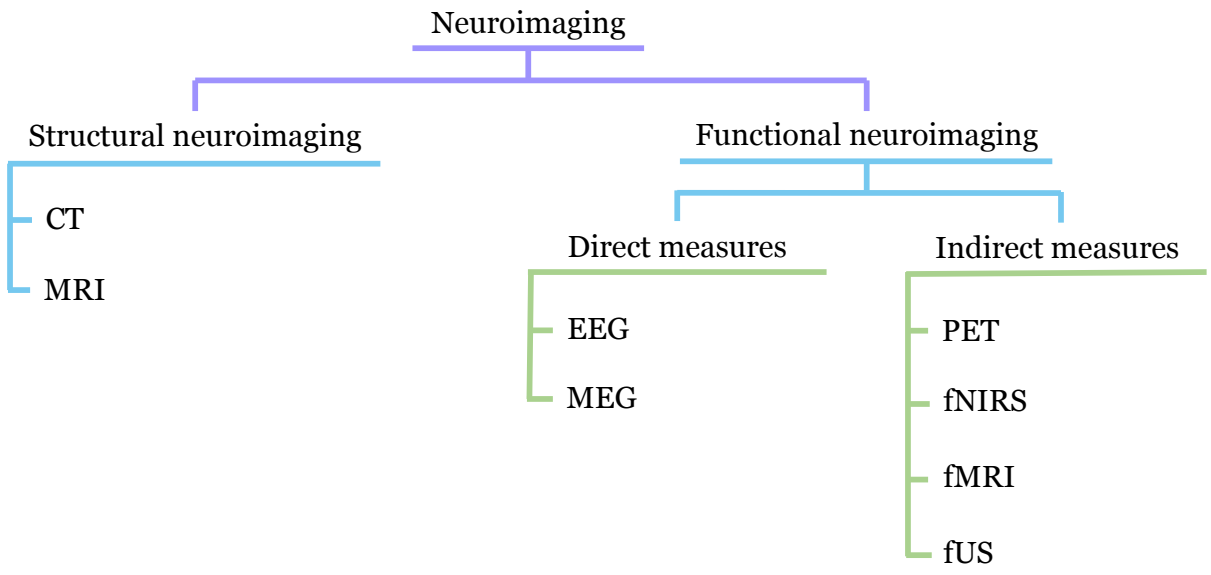

Figure 1.1: **Categorization of neuroimaging modalities.** Taken from [4].

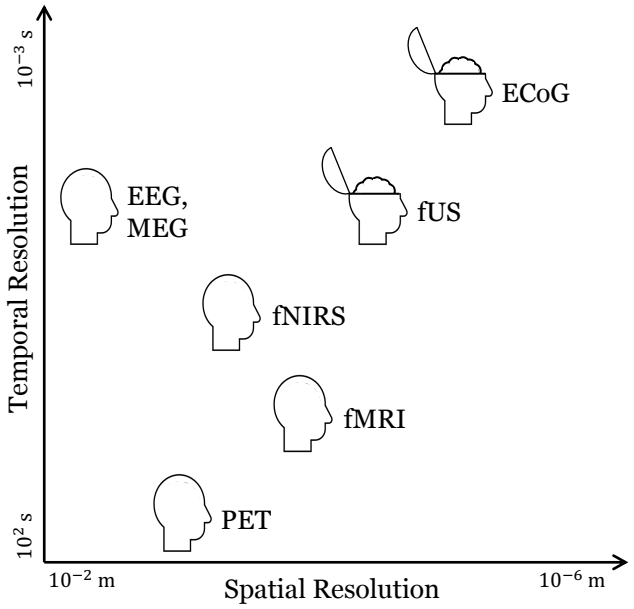

Figure 1.2: **Time-space resolution.** Functional neuroimaging modalities with their temporal and spatial resolutions, where an opened skull refers to invasive imaging. Taken from [4].

1.1 fNIRS Technique

1.1.1 Physical principle and workflow paradigms

Optical methods are used in the assessment of changes in oxygen concentration in blood and tissues, enabling us to quantify the concentration changes of haemoglobin (the molecule responsible for oxygen transport) in both its forms: oxygenated (HbO) and deoxygenated (HbR). Just as cheeks turn red when experiencing strong emotions (e.g., anger), or the face appears bluish while holding one's breath, neural activity also causes oxygenation changes, resulting in changes in absorption coefficient that can be measured with optical methods. The enanched demand of oxygen and glucose due to increased metabolic activity when neurons are firing is known as the heamodinamic response function (HRF), with an increased in blood flow to the working neurons much higher than their actual oxygen consumption.. fNIRS is an optical technique that exploits the properties of red and near-infrared (NIR) light travelling in biological tissues to study cortical haemodinamic activity. Between 650 and 900 nanometers, indeed, bones, skin and water become transparent to NIR light, while HbO and HbR absorb this light differently, since their specific absorption coefficients in this wavelength range are different (fig. 1.3) [7, 8].

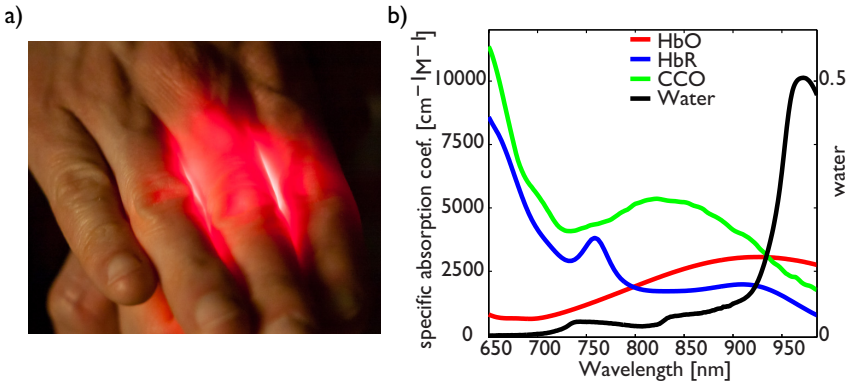

Figure 1.3: **Near-infrared light properties.** a) Demonstration of the transparency of human tissues to red light: covering a white light with a hand, all colours of visible light but red are absorbed by the human tissues. b) Specific absorption spectra of oxy-(HbO) (in red), deoxy-haemoglobin (HbR) (in blue) and cytochrome oxidase (CCO) (in green) and absorption spectrum of water (in black). Between 650 and 900 nm there is an "optical window", through which we can use light to study human tissues. Taken from [7].

Light can interact with a medium in two different ways: scattering and absorption. Light scattering consists of photons moving from a straight pathway into a randomic walk without losing energy; light absorption refers to a reduction in intensity when photons are absorbed by a medium and converted into energy, without modifying their path. In biological tissues, scattering is way more pronounced than absorption. As a result, near-infrared photons in biological tissues display a randomized trajectory and, consequently, a diffuse light field [7].

When NIR light is introduced at the scalp, a relatively predictable quantity of photons pass through tissues returning to the surface of the skin following a banana-shaped trajectory. These photons can be measured at the scalp using photodetectors (see fig. 1.4 and fig. 1.5) [9]. fNIRS requires a source of near-infrared light directed into the scalp, usually via optical fibers, and a detector to measure back-scattered light intensity. Each source and each detector is an optode. A source-detector pair located at a reasonable distance (usually < 45 cm) forms a channel, which measures the changes in light intensity obtained after the light had travelled through the underlying tissues. The separation between source and detector is proportional to the depth reached by the photons travelling from the source to the detector; however, larger separation, although probing deeper regions of the tissues, can compromise the quality of the signal received, since more light will be absorbed: an optimal separation between source and detector must be found in order to maximize the balance between sensitivity to the brain and proportion of photons detected [7].

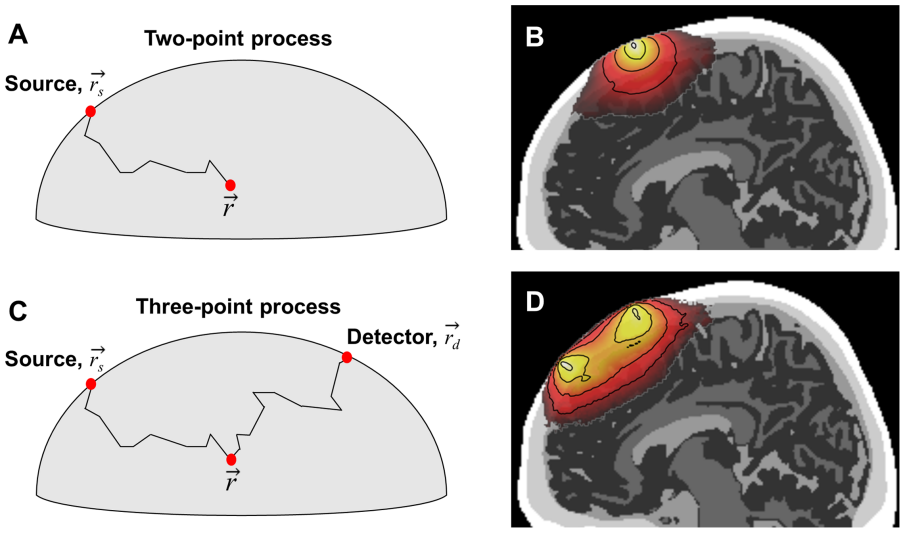

Figure 1.4: **Photon propagation through scattering tissue.** (A) Representation of a single photon moving through tissue, from the source, to an arbitrary point inside the medium. Accumulation of photon weights during this process is the basis of a 2-point Green's function. (B) Example 2-point Green's function, with colours representing the intensity of light reaching any given point in the tissue (truncated after a 5 order-of-magnitude reduction in intensity from peak). (C) Representation of a single photon travelling from the source, to a point in the medium, and on to a detector; the basis of a 3-point sensitivity function. (D) Example 3-point sensitivity function generated from two MC simulations (one for the source, one for the detector) spaced 30 mm apart. Taken from [9].

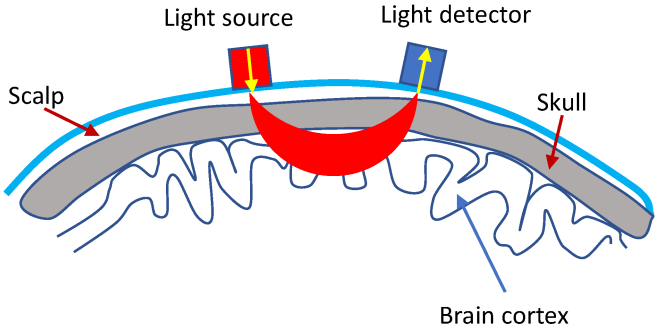

Figure 1.5: **fNIRS schematic demonstration.** Adapted from [10].

1.1.2 Workflow paradigms

Although NIRS methods can measure spontaneous fluctuations in chromophore concentrations, their most common application is in the investigation of functional activation. The typical fNIRS experimental paradigm consists in inducing a state of oxygen demand due to a specific intense neural activation, so an increase of HbR, which is followed by an increase of bloodflow thanks to an homeostatic mechanism called "neurovascular coupling": the increased bloodflow transports HbO and glucose (fig. 1.6). This is called haemodynamic response state. Increased neural activity

causes the release of vasoactive chemicals that induce a dilatation of specific local blood vessels, causing an increase of cerebral blood flow. The enancement of HbO after hyperemia is more significant than the quantity of oxygen used by the neurons to perform the task. As a result of this overcompensated need for oxygenated blood, HRF will show a localized increase in HbO concentration and an associated HbR decrease (fig. 1.7) [7].

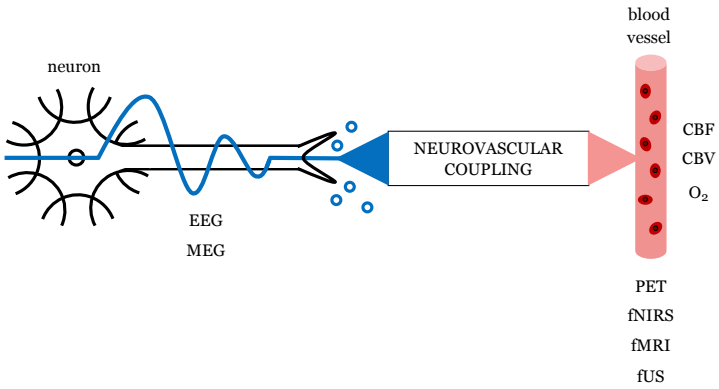

Figure 1.6: **Neurovascular coupling.** Neurovascular coupling describes the relationship between neuronal activity (which takes place using both electrical and chemical signals) and the resulting changes in blood flow. EEG and MEG directly measure neuronal activity whereas PET, fNIRS, fMRI, and fUS provide an indirect measure through neurovascular coupling. Taken from [4].

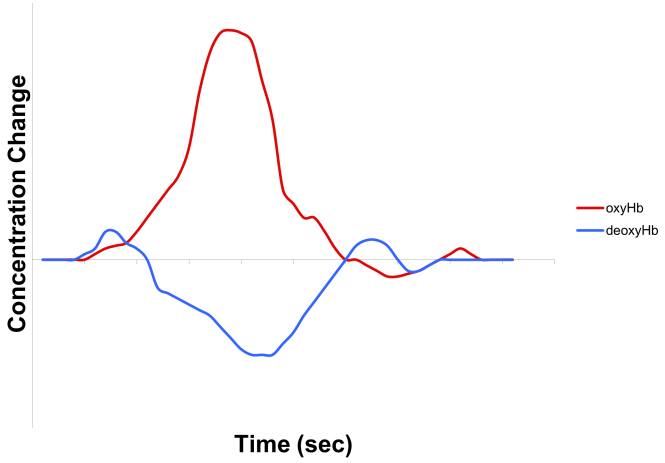

Figure 1.7: **A typical haemodynamic response function (HRF) in adults.** Stimulation is delivered at time zero. The response often starts with a small increase in deoxyHb, followed by an increase in oxyHb and a decrease in deoxyHb concentrations (measured here in arbitrary units). The signals peak several seconds after stimulus onset and then slowly return to baseline, with possible under- and overshoots. Taken from [11].

In fNIRS signal acquisition there are two main paradigms to follow, depending on the target and the patient age: block-design and event-related. In block-design paradigm participants are presented with a train of stimuli, each separated by a brief inter-stimulus interval (ISI). This

approach enhances the haemodynamic response, making it stronger, longer-lasting, and more temporally defined. The HRF typically rises after stimulus onset, stabilizes during sustained stimulation, and returns to baseline once the stimuli stop. Block-designs are used in optical neuroimaging because they yield high-amplitude responses with fewer repetitions, making them especially suitable for populations where short recording sessions are feasible, such as infants or clinical subjects. A typical block-design experiment involves 5 to 15 stimulus blocks, each lasting 5 to 20 seconds. In an event-related paradigm participants undergo many repetitions of a single, short stimulation, separated by some ISI. Many trials (> 40) are required to obtain an accurate HRF because with shorter stimulus duration the SNR will be small, and therefore several repetitions are required to reduce the noise. Event-related paradigms are mainly applied in adult studies when focusing on a specific cognitive process. Choosing between a block or event-related design—and determining the number of repetitions—requires balancing the need for reliable, high-quality signals with the necessity of limiting session duration to prevent participant fatigue or habituation. A key factor in this balance is the ISI: if shorter than 15 seconds, the slow dynamics of the vascular response may lead to overlapping HRFs, complicating interpretation. Designs using shorter ISIs are referred to as fast event-related paradigms. Regardless of the chosen design, jittering the ISI—randomly varying it by a few seconds—is strongly encouraged to prevent physiological rhythms (like respiration or heartbeat) from aligning with the stimulus pattern and distorting the signal. In most fNIRS experiments, the ISI also functions as the baseline or control condition. During this time, participants are generally instructed to remain still, stay quiet, or focus on a neutral visual point. Alternatively, active control tasks may be used—for instance, silent counting—as a control for a verbal fluency task. Careful control selection is vital for ensuring reliable results and reducing false positives caused by motion artifacts or unrelated physiological fluctuations [7]. An example of a session using an event-related paradigm and the HRF of both event-related and block-design paradigms is provided in fig. 1.8 and fig. 1.9.

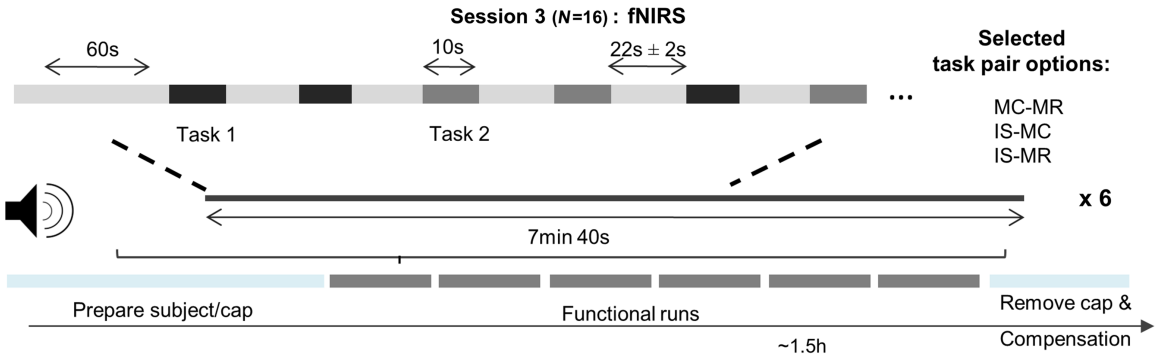

Figure 1.8: **Schematic representation of a functional run during the fNIRS session.** During each mental-task period, participants were acoustically cued to perform one of the two mental-imagery tasks for 10 s while keeping their eyes closed. When participants heard "rest," they were asked to stop the task and await the next instruction. IS, inner-speech; MC, mental-calculation; MR, mental-rotation. Taken from [12].

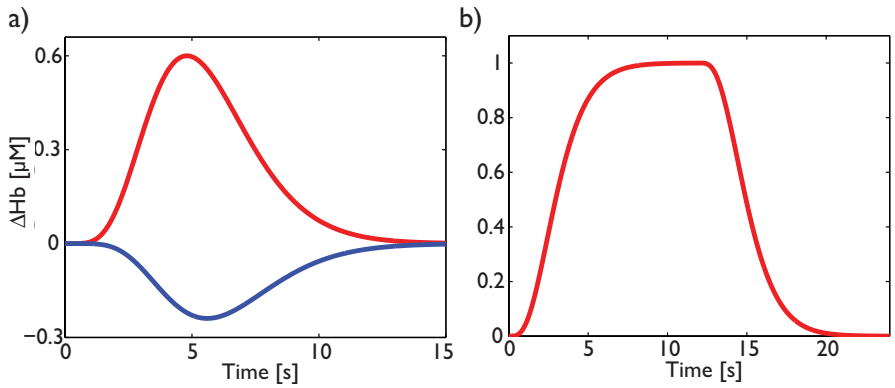

Figure 1.9: **Haemodynamic response.** Examples of simulated haemodynamic response function for a) an event-related paradigm (single event presentation) and b) block-designed paradigm (12 seconds of stimulation). Note the different shape of the HRF in the two figures and how it reaches a plateau in the latter. Taken from [7].

1.1.3 Different fNIRS types

There are three main commercial types of fNIRS systems. Continous wave (CW) systems measure the changes in light intensity using the modified Beer-Lambert law (mBLL):

$$\frac{I}{I_0} = exp^{-xD\mu_a + Losses} \tag{1.1}$$

where I_0 and I are the initial and measured light intensities respectively, x is the separation between source and detectors in each channel, μ_a is the absorption coefficient of the medium and D is the differential pathlength factor (DPF), which accounts for the scattering and extended pathlength in tissues. The factor $Losses$ takes in consideration near-infrared photons lost due to

scattering and the geometry of the scalp. The absorption coefficient is the variable that needs to be determinated, but a direct calculation is impossible due to the number of variables; instead, it can be determined by measuring the change in intensity in two different points in time:

$$\ln \frac{I_2}{I_0} - \ln \frac{I_1}{I_0} = \ln \frac{I_2}{I_1} = xD\Delta\mu_a \tag{1.2}$$

where I_1 and I_2 are the intensities at two separate time points and $\Delta\mu_a$ the variation of the absorption coefficient. Changes in concentrations of absorbing molecules in a tissue are linearly related to the change in the absorption coefficient of that tissue at a given wavelength ($\Delta\mu_{a|\lambda}$) according to:

$$\Delta\mu_{a|\lambda} = \epsilon_{HbO|\lambda}\Delta C_{HbO} + \epsilon_{HbR|\lambda}\Delta C_{HbR} + \cdots \tag{1.3}$$

where $\epsilon_{HbO|\lambda}$ and $\epsilon_{HbR|\lambda}$ are the extinction coefficients of HbO and HbR at wavelength λ, respectively, and ΔC_{HbO} and ΔC_{HbR} the changes in concentration. Using two different wavelengths, one for HbO and one for HbR, yield two different equations, allowing oxygenated and deoxygenated haemoglobin to be estimated from their respective absorbance. The downside of CW-NIRS is that it is unable to calculate absolute absorption coefficients; the presence of DPF and $Losses$ shows that the absorption and the scattering of photons cannot be fully separated, causing uncertainty of the measures. Lastly, the technology is subject to partial volume error (PVE), meaning that the detected signal is a combination of light absorption from both cortical brain tissue (targeted) and extracerebral layers such as the scalp and skull (non-targeted). PVE happens because of the lack of depth sensitivity, uneven sampling (most of the detected light travels through the superficial layers before reaching the brain, so the signal is heavily influenced by the scalp and skull) and the heterogeneous tissue composition [13].

Frequency-domain NIRS (FD-NIRS) systems, also known as frequency-resolved or intensity modulated NIRS system, modulate light source at high frequencies. It measures both the phase shift and the intensity change of the modulated light, allowing it to separately estimate the absorption and scattering coefficients. A modulated laser is required, as well as phasic measurements, to calculate the depth of the light with respect to the systems' incident light; because of this, FD devices are bulkier and more expensive than CW-NIRS system-based [13].

Time-domain NIRS (TD-NIRS) systems, known as time-resolved or time-of-flight NIRS systems, use short light pulses that allow measurement of the time of flight of photons, analyzing the distribution of the amount of time they take to pass through the tissue with single-photon counters, high-speed detectors, and high-speed emitters. This technology offers the most accurate separation between absorption and scattering effects, providing the highest depth sensitivity and information content. The ability to guarantee the best spatial sensitivity and the most accurate

baseline haemoglobin makes TD-NIRS system the most expensive of the three technologies described [13]. NIRS types priciples are illustrated below in fig. 1.10.

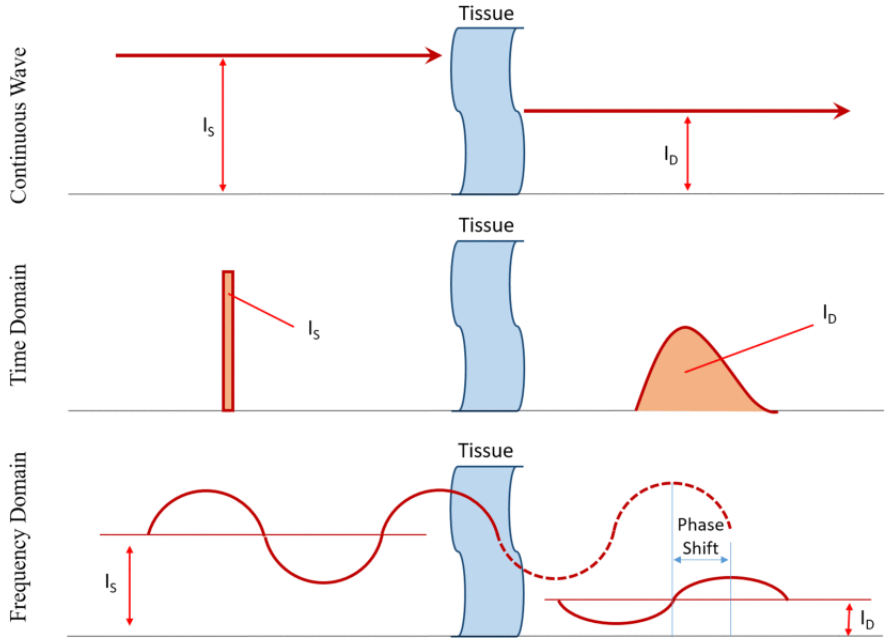

Figure 1.10: **Principles of three different NIRS types.** From top to bottom: CW-NIRS, TD-NIRS, FD-NIRS. Taken from [13].

1.1.4 For whom it is indicated

The fNIRS technique can be employed in any population, but it is particularly suited for vulnerable adults, children and infants. The last category, in particular, has considerably thinner skin and skulls compared to adults. The thickness of these layers is crucial to determine how deeply light can penetrate the cortex, as it must pass through the skin, scalp, cerebrospinal fluid, and other tissues before reaching its target. In infants, light can typically penetrate the cortex to a depth of about 10–15 mm, whereas in adults, the penetration depth is generally limited to 3–5 mm [14]. In addition, infants usually have less hair, which allows for better contact between the scalp and the optodes, reducing light scattering and absorption. These features help minimize the noise and artifacts in the collected signals [7].

1.2 The probe placement problem

1.2.1 Challenges in optode placement

As stated previously, in an fNIRS experiment both the light source emitting photons and the detector receiving them are optodes, and their combination forms a measurement channel. The distance between a source-detector pair, along with the anatomical tissues in between, determines the depth of light penetration and the sensitivity to the underlying cortex [7].

During an experiment, multiple optodes are located on the subject's scalp, usually, using a cap. Unlike EEG, fNIRS lacks a standardized montage because of the different and low number of optodes available in most commercial devices. As a result, designing the optode array is pivotal to the experiment's success, making probe placement a significant challenge that requires careful planning and considerable time investment. Developing optimal optode montages to accurately localize fNIRS signals to specific anatomical regions remains an ongoing challenge in the field [7].

One of the primary limitations of fNIRS is that, although it effectively measures brain activation, it collects data only from the head surface and lacks direct structural information, making it difficult to precisely localize the source of activation on the cortical surface. To spatially interpret fNIRS data, it is necessary to establish a correspondence between the scalp location where the measurement occurs and the cortical region generating the signal. This challenge highlights the importance of careful optode placement and spatial registration techniques to enhance the anatomical accuracy of fNIRS studies [7].

While designing an fNIRS array, these are the factors that should be taken into account:

1. *Source-detector separation*: The distance between the source and the detector directly influences depth sensitivity. As this separation increases, a higher proportion of detected photons will have passed through brain tissue. However, if the separation is too large, insufficient light will reach the detector, leading to unreliable measurements. While the optimal separation depends on the system's dynamic range, 30 mm is commonly used in adult studies, as it strikes a balance between sensitivity to brain tissue and maintaining an adequate signal [7].

2. *Channel density*: Maximizing the number of viable measurement channels within a given scalp area is highly advantageous. A higher channel density increases redundancy, enhancing the robustness of the experiment against potential issues such as poor optical contact with the scalp. Having multiple channels sensitive to the same brain region improves data reliability and ensures better overall signal quality [7].

3. *Channel overlap*: Closely related to channel density, when the target is to perform image

reconstruction of the acquired fNIRS data, one must ensure that measurement channels provide mutually informative data. Changes in haemoglobin concentration within the targeted brain region should be detectable by multiple channels within the array. This means that the Photon Measurement Density Functions (PMDFs), a measure of how much that channel is sensitive to a specific brain area, of different channels should overlap, allowing more accurate spatial localization of hemodynamic activity (see section 1.2.2) [7].

4. *Array coverage*: While increasing channel density and overlap enhances data quality, it should not come at the expense of overall array coverage. The optode arrangement must span a sufficiently large area to account for potential errors in array placement on the individual scalp. Research indicates that placement errors of approximately 1 cm are common, making broad coverage essential to ensure reliable measurements across subjects [7].

5. *Subject comfort*: Ensuring subject comfort is not only an ethical consideration, but also crucial to maintaining data quality. Discomfort can lead to increased movement, which introduces motion artifacts into the recordings. Optical fiber arrays can be heavy, which may limit the number of channels, especially in studies involving infants and children, where excessive weight could be problematic and nonetheless bothersome [7].

1.2.2 Forward problem and sensitivity map

In fNIRS, determining where to place optodes on the scalp to best target specific brain regions requires understanding how light propagates through biological tissue. This is captured by the forward model, which simulates the transport of photons from a light source, through head tissues, to a detector (fig. 1.11). The result is a spatial sensitivity profile—referred to as a PMDF—that quantifies the region of the brain each optode pair is sensitive to. Unlike image reconstruction, which seeks to localize brain activity from the scalp measurements (the inverse problem), probe placement only requires solving the forward model to generate accurate sensitivity maps. These maps are then used to guide the layout of optodes, ensuring optimal spatial coverage of target brain areas.

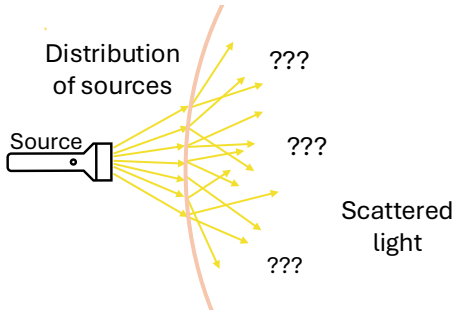

Figure 1.11: **Representation of the forward problem.** Photons traveling through a medium with varying optical properties are scattered by it; the forward problem aims to estimate the most likely path they will follow.

The interaction between light and medium can be mathematically formulated as:

$$y = A(x) \tag{1.4}$$

where y represents the measured data, x describes the spatial distribution of the object's optical properties, and A the forward operator, a function that maps the optical properties of the tissue to the measured signal. Linearizing the equation, this association becomes:

$$\Delta y = J\Delta x \tag{1.5}$$

where Δy represents variations in measured optical parameters (such as intensity changes), Δx corresponds to changes in optical properties of the tissue, and $J = \frac{dA}{dx}$ is the Jacobian matrix. The Jacobian matrix represents the sensitivity of each measurement to changes in optical properties at different locations in the brain, telling how much each detector's reading will change if there is a perturbation of optical properties at a specific location in the tissue. The Jacobin matrix is the output of the forward problem.

The Rytov approximation is commonly used to model light propagation and to solve the forward problem, by expressing the perturbed light fluence (the photon density) as an exponential deviation from the baseline fluence:

$$\varphi = \varphi_0 exp(\varphi_{scat}) \tag{1.6}$$

where φ is the perturbed light fluence, φ_0 is the baseline fluence in an unperturbed medium, and φ_{scat} represents the effect of scattering or absorption changes on the optical field.

More generally, the measured data can be expressed as:

$$y = -\log \frac{\varphi(r_{s,i}, r_{d,i})}{\varphi_0(r_{s,i}, r_{d,i})} \tag{1.7}$$

Here, $r_{s,i}$ and $r_{d,i}$ indicate the positions of the source, s, and the detector, d, respectively, for a given measurement. In eq. (1.5) the elements of Δx correspond to the perturbations in absorption, μ_a, at different spatial locations, denoted as $x_j = \Delta_{a,j}$. The matrix J quantifies how variations in optical properties influence the detected signals and is derived from the photon diffusion equation (see eq. (1.9)).

By applying the Rytov approximation, the sensitivity matrix can be written as:

$$J_{i,j} = \varphi_0(r_s,r_j)\varphi_0(r_j,r_d) \tag{1.8}$$

where $\varphi_0(r_s,r_j)$ represents the fluence at point r_j due to the source s, and $\varphi_0(r_j,r_d)$ accounts for the contribution of light reaching the detector d. These terms are computed by assuming an unperturbed medium.

The matrix J serves as a bridge between scalp measurements and changes in absorption coefficient occurring in the cortex, establishing a mathematical framework that links the observed measurements to the underlying structure of the tissue.

The forward problem is solved for each optode of the array, regardless it being a source or a detector. In order to compute the Jacobian, channel-specific sensitivity maps have to be computed. PMDF is a mathematical function introduced by Arridge and Schweigner in 1995 that provides a model of the probability that a given photon transmitted from the source and measured at the detector has travelled through a given tissue region. PMDF allows to accurately characterize the spatial volume detected by an fNIRS channel. It tells where the photons from a particular source are likely to travel—defining the *banana-shaped* spatial distribution—and which brain regions they will most strongly interact with. The width of the banana-shaped region depends on factors such as tissue scattering and absorption properties. Higher scattering tends to narrow the PMDF, while higher absorption can slightly reduce its width. By leveraging PMDF, the precision and reliability of fNIRS spatial localization can be significantly enhanced. PMDF is obtained applying the Green's function for light diffusion equation, expressed in terms of the photon density or light fluence ($\Phi(\mathbf{r},t)$):

$$\left\{\nabla \cdot \boldsymbol{\kappa}(\mathbf{r})\nabla - \gamma(\mathbf{r}) \ - \frac{\partial}{\partial t}\right\}\boldsymbol{\Phi}(\mathbf{r},t) = -q(\mathbf{r},t) \tag{1.9}$$

where $q(\mathbf{r},t)$ is a light source,$\gamma(\mathbf{r}) = \mu_a c$ and $\kappa(\mathbf{r})$ is given by

$$\kappa(\mathbf{r}) = \frac{c}{3\big[\mu_a(\mathbf{r}) + \mu_s'(\mathbf{r})\big]} \tag{1.10}$$

where $\mu_s' = \big(1 - \overline{f}\big)\mu_s$ is the reduced-scattering coefficient, μ_a and μ_s are the absorption and

scattering coefficients, respectively (in dimensions of inverse length), c is the speed of light, and $\overline{f}$ is an anisotropy factor $(0 \leq \overline{f} \leq 1)$. Resolving the equation yields the following solution:

$$PMDF(\mathbf{r},t) = \int_V G\big(\mathbf{r}, \mathbf{r}^{'},t\big)\rho\big(\mathbf{r}^{'},t\big)dV \tag{1.11}$$

where $PMDF(\mathbf{r}, t)$ is the photon-measurement density function at a point **r** and time t, $G(\mathbf{r}, \mathbf{r'}, t)$ is the Green's function of the diffusion equation, which describes the probability distribution of photons reaching point **r** at time t that originated at point **r'** at time $t = 0$, $\rho(\mathbf{r'}, t)$ is the photon density at point **r'** and time t, and the integral is taken over the volume V of the detector or measurement point [15, 16]. Arridge's interpretation of PMDFs is as follows. Suppose we have an experiment that measures property **M** at a set of source-detectors pairs $(\xi_1,\xi_2,\cdots,\xi_N) \times (\zeta_1,\zeta_2,\cdots,\zeta_S)$, and we divided the domain Ω into L non overlapping elemental regions $\delta V(\mathbf{r}_1^{'}), \delta V(\mathbf{r}_2^{'}), \cdots, \delta V(\mathbf{r}_L^{'})$ such that $\Omega = \cup_{i=1}^{L} \delta V(\mathbf{r}_i)$, then the Jacobian can be discretized:

$$\begin{bmatrix} \Delta M_{\zeta_1}(\xi_1) \\ \Delta M_{\zeta_2}(\xi_2) \\ \vdots \\ \Delta M_{\zeta_i}(\xi_i) \\ \vdots \\ \Delta M_{\zeta_s}(\xi_N) \end{bmatrix} = \begin{bmatrix} J_p^{(M)}(\xi_1,\zeta_1;\mathbf{r}_1') & \cdots & J_p^{(M)}(\xi_1,\zeta_1;\mathbf{r}_j') & \cdots & J_p^{(M)}(\xi_1,\zeta_1;\mathbf{r}_L') \\ J_p^{(M)}(\xi_2,\zeta_1;\mathbf{r}_1') & \cdots & J_p^{(M)}(\xi_2,\zeta_1;\mathbf{r}_j') & \cdots & J_p^{(M)}(\xi_2,\zeta_1;\mathbf{r}_L') \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ J_p^{(M)}(\xi_k,\zeta_i;\mathbf{r}_1') & \cdots & J_p^{(M)}(\xi_k,\zeta_i;\mathbf{r}_j') & \cdots & J_p^{(M)}(\xi_k,\zeta_i;\mathbf{r}_L') \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ J_p^{(M)}(\xi_N,\zeta_S;\mathbf{r}_1') & \cdots & J_p^{(M)}(\xi_N,\zeta_S;\mathbf{r}_j') & \cdots & J_p^{(M)}(\xi_N,\zeta_S;\mathbf{r}_L') \end{bmatrix} \begin{bmatrix} \Delta\mathbf{p}(\mathbf{r}_1^{'}) \\ \vdots \\ \Delta\mathbf{p}(\mathbf{r}_j^{'}) \\ \vdots \\ \Delta\mathbf{p}(\mathbf{r}_L^{'}) \end{bmatrix} \tag{1.12}$$

The PMDFs are the rows of this matrix. The columns represent the perturbations in the data that are due to a point inhomogeneity [15]. In this definition $\Delta p(r)$ corresponds to Δx in eq. (1.5). The magnitude of the entries tells us how much a particular voxel contributes to the measurement at each detector.

The interpretation of PMDFs can indeed be nuanced. In Arridge's 1995 work, it is emphasized that PMDFs should ideally not overlap when interpreting data for optical tomography. This is because overlapping PMDFs can complicate the inverse problem, making it harder to accurately reconstruct the internal optical properties of the tissue being imaged [16]. However, in practical applications, some degree of overlap is often unavoidable and can even be beneficial for improving spatial localization. Overlapping PMDFs can enhance the sensitivity and robustness of measurements by providing more comprehensive coverage of the region of interest [17]. Changes in NIRS photon sensitivity caused by different source-detector distances are shown in fig. 1.12.

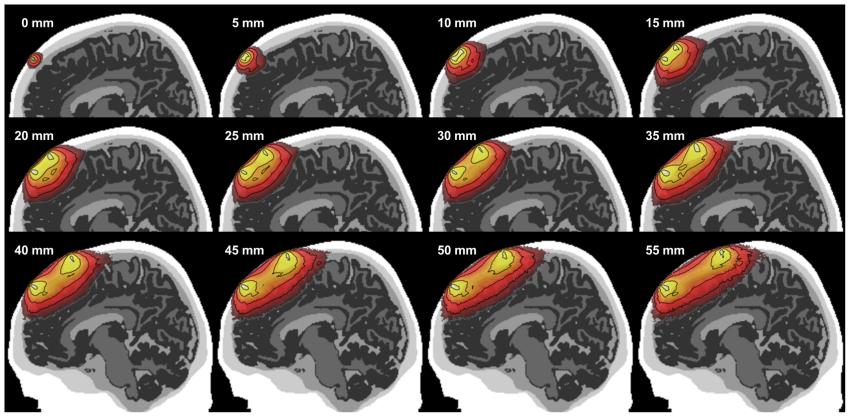

Figure 1.12: **Photon sensitivity profile at a broad range of source-detector separations.** Contours are drawn for each order of magnitude loss in sensitivity from peak and are truncated after 5 orders of magnitude. Taken from [9].

Once sensitivity maps have been computed, the next step is to interpret them in terms of brain anatomy. To do this, the sensitivity maps must be registered to a standardized brain space, such as the Montreal Neurological Institute (MNI) coordinate system. This process allows for comparison across participants and studies and ensures that the measurements are anatomically meaningful. It also helps in targeting specific functional brain areas, which is crucial for interpreting fNIRS data in the context of cognitive tasks or clinical assessments [18].

1.2.3 The MNI Coordinate System and atlas for standardized normalization in neuroimaging

To enable accurate and comparable probe placement across subjects in fNIRS, it is essential to use a standardized 3D coordinate system. In this context, the Montreal Neurological Institute (MNI) coordinate space allows researchers to map optode locations onto a common brain template, supporting inter-subject alignment and group-level analysis. In neuroimaging studies, functional data must be integrated across subjects to generate meaningful inferences at the population level. While single-subject functional images can provide insights for individual case studies or diagnoses, many studies aim to generalize findings across a group. This requires structural normalization, as functional data are inherently tied to individual anatomical differences. The challenge arises from the natural variability in brain structures—each person's cortical anatomy differs in size, shape, and sulcal organization, making direct comparisons between subjects difficult [18].

One of the earliest approaches to standardizing brain structure was introduced with the Talairach atlas. This system was based on the brain of a single elderly Caucasian woman and introduced three key innovations: a coordinate system to identify brain locations relative to

anatomical landmarks, a spatial transformation to align different brains, and an anatomical atlas for standard reference. The Talairach system relies on the anterior commissure (AC) and posterior commissure (PC) as invariant landmarks, defining a three-dimensional coordinate system with the AC as the origin (fig. 1.13 and fig. 1.14). Although widely adopted, this approach has limitations, primarily because it is based on a single brain, making it a biased representation of general neuroanatomy [19].

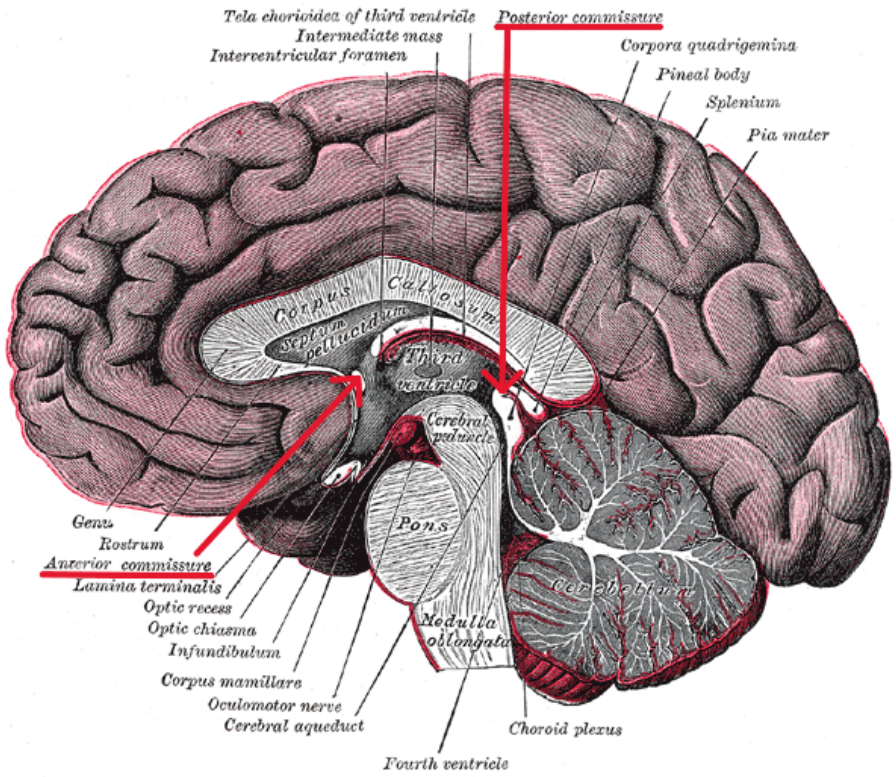

Figure 1.13: **The AC and PC landmarks in human brain.** Adapted from [20].

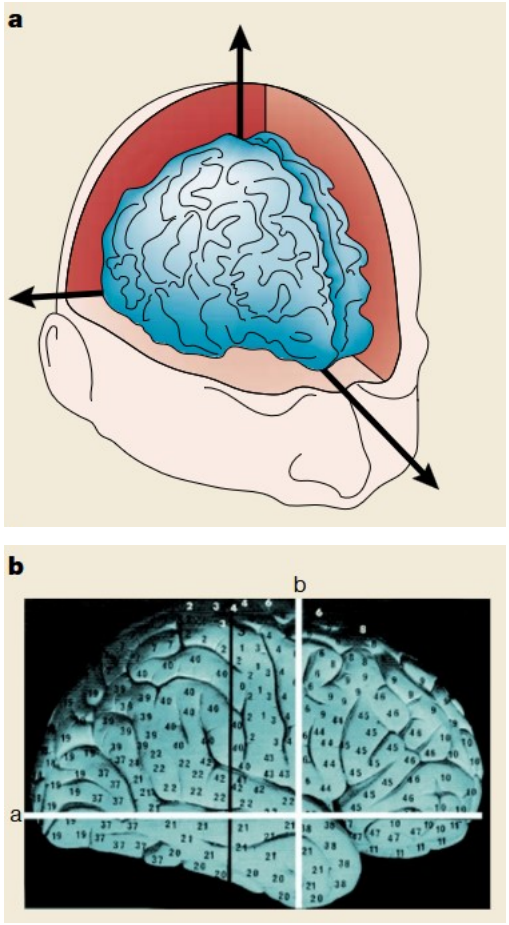

Figure 1.14: **The Talairach and Tournoux atlas.** a) Illustration of the Talairach coordinate system, where the AC is designated as the origin. The brain is rotated to align with the AC–PC line, which is placed on a horizontal plane. b) three orthogonal axes that define the Talairach space: y-axis (AC–PC line), z-axis (vertical through AC and interhemispheric fissure), and x-axis (perpendicular to y and z, through AC). Adapted from [19].

To address these limitations, the MNI developed a new standardized brain template derived from multiple MRI scans. In the early 1990s, Evans and colleagues introduced the concept of a statistical MRI atlas to overcome the shortcomings of single-subject templates [21, 22]. This effort led to the creation of the MNI305 atlas, constructed through a two-step process. First, anatomical landmarks adapted from the Talairach and Tournoux atlas were manually identified on T1-weighted MRI scans of 305 healthy young adults (239 males, 66 females, age 23.4 ± 4.1 years). These landmarks were aligned using least squares linear regression to match the AC-PC line of the original Talairach and Tournoux brain, producing an initial average volume approximating Talairach space. In the second step, each individual brain was automatically registered on this average using a whole-brain (9-parameter) linear transformation [23]. Unlike Talairach's piecewise method, this approach reduced manual errors and yielded a sharper average. The final MNI305 template approximates the Talairach space, though with a slight offset—particularly in

the Z-axis (approximately +3.5 mm) (fig. 1.15). Furthermore, residual non-linear anatomical differences between subjects introduced a "virtual convolution" effect, slightly enlarging the resulting template compared to most individuals [24]. This template ultimately defined what is now called the MNI coordinate system space (fig. 1.16).

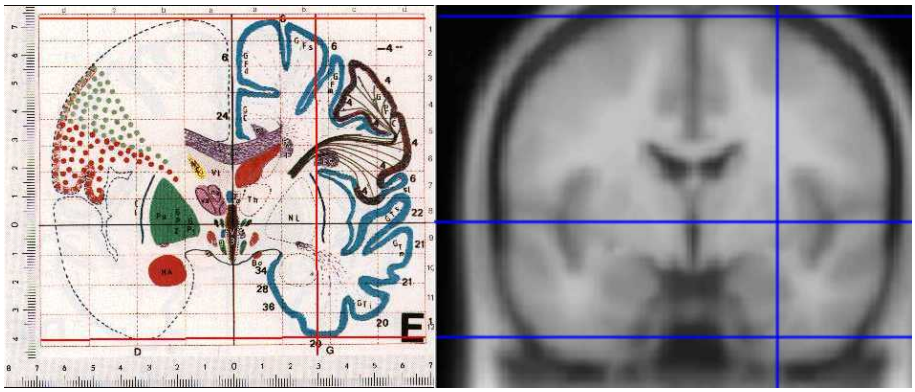

Figure 1.15: **Comparison between Talairach atlas and MNI 152 atlas.** Coronal brain slice of the section of the Talairach atlas (left), next to the equivalent section of the MNI152 average brain section from the SPM distribution (right). The section is at y = -4; there is a vertical line at 30 mm in x, and horizontal lines at z = 0, 73 and -41, the AC top and bottom of the Talairach brain. Taken from [25].

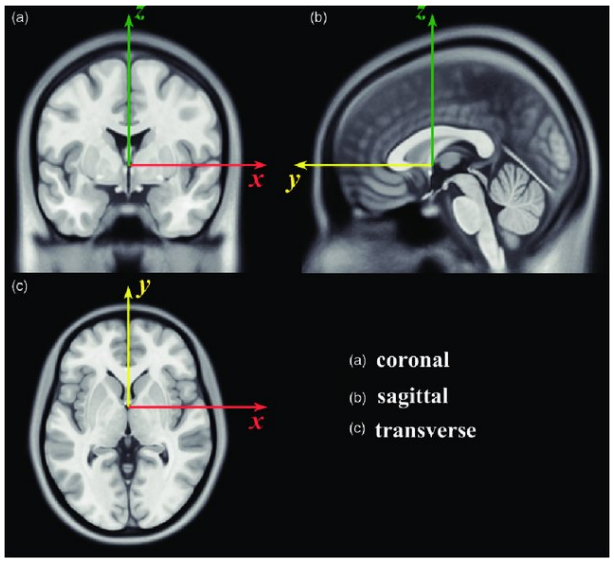

Figure 1.16: **The MNI coordinate system of the brain.** Taken from [26].

The MNI305 was later refined into the ICBM152 template—where ICBM stands for International Consortium for Brain Mapping —, now widely adopted in neuroimaging, which preserves major sulcal landmarks while maintaining a representative brain structure for general population-level studies [18]. Although inspired by Talairach principles, MNI templates diverge in shape and size, especially in regions like the temporal lobes, which are about 10 mm lower in

the MNI space [19]. These distinctions are particularly relevant in fNIRS, where precise spatial localization is essential. Since fNIRS lacks structural information, researchers rely on standard templates such as MNI152 or Colin27—template in the MNI space developed in 1998 and widely used for its anatomical sharpness [27]—to construct head models, calculate PMDFs and decide probe placement for each experiment. These templates help in optode placement, providing relative positions of landmarks, such as the EEG 10-20 standardized system or its higher density versions, which can be used to define optode positions and translate those positions in the physical world. A comparison between templates is provided in fig. 1.17.

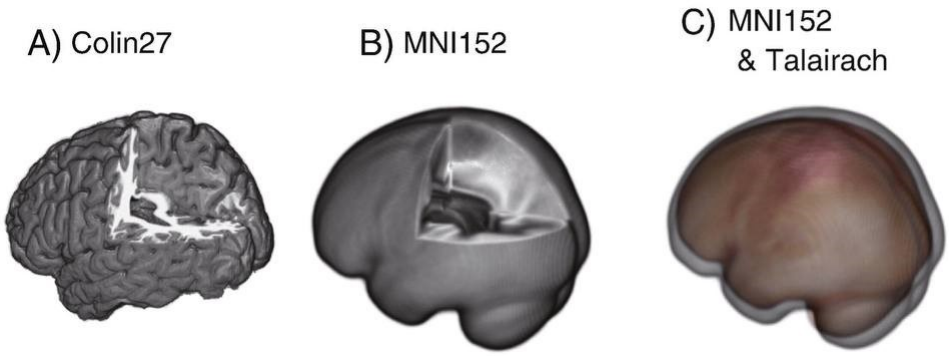

Figure 1.17: **Comparison among standard brains.** All standard brains are aligned in and viewed from the same angle. (A) Colin27 with macro-anatomical information. (B) MNI152 template with smoothed cortical structures due to averaging. (C) MNI and Talairach templates aligned in the same coordinate system. Although they have similar shapes, the Talairach template is slightly smaller. Adapted from [18].

1.2.4 *Array Designer*

The fNIRS community has developed several approaches to optimize optode-layout designs using light-sensitivity profiles in order to help researchers spend less time on the design part itself. Some of them require a manual and subjective input from the user (like AtlasViewer and fNIRS Optodes' Location Designer (fOLD), whilst few of them are more objective and automated—such as *Array Designer*, which is going to be explained in detail in this section [28].

Regardless the approach, the first step a researcher should pursue to create the array template, is to define the regions of interest (ROIs) (fig. 1.18), since, due to the limited number of available optodes, the montage will target only those ROIs. ROI selection is critical in neuroimaging studies, as it determines which cortical areas are analyzed for functional activity. Approaches to define ROIs can be distincted in two methodologies: the first is using MNI coordinates based on published papers or old personal studies. The second is using parcellated brain atlases, like Broadmann Areas, AAL and Cerebrum Atlas (CerebrA) [29]. Broadmann Areas is based from hystological structure and AAL comes from a a single subject's MRI segmented into 116

anatomical regions (90 cortical and 26 cerebellar). The CerebrA parcellation is an anatomical brain atlas that provides detailed labeling of cortical and subcortical regions on the MNI-ICBM152 2009c symmetric brain template (fig. 1.19) [30].

Researchers typically define an ROI based on their research question and align optode placement to maximize sensitivity within the target region. The simplest and most common optode layout design approach consists in assigning source and detector locations on the head subjectively to cover a given cortical ROI according to the standardized 10-20 EEG system or its extended versions. Standardized coordinate systems, such as MNI space, facilitate the process by providing a common anatomical reference, ensuring meaningful comparisons across studies and populations. While population-level parcellated brain templates, such as CerebrA, offer standardized ROIs in this space, subject-specific atlases derived from individual fMRI data can also be registered to MNI space, allowing for more precise localization of optodes arrays tailored to each participant's anatomy [12].

A more objective and automatic approach is instead *Array Designer*, a MATLAB toolbox that can be used to design optimal arrays exploiting the channel sensitivity profiles given a certain amount of sources (nS) and detectors (nD), either on a standard head model or on subject-specific anatomy. The toolbox consists in two separate graphical user interfaces (GUIs): ArrayDesignerAPP.mlapp (ADA, fig. 1.21) and ROIBuilderAPP.mlapp (ROIBA, fig. 1.22). In ADA users can customize parameters such as the number of optodes and the minimum and maximum distances between sources and detectors. In ROIBA ROIs can be defined starting from two different atlases that can be chosen—at the state-of-the-art the MNI152 symmetric and the MNI152 asymmetric atlas. If the symmetric atlas is selected, users can build an ROI manually by drawing it or by choosing from the CerebrA parcellation the interested areas; whereas, if the asymmetric atlas is picked, users can only draw it. The process of manually choosing the interested points on the atlas happens via a point-and-click interface. If needed, an ROI can be mirrored thanks to a "Mirror ROI" button. Some examples of ROIs that can be generated by App Designer are provided in fig. 1.23. Based on these inputs, *Array Designer* computes an optimal optode layout to target the specified ROIs. Each optode is labelled using the 10–5 or 10–2.5 coordinate system, and the tool provides detailed output, including total sensitivity (in millimeters), percentage of ROI coverage, and source-detector distance statistics (average, minimum, and maximum) [31].

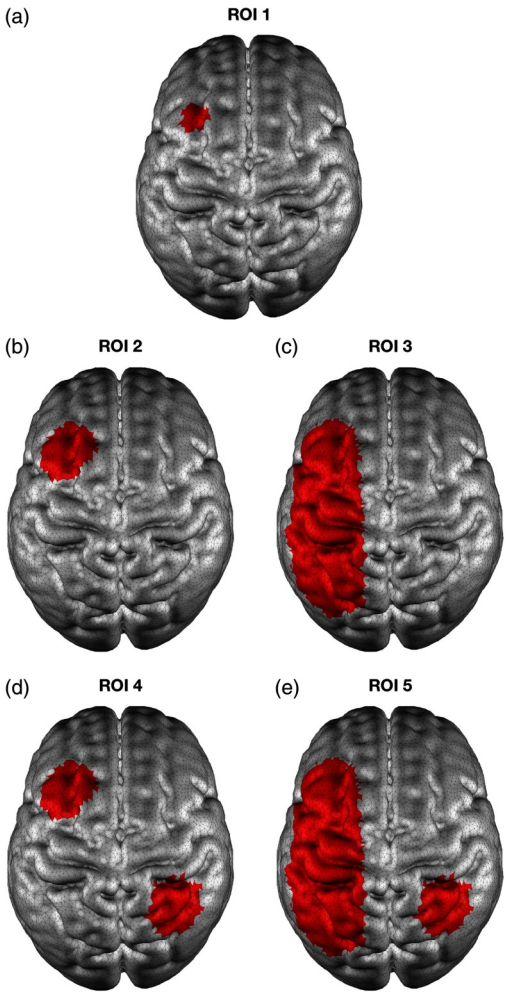

Figure 1.18: **Five example of ROIs.** (a) ROI 1; (b) ROI 2; (c) ROI 3; (d) ROI 4; and (e) ROI 5. Taken from [31].

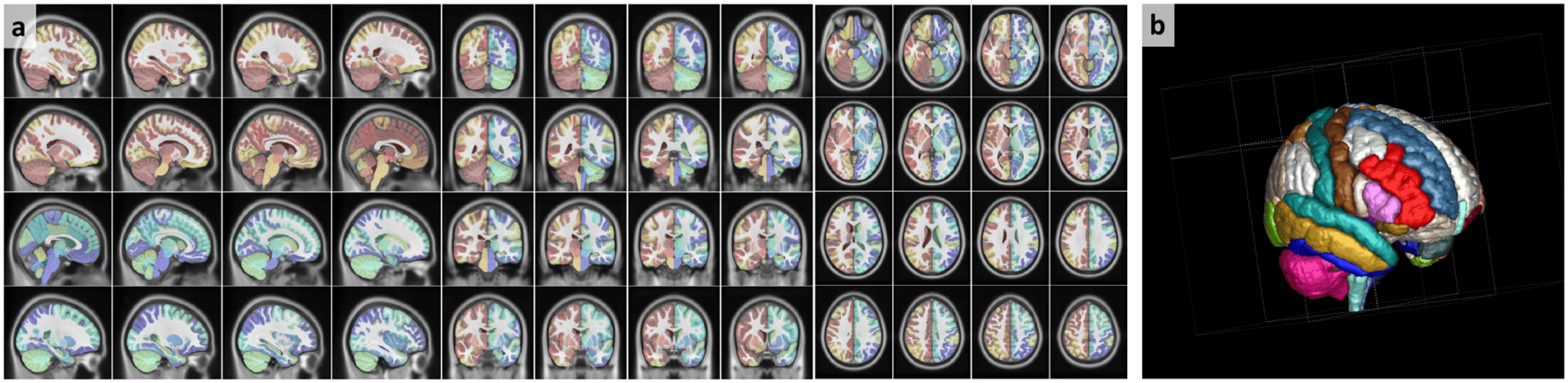

Figure 1.19: **Cerebrum Atlas.** Warped CerebrA atlas overlaid on the ICBM152 non-linear 2009 symmetric average MRI template. Adapted from [30].

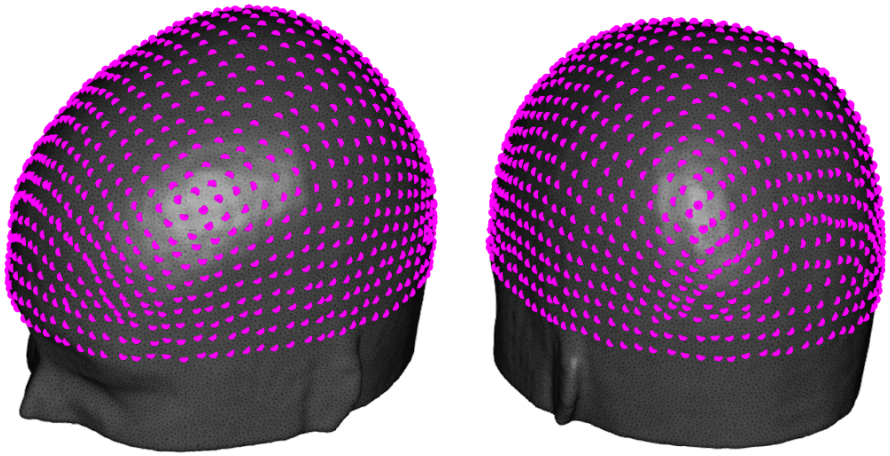

Figure 1.20: **10-2.5 system.** In magenta, all points of the 10-2.5 system, the solution space, are visualized over the adult MNI152 scalp surface mesh. Taken from [31].

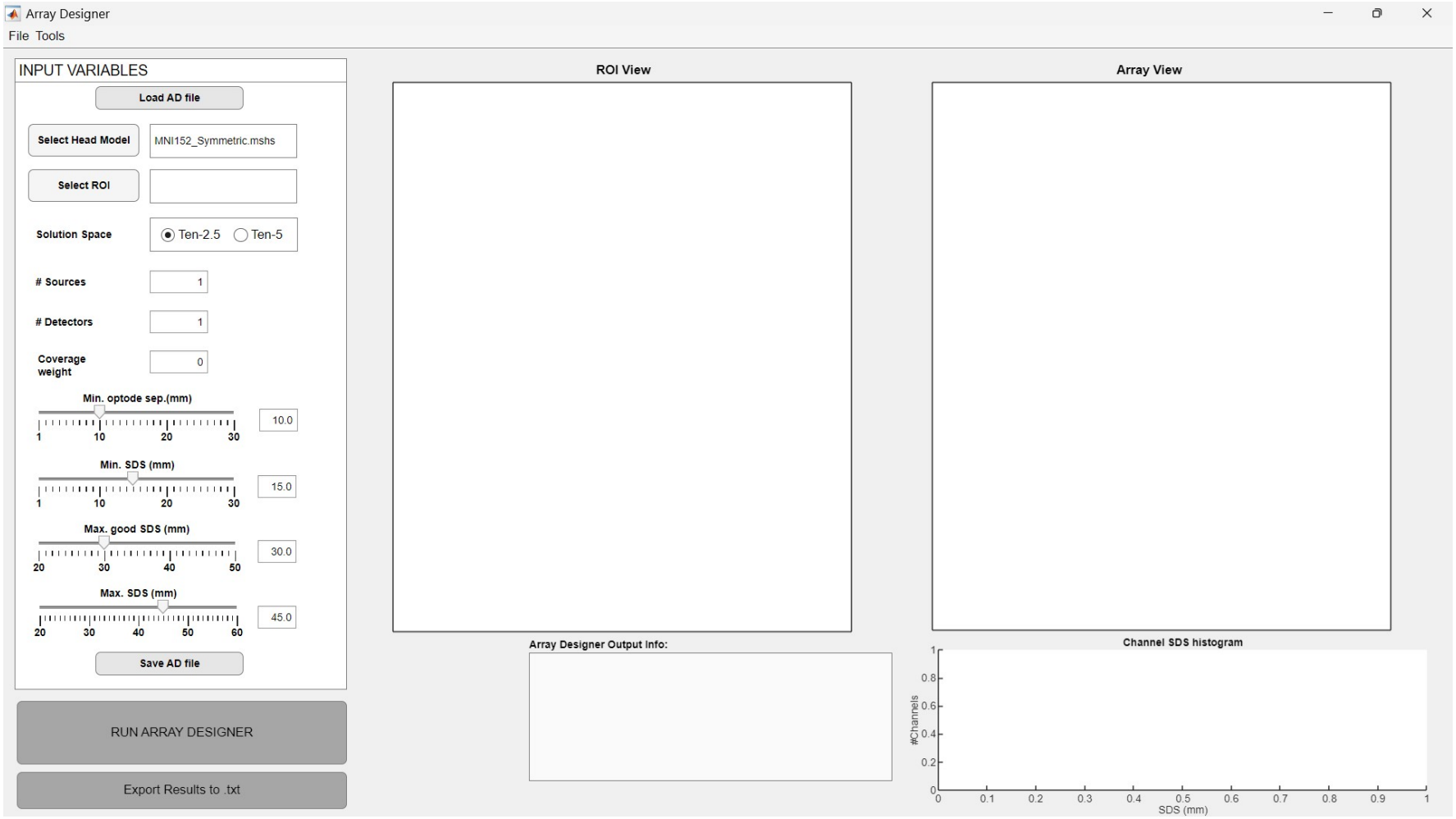

Figure 1.21: ***Array Designer*** **User Interface.**

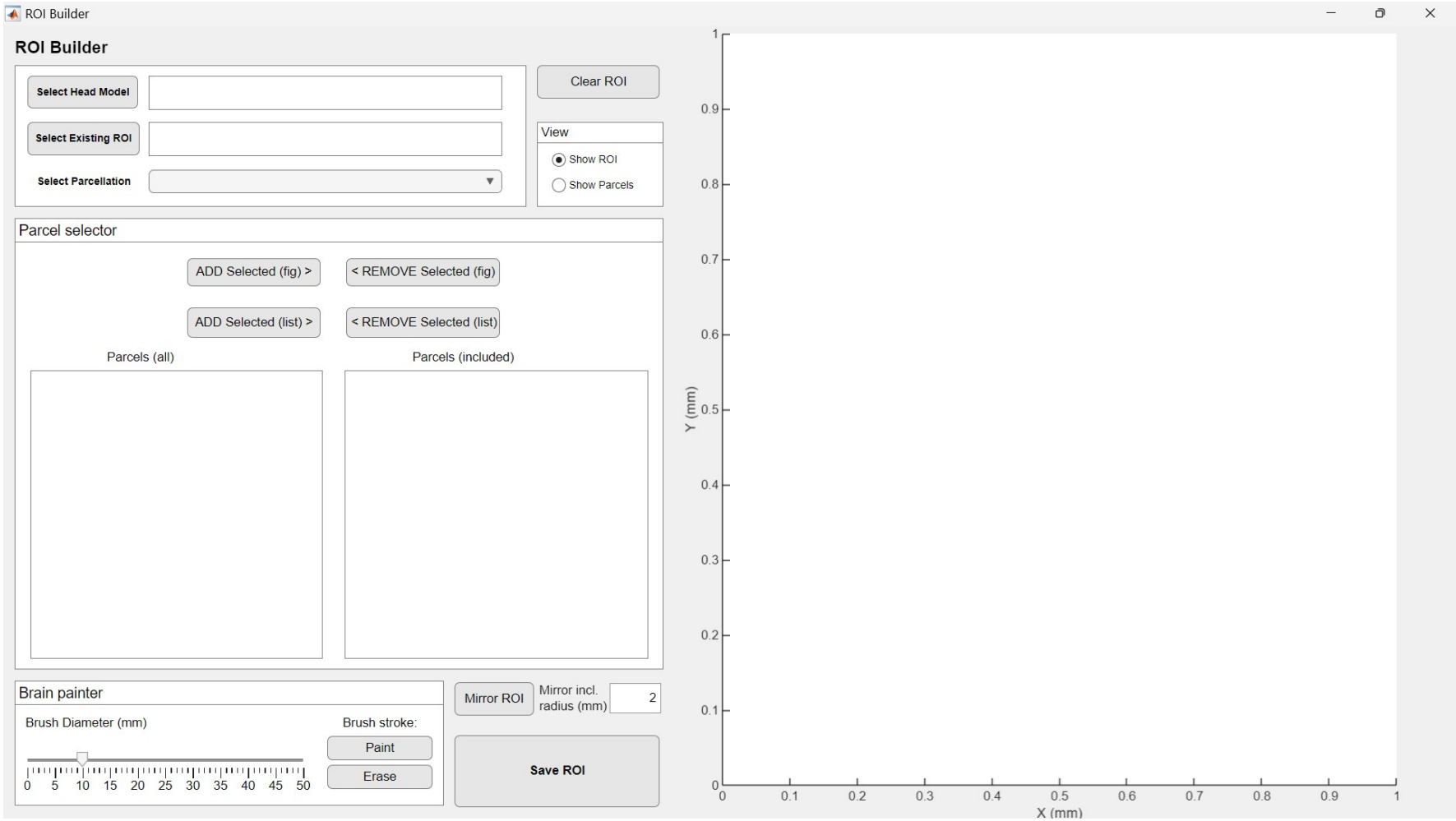

Figure 1.22: ***ROI Builder*** **User Interface.**

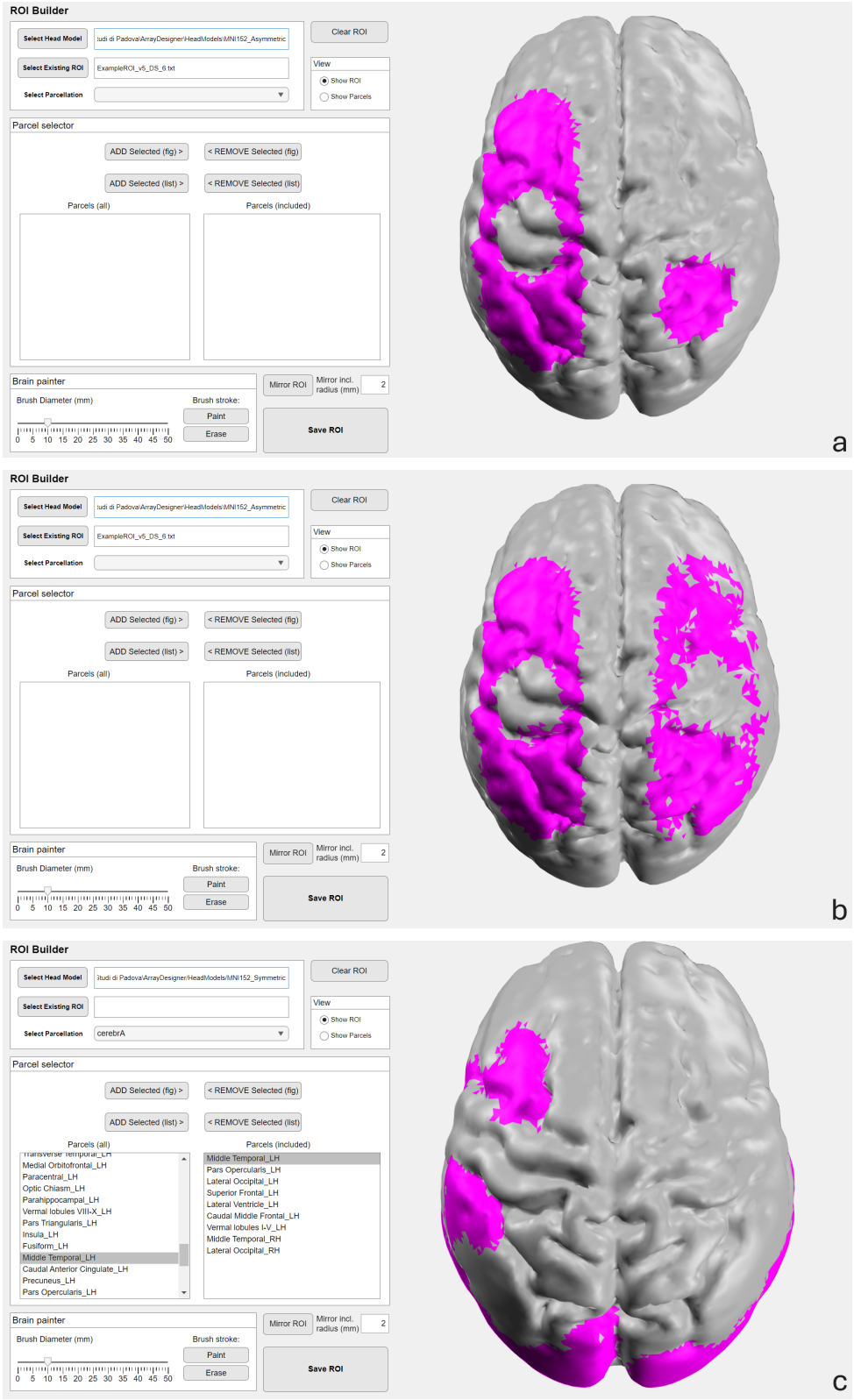

Figure 1.23: **Three visualizations of *ROI Builder*.** a) Upload of a pre-existing ROI; b) ROI mirrored using ”Mirror ROI” button; c) ROI built from the CerebrA parcellation of the MNI152 asymmetric atlas.

1.3 Purpose of the work

At its current state, *ROI Builder App* whitin *Array Designer* presents several limitations—most notably, the lack of functionality to create ROIs directly from MNI coordinates reported in the literature. To address this, the present work focused on implementing a new feature that allows users to generate ROIs from MNI coordinates using the MNI152 asymmetric atlas. The tool was further enhanced to support importing coordinate lists from external files, streamlining the ROI creation process. The aim of this project is to provide documentation and evaluation of the development process and improvements made to the *Array Designer* application within the MATLAB environment. The work begins with an overview of the development framework, detailing the tool view, GUI components, and the architecture of callback functions that govern the application's logic (see chapter 2).

Following this, the core of the thesis is dedicated to the implementation of new features and enhancements. These include the integration of a drop-down menu for MNI coordinate selection, an interactive system to save coordinates, improvements to the existing codebase, and refinements to the user interface design (UI design). Each improvement is supported by a comparative analysis with the original code where applicable (see chapter 3).

The project concludes with a summary of the results achieved, alongside a discussion of the application's reliability and effectiveness. Potential future developments are also outlined, such as the integration of a new anatomical atlas or the introduction of alternative brain parcellation schemes (see chapter 4).

The appendices provide additional resources, including the MATLAB code and the dataset used during development.

In general, this work serves as both a technical report and a roadmap for further evolution of the tool.

Chapter 2

MATLAB App Designer

2.1 Graphical user interfaces

The most simple way to create GUIs in MATLAB is building them in a `.m` file using handle classes. These follow a hierarchical structure: a parent object (typically a figure window) contains various nested components, such as pushbuttons, text boxes, and sliders. Parent interface objects can be a `figure`, `uipanel`, or `uibuttongroup`. The `figure` function creates the main GUI window. A `uipanel` groups interface elements together, while a `uibuttongroup` is designed to contain buttons—such as radio or toggle buttons. At the top of the GUI hierarchy is the *Figure Window*, which acts as the container for all user interface components. The typical procedure begins by creating this figure using the `figure` function and storing its handle in a variable, which allows users to modify its properties later. During the initial setup phase, it is common to make the figure invisible (`'Visible', 'off'`), add and configure all UI elements, and finally set it to visible once complete. If no other figures are open, the one created becomes the current figure, accessible using the `gcf` function. The parent of a figure is the screen itself, which can be accessed via `f.Parent` or the root object `groot`. Most GUI elements are created using the `uicontrol` function. This function takes property-value pairs to define characteristics of the control. One of the key properties is `'Style'`, which determines the component type (e.g. `'text'`, `'edit'`, `'pushbutton'`). By default, `uicontrol` creates objects in the current figure, but a specific parent can be specified as `uicontrol(parent,...)`. An example of a GUI created using a `.m` file (code 2.1) in MATLAB is reported in fig. 2.1.

Code 2.1: Code for generating a GUI with a static text box.

```
function exampleGUI
% exampleGUI creates a GUI with just a static text box.
% To run the function: exampleGUI or exampleGUI()

% Create the GUI but keep it invisible during initialization
f = figure('Visible', 'off', ...
    'Color', 'white', ...
    'Units', 'Normalized', ...
    'Position', [.25, .5, .35, .3]);

htext = uicontrol('Style', 'text', ...
    'Units', 'Normalized', ...
    'Position', [.5, .5, .5, .5], ...
    'String', 'Hello, World!');

% Add a name to the window and center it on the screen
f.Name = 'Example GUI';
movegui(f, 'center');

% Make the GUI visible
f.Visible = 'on';
end
```

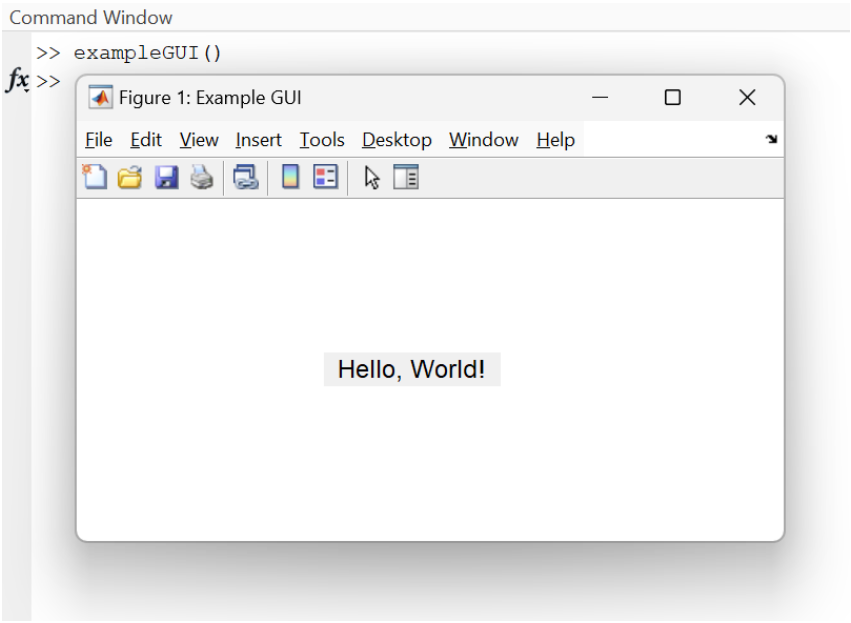

Figure 2.1: MATLAB GUI with static text evoked from function `exampleGUI.m` in Command Window.

In this example, `f` is the variable through which the figure's properties can be accessed. The figure acts as a white canvas, and the option `'Units','Normalized'` allows the GUI to scale proportionally when the window is resized. The vector `[.25, .5, .35, .3]` specifies the position and size of the window: one quarter from the left, halfway up from the bottom, and with width and height of 35% and 30% of the screen, respectively.

The variable `htext` defines a simple static text box using the `uicontrol` function. Its properties specify the layout and the string to display—in this case, the classic `Hello, World!` commonly used when learning a new programming language.

2.2 Tool view

Instead of using `.m` file MATLAB's App Designer allows users to build GUIs by combining a visual layout environment with object-oriented programming (OOP). Each app created in App Designer is implemented as a class derived from the *matlab.apps.AppBase* superclass. To open the App Designer environment, users can either type `appdesigner` in the Command Window or select "App" from the "New" button in the MATLAB toolstrip. Upon launching, the app opens in *Design View*, presenting a blank layout ready for editing as displayed in fig. 2.2.

The interface is divided into three main sections: on the left is the Component Library (fig. 2.3), which provides a wide range of UI elements such as buttons, sliders, text areas, and axes for image display; in the centre lies the Canvas, where these components can be visually arranged; and on the right is a panel that includes the *Component Browser* and the *Property Inspector*. The *Component Browser* displays all added elements in a parent-child hierarchy, while the *Property Inspector* allows users to customize each component's attributes—including name, font, size, position, and interactivity. The top of the interface contains a toolstrip that enables users to create a new app from scratch, open an existing one, save projects, describe them, or share them.

When components are added to the canvas, App Designer automatically generates the corresponding code in the background. Each added component is associated with a new property in the app class, and its default settings are initialized automatically. Users can modify these properties either through the graphical *Property Inspector* or directly by editing the code in *Code View*.

Switching to *Code View* reveals the underlying class definition of the app. By default, the class is named (`App1.mlapp`) and inherits from `matlab.apps.AppBase`. The `.mlapp` extention is short of "MATLAB App". This class includes several key sections: a constructor method (`App1`) that initializes the app and creates its layout, a destructor method (`delete`) that ensures the app closes properly, a private methods block for internal logic, and a public methods block

for functions accessible to the user—such as callback functions that define interactive behaviour. In this view, the left panel changes to show the *Code Browser*, which lists all functions and properties, and an *App Layout Preview*, providing a visual summary of the interface. Notably, App Designer generates a section of gray-shaded code that cannot be manually edited; this portion ensures the structural integrity of the app and is automatically managed.

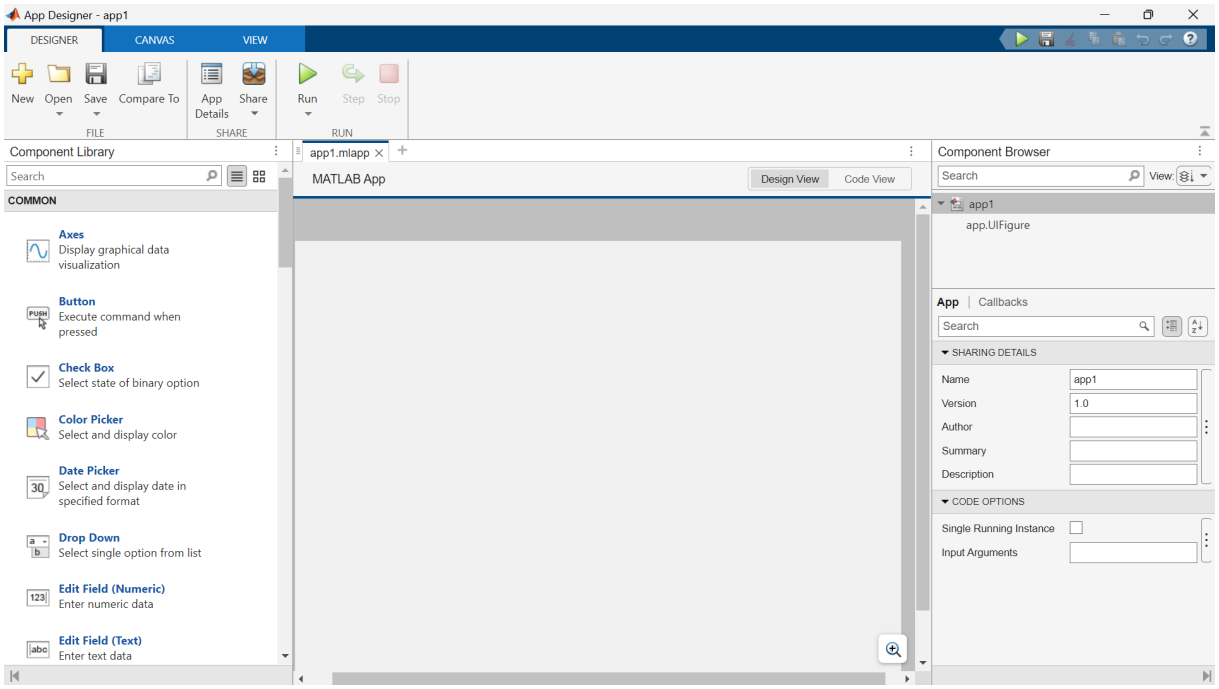

Figure 2.2: App Designer view.

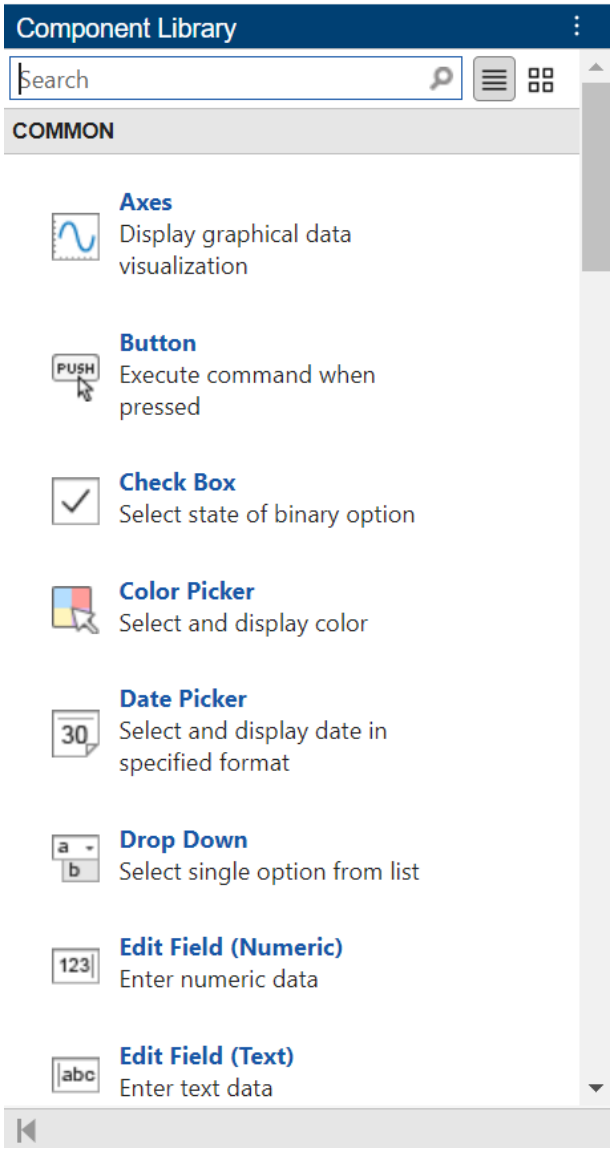

Figure 2.3: Component library section.

In App Designer, a similar GUI can be created with code 2.2 and will appear as in fig. 2.4.

Code 2.2: App Designer Code View of a GUI with a static text box.

```
classdef trial < matlab.apps.AppBase
    % Properties that correspond to app components
    properties (Access = public)
        UIFigure          matlab.ui.Figure
        HelloWorldLabel   matlab.ui.control.Label
    end

    % Component initialization
    methods (Access = private)
```

```
        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 640 480];
            app.UIFigure.Name = 'MATLAB App';

            % Create HelloWorldLabel
            app.HelloWorldLabel = uilabel(app.UIFigure);
            app.HelloWorldLabel.FontSize = 48;
            app.HelloWorldLabel.Position = [180 210 282 63];
            app.HelloWorldLabel.Text = 'Hello, World!';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = trial

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

Figure 2.4: App Designer GUI with static text.

To run the app, click the green *Run* arrow in App Designer. A dialog will appear, asking for the file name. The default name is `App1.mlapp`.

Once a GUI is created, enabling user interaction requires the implementation of *callback functions* (see section 2.3).

2.3 Callback functions

When the user performs an action in the GUI —known as an *event*— a corresponding response is triggered through a *callback function*. This programming paradigm is called *event-driven programming*. A callback function must be accessible on the MATLAB path. One way to ensure this is to define it as a nested function within the main GUI function. When a callback is nested, it has access to the parent function's variables via handles. A GUI can have multiple callback functions, each responding to different events.

In App Designer, callback functions are similarly used to define how the application reacts to user interactions. A typical callback function has the form `function callbackFcnName(src, event)`. Here, `src` refers to the UI component (the source of the event), and `event` is a structure containing information specific to the triggered event. App Designer does not automatically create callback function definitions. To define one, right-click the component in *Design View*, then select *Callbacks* to choose from a list of available callback types for that component. An app can contain multiple components, each with its own set of properties. To access a specific property of a component, use dot notation in the form `app.Component.Property`.

Below is a list of common callback functions in App Designer:

1. `'StartupFcn'`: Runs once when the app starts.
2. `'ClickedFcn'`: Executes when a mouse click is detected.
3. `'ButtonPushedFcn'`: Executes when a button is clicked.
4. `'ValueChangingFcn'`: Executes continuously as the component value changes live (e.g., dragging a slider or typing in a field).
5. `'ValueChangedFcn'`: Executes when the value of a component changes and the user confirms it (e.g., pressing Enter or clicking elsewhere).
6. `'SelectionChangedFcn'`: Executes when a new item is selected in a group (e.g., "Radio" buttons).

Code 2.3 presents a simple example of an App Designer GUI where pressing a button displays the text input by the user in a label. It also includes a second button to clear both fields.

Code 2.3: **Dynamic GUI with App Designer.**

```
classdef trial < matlab.apps.AppBase
    % Properties that correspond to app components
    properties (Access = public)
        UIFigure      matlab.ui.Figure
        EraseButton   matlab.ui.control.Button
        OutputField   matlab.ui.control.Label
        InputField    matlab.ui.control.EditField
        DisplayButon  matlab.ui.control.Button
    end

    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: DisplayButon
        function DisplayButonPushed(app, event)
            app.OutputField.Text = app.InputField.Value;
        end

        % Button pushed function: EraseButton
        function EraseButtonPushed(app, event)
            app.InputField.Value = "";
            app.OutputField.Text = "";
        end
```

```
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Color = [1 1 1];
            app.UIFigure.Position = [100 100 640 480];
            app.UIFigure.Name = 'MATLAB App';

            % Create DisplayButon
            app.DisplayButon = uibutton(app.UIFigure, 'push');
            app.DisplayButon.ButtonPushedFcn = createCallbackFcn(app,
                @DisplayButonPushed, true);
            app.DisplayButon.FontSize = 36;
            app.DisplayButon.FontWeight = 'bold';
            app.DisplayButon.Position = [31 267 226 54];
            app.DisplayButon.Text = 'Display Text';

            % Create InputField
            app.InputField = uieditfield(app.UIFigure, 'text');
            app.InputField.HorizontalAlignment = 'center';
            app.InputField.FontSize = 18;
            app.InputField.FontColor = [0.651 0.651 0.651];
            app.InputField.Position = [304 267 289 54];
            app.InputField.Value = 'Insert text here';

            % Create OutputField
            app.OutputField = uilabel(app.UIFigure);
            app.OutputField.HorizontalAlignment = 'center';
            app.OutputField.FontSize = 18;
            app.OutputField.FontColor = [0.651 0.651 0.651];
            app.OutputField.Position = [304 197 289 54];
            app.OutputField.Text = 'Text is printed here!';

            % Create EraseButton
            app.EraseButton = uibutton(app.UIFigure, 'push');
            app.EraseButton.ButtonPushedFcn = createCallbackFcn(app,
                @EraseButtonPushed, true);
            app.EraseButton.FontSize = 36;
```

```
            app.EraseButton.FontWeight = 'bold';
            app.EraseButton.Position = [64 197 160 54];
            app.EraseButton.Text = 'Erase';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = trial

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

2.3.1 Drop-Down Menus

Drop-down menus (`uidropdown` in App Designer) are useful components when the app should offer users a limited set of predefined options, like *Array Designer* now can do. For example, they are commonly used for selecting units, methods, or categories. In App Designer, the drop-down component can be added from the *Design View*. A drop-down menu typically works with the `ValueChangedFcn` callback, which gets triggered when the user selects a new item. The syntax

of the callback is the same as other components:

```
function DropdownValueChanged(app, event)
    selectedItem = app.DropDown.Value;
    % Code that reacts to the selected item
end
```

Chapter 3

New features and improvements

To facilitate a better understanding of the code presented in this chapter, some essential explanations must first be provided. All operations are carried out using the `MNI_symmetric` atlas. In *Array Designer*, the cortical surface of the brain is modelled as a mesh composed of numerous points (nodes) connected in a triangulated pattern (faces) (fig. 3.1). This mesh is referred to in the file as `gmSurfaceMesh` (gray matter Surface Mesh), which is a MATLAB structure containing three components:

- A matrix called `node`, consisting of 14 646 rows and four columns. Each row represents a point on the surface, identified by its three-dimensional coordinates and an additional value indicating the parcel to which it belongs.
- A matrix named `face`, which defines the triangular faces formed by connecting the surface points.
- A structure called `parcellation`, which holds details of the parcellation scheme used—in this case, the *CerebrA* atlas.

In *Array Designer*, an ROI is visualised as a group of mesh nodes highlighted in pink, as shown in fig. 1.18.

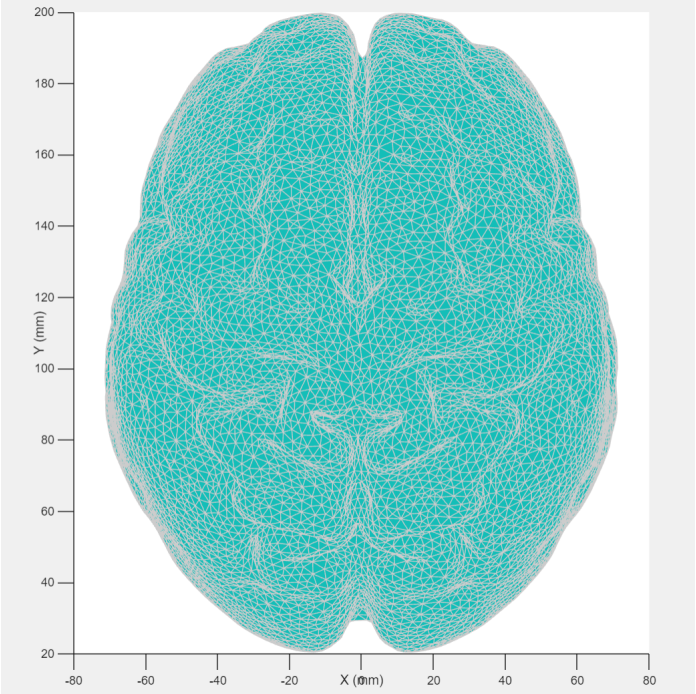

Figure 3.1: **Gray matter mesh created connecting 14 646 points.**

As reported in section 1.2.4, previously in *ROI Builder* an ROI could be created either drawing it on the cortical surface or selecting one or more parcels. The novel functionality introduced with this work allows instead to create an ROI starting from the MNI coordinates of interest that a user would like to cover. Therefore, for this purposes, an additional input field—`app.RadiusmmEditField.Value`—was introduced to allow the user to define the radius of the region of interest (fig. 3.2). The core idea is that each point on the mesh can serve as the centre of a sphere whose radius is specified by this input. This sphere determines the extent of the region: a larger radius includes more nearby nodes, whereas a smaller radius restricts the selection to a more limited area (fig. 3.3).

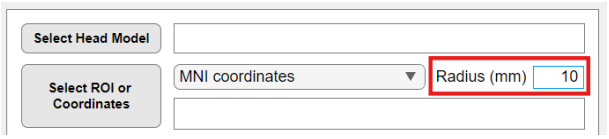

Figure 3.2: **Radius edit field with default length set to 10 mm.**

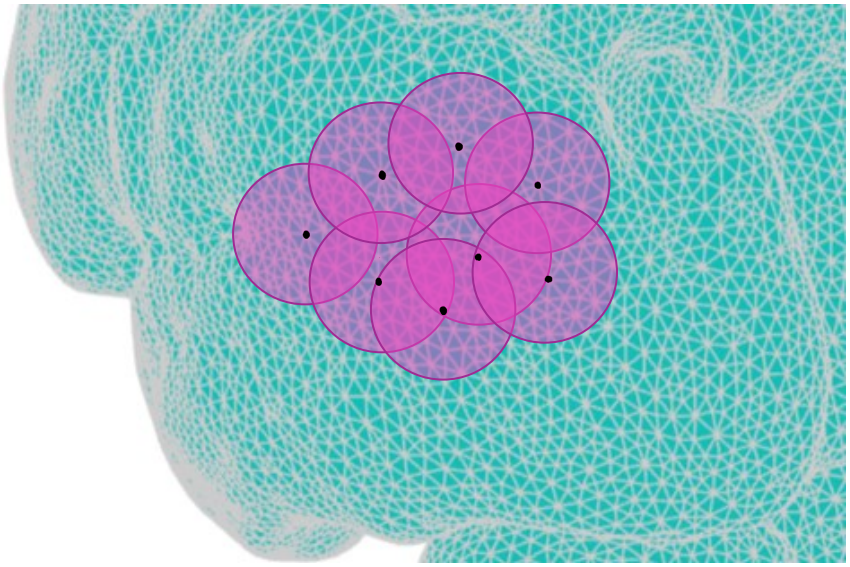

Figure 3.3: **Schematic representation of a mesh created from eight points and a chosen radius.**

3.1 Drop-down Menu for MNI Coordinates

To provide users an efficient method for selecting how to create their ROIs, either selecting a pre-existing ROI or choosing which MNI coordinates were of interest in order to create the ROI, a mechanism was needed to choose between multiple input options. A drop-down menu was identified as the most suitable solution, as it allows all available choices to be clearly presented.

In *Design View*, a drop-down element named `app.Dropdown` was added, offering two options: ”Existing ROI” and ”MNI coordinates”. The ”Select ROI” button label was updated to ”Select ROI or coordinates” to reflect this broader functionality.

To enable button-click interaction based on the selected option, the corresponding callback function was modified in *Code View*. The function `SelectROIorCoordinatesButtonPushed(app, event)`—previously named `SelectROI(app, event)`—was updated to include a `switch-case` statement, allowing it to handle the two distinct cases. For the ”Existing ROI” case, the original code for loading ROI files was preserved.

When ”MNI coordinates” is selected, the application prompts the user to upload a `.txt` or `.csv` file from the `ArrayDesigner` directory, containing 3D coordinate points. This requires retrieving the path to the relevant repository (`app.repoPath`), as shown in code 3.1.

Code 3.1: Code for acquiring the path from a selected file.

```
[filename, pathname, ~] = uigetfile(fullfile(app.repoPath, 'ArrayDesigner',
    '.txt;.csv'), 'Select MNI coordinates');
if filename == 0
```

```
return
end
fullpath = [pathname filename];
[pathname, filename, ext] = fileparts(fullpath);
app.pathnameMNI = fullfile(pathname, [filename ext]);
app.CurrentROIorMNIText.Value = [filename ext];

% Update text field
app.HeadModelPathText.Value = app.pathnameHeadModel;
```

As shown above, the `uigetfile` function prompts the user to select either a `.txt` or `.csv` file. Once a file is chosen, its full path is saved in the `app.pathnameMNI` property, while the filename (including its extension) is displayed in the text field `app.CurrentROIorMNIText.Value`—as illustrated in fig. 3.4.

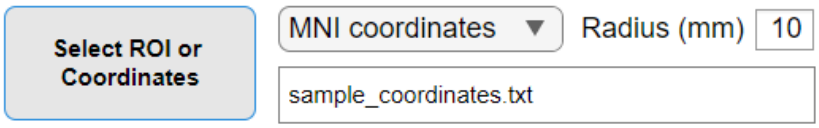

Figure 3.4: **Text field displaying the file name.**

The next step is to check whether each of the uploaded coordinates meet a maximum distance threshold of 1 mm from the `gmSurfaceMesh` points, to verify if they all lie sufficiently close to the actual grey matter surface modelled in the program. The coordinates are read into the matrix `upldCoor` using the `readmatrix` function and transposed, if necessary, to ensure a shape of 3 rows and n columns. Since the provided coordinates may not exactly match any node in `gmSurfaceMesh`, the Euclidean distance from each coordinate to every node is computed. For each point (i.e., each row of `upldCoor`), the closest node is identified by calculating the norm of the difference to all mesh nodes and the original coordinate of interest is assigned to that node. The smallest distance for each point is stored in the vector `distcnmin`, using two nested `for` loops (see code 3.2).

Code 3.2: Code for checking if the provided file contains coordinates belonging to `gmSurfaceMesh`

```
% Nearest nodes for MNI coordinate
upldCoor = readmatrix(fullpath); % Add the xyz coordinate file in a matrix
if size(upldCoor,1)<size(upldCoor,2) % If is 3xM
    upldCoor = upldCoor';
end

nodes = app.gmSurfaceMesh.node;
```

```
% Check that the file contains transformed coordinates
distcnmin = zeros(size(upldCoor, 1), 1);
for i = 1:size(upldCoor, 1) % For each uploaded coordinate
    distcn = zeros(size(nodes, 1), 1); % Initialize a vector for distances
    for j = 1:size(nodes, 1) % For each node
        % Calculation of 3D distances
        distx = abs(upldCoor(i, 1) - nodes(j, 1));
        disty = abs(upldCoor(i, 2) - nodes(j, 2));
        distz = abs(upldCoor(i, 3) - nodes(j, 3));
        norm = sqrt((distx)^2 + (disty)^2 + (distz)^2);
        distcn(j) = norm;
    end
    distcnmin(i) = min(distcn);
end
```

Next, we verify whether all distances satisfy the 1 mm threshold. If they do, this implies that `find(distcnmin > 1)` returns no indices and `isempty` returns `true`, as shown in code 3.3. In that case, the property `app.MNIcenters` is populated with the uploaded coordinates. Otherwise, if `isempty` returns `false`, meaning that at least one point does not meet the threshold, the code move to the "else" statement, where the cooridnates are computed to the closest node of `gmSurfaceMesh` (see code 3.6).

Code 3.3: Already transformed coordinates are computed to create the fist column of the ROI file.

```
% File with already transformed coordinates
if isempty(find(distcnmin > 1)) % If there are no distances that do not
                                % exceed the threshold of 1mm (each
                                % coordinate is almost coincident with
                                % the point on gmSurfaceMesh)
    app.MNIcenters = zeros(size(upldCoor,1),size(upldCoor,2));
        % Empty matrix initialization where the centers will go to
        % calculate the spheres
    for i= 1:size(upldCoor,1)
        for j= 1:size(upldCoor,2)
            app.MNIcenters(i,j) = upldCoor(i,j);
        end
    end
    firstColumnROI(app) % Calculation of column 1 of the ROI file
```

The resulting ROI is going to be saved as a $2 \times n$ `.txt` or `.csv` file (see section 3.2): the first column contains the indices of `gmSurfaceMesh` nodes within the ROI, and the second column contains the corresponding scalp indices. The first column is generated using the `firstColumnROI(app)` function (see code 3.4), which identifies mesh nodes within a user-

defined radius. The scalp indices in the second column are computed by finding, for each ROI node on the cortical surface, the closest scalp node within a specified search margin. This mapping reduces the number of scalp nodes considered in subsequent processing and serves to restrict the search space for *Array Designer* when optimizing the optode placement on the scalp.

Code 3.4: `firstolumnROI` function used to create the first column of the ROI file displayed on the brain mesh.

```
% Function for creating the first column of the ROI txt file
function firstColumnROI(app)
    nodes = app.gmSurfaceMesh.node;
    radius = app.RadiusmmEditField.Value; % Radius from input
    distcnSelected = [];
    for i = 1:size(app.MNIcenters, 1) % For each mni center
        distcn = zeros(size(nodes, 1), 1); % New distances vector
        for j = 1:size(nodes, 1) % For each node
            % Calulation of 3D distances
            distx = abs(app.MNIcenters(i, 1) - nodes(j, 1));
            disty = abs(app.MNIcenters(i, 2) - nodes(j, 2));
            distz = abs(app.MNIcenters(i, 3) - nodes(j, 3));
            norm = sqrt((distx)^2 + (disty)^2 + (distz)^2);
            distcn(j) = norm;
        end
        % Nodes indexes < radius
        ind = find(distcn < radius);

        % Indices in the current cell
        distcnSelected = [distcnSelected; ind];
    end
    distcnSelected = [distcnSelected; app.nnode_ind];
    distcnSelected = unique(distcnSelected);
    app.ROI.gmNodeList = distcnSelected;

    updateFigures(app)
end
```

If the coordinates do not meet the threshold, they are transformed to the closest matching nodes using the `DOTHUB_nearestNode(point, nodes)` function. This function, originally provided in a previous version of *Array Designer*, identifies the nearest `gmSurfaceMesh` node to a given 3D coordinate (see code 3.5). A private property called `nnode_ind` (nearest node index) is introduced to store the result of this mapping. The transformation is performed within a `for` loop in code 3.6.

Code 3.5: DOTHUB_nearestNode(point, nodes) function used to find the closest mesh node.

```
function [nnode, ind, offset] = DOTHUB_nearestNode(point,nodes)
%This function outputs the node (1x3) present in the list nodes (Mx3) that
%is closest
%to the 3D location specified by point (1x3) and the offset in whatever
%dimensions are provided.

%Inputs (point,nodes): the single point and the list of nodes
%Outputs [nnode, offset]: the index of the specific node in the list
%'nodes' which is nearest to the input point, and the offset (euclidean
%error).

% ##########################################################################
% RJC, UCL, April 2020

if size(point,2)~=3
    point = point';
    if size(point,2)~=3
        error('Dimensions of input point not 1x3');
    end
end

if size(nodes,2)~=3
    nodes = nodes';
    if size(nodes,2)~=3
        error('Dimensions of input nodes not Mx3');
    end
end

M = size(nodes,1);

if size(point,1)>1
    for i = 1:size(point,1)
        tmpPoint = point(i,:);
        dists = sqrt(sum((nodes - repmat(tmpPoint,M,1)).^2,2));
        [offset(i),ind(i)] = min(dists);
    end
else
    dists = sqrt(sum((nodes - repmat(point,M,1)).^2,2));
    [offset,ind] = min(dists);
end

nnode = nodes(ind,:);
```

Code 3.6: Coordinate transformation using `DOTHUB_nearestNode` and Euclidian distance of each point from the calculated counterpart.

```
else
    app.MNIcenters = zeros(size(upldCoor,1),size(upldCoor,2));
        % Empty matrix initialization where the centers will go to
        % calculate the spheres
    for i=1:size(upldCoor,1) % For each xyz value, find the corresponding
                             % center
        puntoMNI = upldCoor(i,:);
        [nnode, app.nnode_ind(i,1), ~] = DOTHUB_nearestNode(puntoMNI, nodes
            ); % Find the nearest node
        app.MNIcenters(i, :) = nnode; % Add the closest node to the matrix
    end

    % Check if the new points are too distant from the position in the
    % uploaded file
    exceedsThreshold =[];
    for i = 1:size(app.MNIcenters,1)
        distx = abs(app.MNIcenters(i,1) - upldCoor(i,1));
        disty = abs(app.MNIcenters(i,2) - upldCoor(i,2));
        distz = abs(app.MNIcenters(i,3) - upldCoor(i,3));
        norm = sqrt((distx)^2 + (disty)^2 + (distz)^2);
        dist(i,:) = [norm, distx, disty, distz];

        if norm > 10 % If the distance of each point exceed 1cm
            exceedsThreshold = [exceedsThreshold; i, norm, upldCoor(i, :),
                distx, disty, distz]; % Add index and coordinates to the
                                      % results vector
        end
    end
```

After transforming the coordinates to their nearest nodes on the `gmSurfaceMesh`, code 3.6 performs an additional step: it calculates the Euclidean distance between each original coordinate and its mapped counterpart. If the distance exceeds a threshold of 10 mm, the coordinate is flagged as potentially inaccurate and the results are compiled into the `exceedsThreshold` matrix, which stores the index of each problematic point, its total Euclidean distance from the mesh, the original coordinate values, and the individual deviations in the X, Y, and Z directions.

A final check is performed to determine if any of the new coordinates exceed the acceptable threshold. If they do, meaning that `isempty(exceedsThreshold)` returned `true`, a message is printed to the command window detailing which points are problematic (see code 3.7).

Code 3.7: Code to dispaly on Command Window if the provided coordinates are acceptable.

```
if ~isempty(exceedsThreshold) % Check if some coordinates are too far apart
    disp('The following points exceed the threshold of 10 mm! ');
    for j = 1:size(exceedsThreshold, 1)
        % Check if one or more coordinates exceed the threshold
        exceedingCoordinates = [];
        if exceedsThreshold(j,6) >= 10
            exceedingCoordinates = [exceedingCoordinates, "X"];
        end
        if exceedsThreshold(j,7) >= 10
            exceedingCoordinates = [exceedingCoordinates, "Y"];
        end
        if exceedsThreshold(j,8) >= 10
            exceedingCoordinates = [exceedingCoordinates, "Z"];
        end

        if ~isempty(exceedingCoordinates)
            % If at least one coordinate exceeds threshold
            fprintf('Index: %d | Coordinate (x, y, z): (%.2f, %.2f,
                %.2f) | 3D distance: %.2f mm \n', ...
                exceedsThreshold(j, 1), exceedsThreshold(j, 3), ...
                exceedsThreshold(j, 4), exceedsThreshold(j, 5), ...
                exceedsThreshold(j, 2));
            fprintf('Coordinates exceeding the 10 mm threshold:
                %s\n', strjoin(exceedingCoordinates, ', '));

        else  % If threshold is exceeded but none of the
              % coordinates exceed it
            fprintf(['Index: %d | Coordinate (x, y, z): (%.2f,
                %.2f, %.2f), 3D distance 3D: %.2f mm, \n', ...
                'No specific coordinate exceeds the threshold of 10
                mm. Provide a better point. \n'], ...
                exceedsThreshold(j, 1), exceedsThreshold(j, 3), ...
                exceedsThreshold(j, 4), exceedsThreshold(j, 5), ...
                exceedsThreshold(j, 2));
        end
        fprintf('---\n');
    end
```

After passing the `if` statement, in code 3.7 `exceedsThreshold` matrix is then parsed in a loop, and for each flagged coordinate, a detailed message is printed in the Command Window. The message includes the point index, its coordinate values, the overall 3D distance to the closest mesh node, and which specific axes contributed to the deviation. If none of the individual axes exceed the 10 mm threshold but the total distance still does, a message indicates that no single component is responsible, and the user is advised to provide a better input point (fig. 3.5).

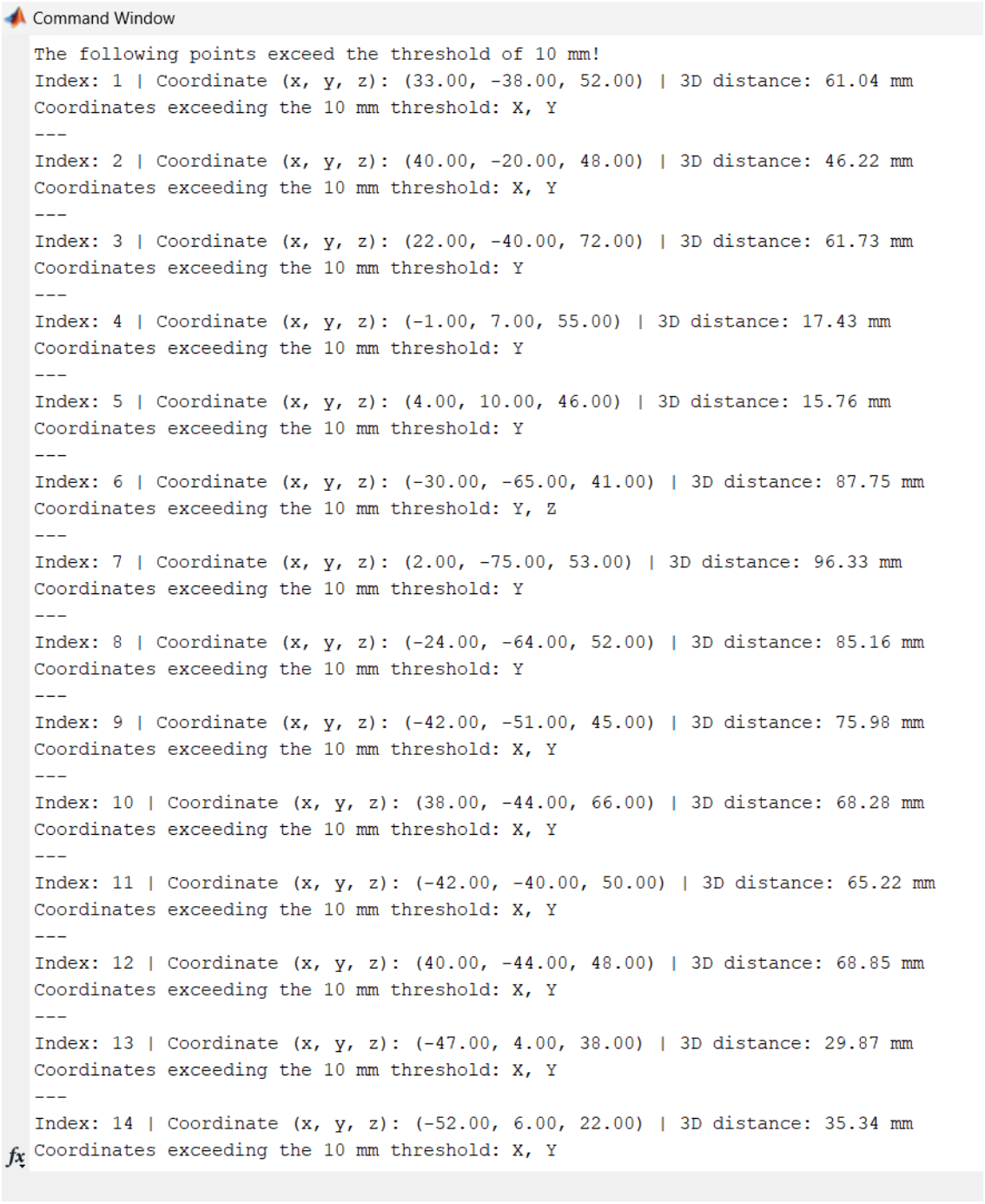

Figure 3.5: **Command Window message displayed when coordinates provided do not meet the criteria.**

3.2 Saving coordinates

Once the coordinates have been provided and any necessary calculations have been completed, a window appears prompting the user to save the ROI coordinates. This window is created using the `uifigure` function and the saved file is located within the `MNI_symmetric` folder under the `ROIs` directory in the *Array Designer* path (see code 3.8). The window, titled "Save File", appears as a fixed-size blank canvas centred on the user's screen. A grid layout including two columns and three rows is created to organise the interface components. These consist in a text field for entering the file name, a drop-down menu to select the file extention and two buttons, one to save and another to cancel the operation (fig. 3.6).

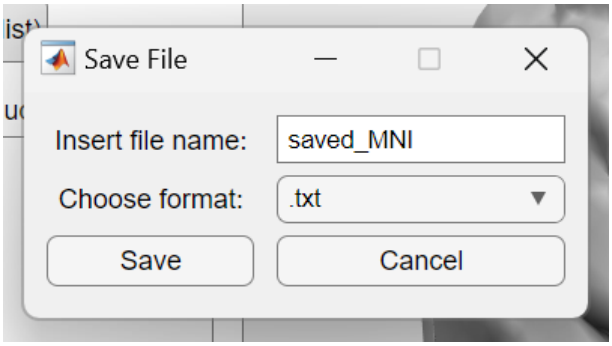

Figure 3.6: **Dialog window for file saving.**

Component positions within the grid are specified using the "Layout" property of each UI element, such as `app.tmp_savedfile.Layout.Row` and `app.tmp_savedfile.Layout.Column`.

Code 3.8: Code to generate a window for saving the ROI after providing MNI coordinates.

```
else
    % File saving and plotting
    % Window creation
    app.save_menu = uifigure('Name', 'Save File', 'Position', [500, 300,
        250, 100], 'Resize', 'off');

    % Creating a grid layout with 3 rows and 2 columns
    grid = uigridlayout(app.save_menu, [3, 2]);
    grid.RowHeight = {'fit', 'fit', 'fit'};
    grid.ColumnWidth = {'6.5x', '9x'};
    grid.Padding = [10, 10, 10, 10];
    grid.RowSpacing = 5;
    grid.ColumnSpacing = 10;

    % File name label
    nameLabel = uilabel(grid, 'Text', 'Insert file name:', ...
        'HorizontalAlignment', 'center', ...
```

```
        'FontSize', 12);
    nameLabel.Layout.Row = 1; nameLabel.Layout.Column = 1;

    % Filename
    app.tmp_savedfile = uieditfield(grid, 'text', ...
        'Value', 'MNI_salvate', ...
        'HorizontalAlignment', 'left', ...
        'FontSize', 11);
    app.tmp_savedfile.Layout.Row = 1;  app.tmp_savedfile.Layout.Column = 2;

    % Extension label
    extLabel = uilabel(grid, 'Text', 'Choose format:', ...
        'HorizontalAlignment', 'center', ...
        'FontSize', 12);
    extLabel.Layout.Row = 2; extLabel.Layout.Column = 1;

    % Extension drop-down
    app.tmp_savedext = uidropdown(grid, ...
        'Items', {'.txt', '.csv'}, ...
        'Value', '.txt', ...
        'FontSize', 11);
    app.tmp_savedext.Layout.Row = 2; app.tmp_savedext.Layout.Column = 2;

    % Save button
    saveButton = uibutton(grid, ...
        'Text', 'Save', ...
        'ButtonPushedFcn', @(btn, event) saveCoordinates(app), ...
        'FontSize', 12);
    saveButton.Layout.Row = 3; saveButton.Layout.Column = 1;

    % Cancel button
    cancelButton = uibutton(grid, ...
        'Text', 'Cancel', ...
        'ButtonPushedFcn', @(btn, event) cancelSaving(app), ...
        'FontSize', 12);
    cancelButton.Layout.Row = 3; cancelButton.Layout.Column = 2;

    firstColumnROI(app)
end
```

If the user chooses to proceed and clicks the ”Save” button, the `ButtonPushedFcn` callback is triggered. This invokes the `saveCoordinates(app)` function (see code 3.9), which handles the saving process. It first checks whether a file name has been entered. If not, the operation is aborted and no file is saved. If a valid name is provided, the file path is assembled using the `fullfile`

function, and the coordinate matrix is saved in the selected format using a `switch-case` structure. After saving, the save window is closed via the `delete()` function.

Code 3.9: `saveCoordinates` function that executes the file saving process.

```
% Save coordinates button
function saveCoordinates(app)
    % Check if user had entered a chosen file name
    if isempty(app.tmp_savedfile)
        disp('Operation canceled. No files will be saved.');
        return;
    end

    % Full name of the file
    fileName = [app.tmp_savedfile.Value, app.tmp_savedext.Value];
    MNIdirectoryPath = fullfile(app.pathnameHeadModel, 'MNI coordinates',
       fileName);

    % Saving file based on extension
    switch app.tmp_savedext.Value
        case '.txt'
            writematrix(app.MNIcenters, MNIdirectoryPath, 'Delimiter', '\t'
            ); % Saved as .txt
        case '.csv'
            writematrix(app.MNIcenters, MNIdirectoryPath); % Saved as .csv
    end

    delete(app.save_menu); % losing the saving menu
end
```

On the contrary, if the user decides to cancel the operation by clicking the ”Cancel” button, the `ButtonPushedFcn` callback triggers the `cancelSaving(app)` function (see code 3.10), which immediately closes the save window and displays a message in the Command Window to inform the user (fig. 3.7).

Code 3.10: `cancelSaving` function that aborts the saving operation.

```
% Cancel button
function cancelSaving(app)
    disp('Saving cancelled');
    delete(app.save_menu); % Closing the saving menu
end
```

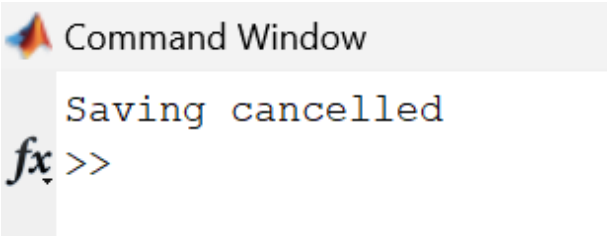

Figure 3.7: **Message displayed in Command Window after cancelling the saving operation.**

Although it would be possible to include the file-saving logic directly within the `uibutton` definitions, in the implementation it has been chosen to create two separate functions—`saveCoordinates(app)` and `cancelSaving(app)`—to promote better code organisation and maintainability.

3.3 Existing code improvement

Several adjustments were made to enhance the usability and maintainability of the code. First, the path-handling process within the `SelectHeadModelButtonPushed(app, event)` function was improved by using the `fullfile(filepartN)` function. This approach ensures cross-platform compatibility between different operating systems by eliminating the need to manually specify backslashes (\) or forward slashes (/) in file paths. Following this, the path is reconstructed by explicitly checking whether the operating system is Linux/macOS or Windows (see fig. 3.8).

```
209 % Button pushed function: SelectHeadModelButton
210 function SelectHeadModelButtonPushed(app, event)
211     [filename, pathname, ~] = uigetfile(fullfile(app.repoPath, '*.mshs'),
212     if filename==0
213         return
214     end
215     fullpath = [pathname filename];
216     [pathname,filename,ext] = fileparts(fullpath);
217     if isunix % check if the OS is MacOS or Linux
218         pathnameHeadModeltmp = [pathname '/' filename];
219     elseif ispc % check if is the OS is Microsoft
220         pathnameHeadModeltmp = [pathname '\' filename];
221     end
```

```
143 % Button pushed function: SelectHeadModelButton
144 function SelectHeadModelButtonPushed(app, event)
145     [filename, pathname, ~] = uigetfile([app.repoPath '/*.mshs'],'Select
146     if filename==0
147         return
148     end
149     fullpath = [pathname filename];
150     [pathname,filename,ext] = fileparts(fullpath);

151     pathnameHeadModeltmp = [pathname '/' filename];
```

Figure 3.8: **On the left: original code. On the right: old code.**

The use of `fullfile(filepartN)` was consistently applied throughout the codebase wherever path manipulation was required (see Figure fig. 3.9).

Additionally, an issue related to cleaning the workspace—specifically when changing from the symmetric to the asymmetric atlas and selecting ”Clear ROI” button—was resolved. Previously, the parcellation list was not properly removed, resulting in graphical inconsistencies. This was corrected by programmatically selecting the second item in the `gmSurfaceMesh.parcellation`

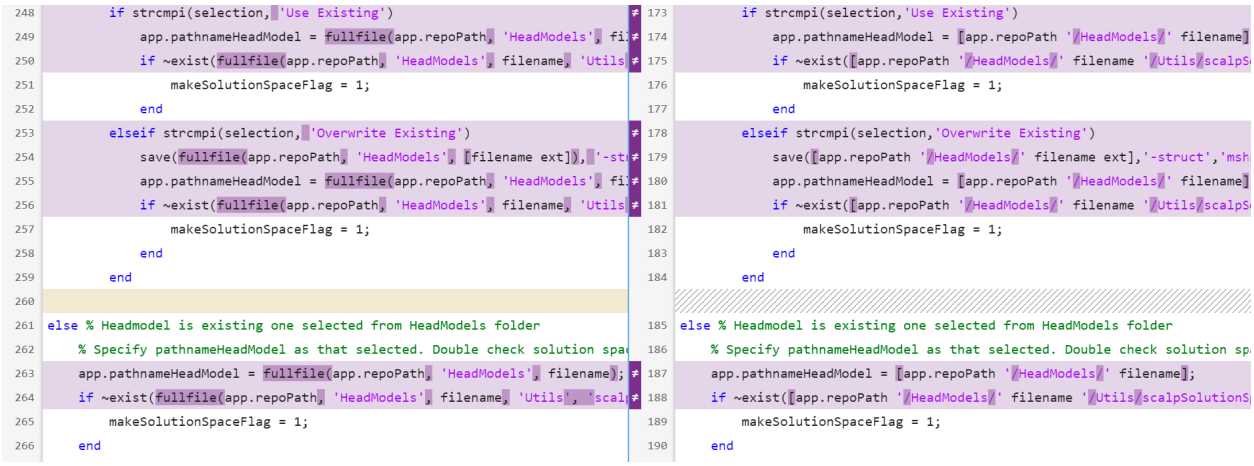

Figure 3.9: **On the left: new code. On the right: old code.**

structure, which contains the indexed parcellation matrix, ensuring that the previous parcellation is correctly cleared (see Figure fig. 3.10).

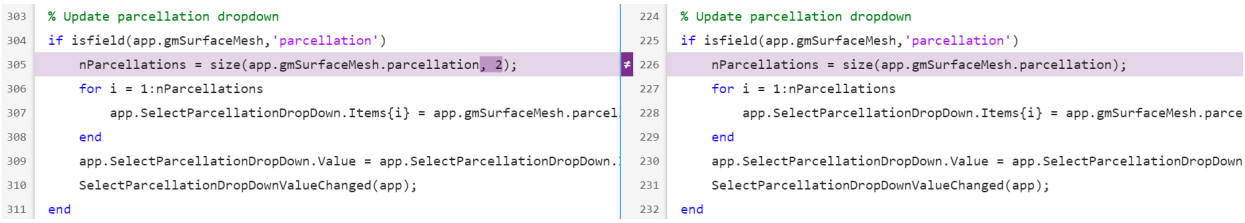

Figure 3.10: **On the left: new code. On the right: old code.**

3.4 UI design improvement

In addition to the modifications to the UI introduced by the addition of the drop-down menu and the radius input textbox, several minor adjustments were made to enhance the overall clarity and proportionality of the interface. These refinements were implemented using *Design View* in the App Designer environment. Specifically, some panels were slightly resized—either enlarged or reduced—and aligned more consistently with adjacent components. The differences can be observed by comparing Figure fig. 1.22 with Figure fig. 3.11.

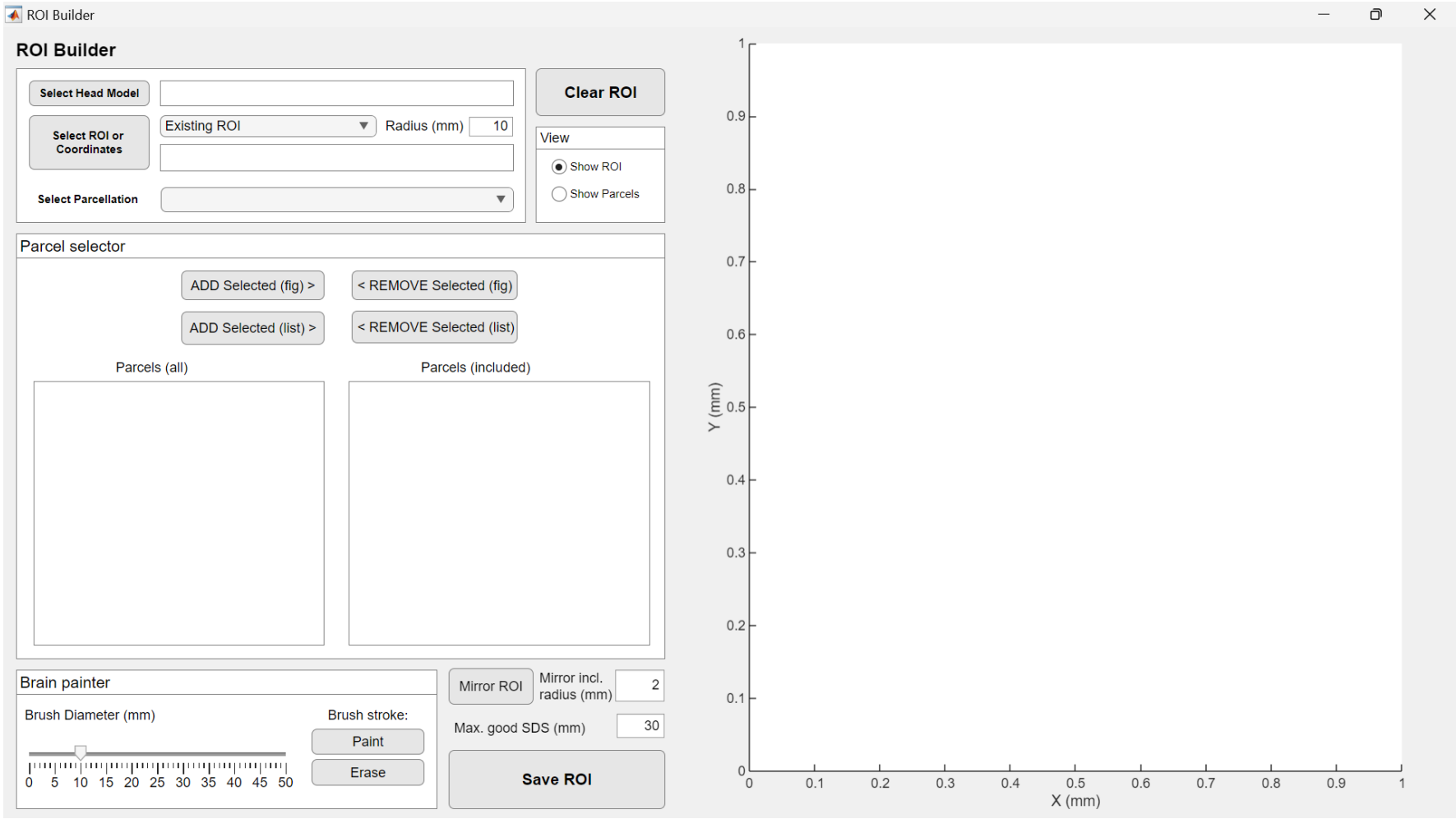

Figure 3.11: **Final *ROI Builder* User Interface.**

Chapter 4

Conclusions

fNIRS can be a powerful, accessible, and easy-to-use neuroimaging technique, but there are still some processes that are subjective and not automatic, e.g., the array design process. It is therefore important to realize and improve tools that make this process more objective and automatic. Among these tools, *Array Designer* stands out as a valuable solution, but an upgrade was required to incorporate new important features that could be very useful to users.

The improvements introduced in this project, although small additions in terms of lines of code, represent a significant enhancement for user experience. Researchers can now generate ROIs not only from standard `ROI.txt` files but also directly from MNI coordinates. In addiction, the feedback provided through the Command Window and the ability to export computed coordinates increase the tool's flexibility.

Several developments could further enrich *ROI Builder*. These include the integration of additional MNI152-based parcellations and the support for alternative atlases, including subject-specific ones.

This project represents a step forward in supporting fNIRS researchers through improved tool usability and customizability. Continued development in this direction will help standardize and simplify optode layout design, ultimately contributing to more robust and reproducible neuroimaging studies.

Bibliography

[1] M. J. Fulham, «Neuroimaging», in *Encyclopedia of Neuroscience*, L. R. Squire, Ed., Cambridge, MA, USA: Academic Press, 2004, pp. 459–469. DOI: doi.org/10.1016/B978-008045046-9.00309-0.

[2] M. E. Raichle, «A brief history of human brain mapping», *Trends in Neurosciences*, vol. 32, no. 2, pp. 118 –126, Feb. 2009. DOI: 10.1016/j.tins.2008.11.001.

[3] L. Minati, «Neuroimaging techniques: A conceptual overview of physical principles, contribution and history», *AIP Conference Proceedings*, vol. 839, pp. 503–519, Jun. 2006. DOI: 10.1063/1.2216661.

[4] A. Erol and B. Hunyadi, *Chapter 12 - Tensors for neuroimaging: A review on applications of tensors to unravel the mysteries of the brain*. Academic Press, 2022. DOI: 10.1016/B978-0-12-824447-0.00018-2.

[5] L. Zhang et al., «Deep fusion of brain structure-function in mild cognitive impairment», *Medical Image Analysis*, vol. 72, no. 102082, Aug. 2021. DOI: 10.1016/j.media.2021.102082.

[6] A. Gholipour, N. Kehtarnavaz, R. Briggs, M. Devous, and K. Gopinath, «Brain functional localization: A survey of image registration techniques», *IEEE Transactions on Medical Imaging*, vol. 26, no. 4, pp. 427–451, Apr. 2007. DOI: 10.1109/TMI.2007.892508.

[7] S. Brigagoi, G. Rizzo, and M. Veronese, *Basic neuroimaging: a guide to the methods and their applications*. London: Create Space, 2017.

[8] A. A. T. Garcia, *Biosignal Processing and Classification Using Computational Learning and Intelligence: Principles, Algorithms, and Applications*. London, England: Academic Press, 2022, pp. 41–42. DOI: 10.1016/C2019-0-00985-5.

[9] G. E. Strangman, Z. Li, and Z. Quan, «Depth sensitivity and source-detector separations for near infrared spectroscopy based on the colin27 brain template», *PLOS ONE*, vol. 8, no. 8, e66319, Aug. 2013. DOI: 10.1371/journal.pone.0066319.

[10] R. Li, D. Yang, F. Fang, K.-S. Hong, A. L. Reiss, and Y. Zhang, «Concurrent fnirs and eeg for brain function investigation: A systematic, methodology-focused review», *Sensors*, vol. 22, no. 15, p. 5865, Aug. 2022. DOI: 10.3390/s22155865.

[11] J. Gervain et al., «Near-infrared spectroscopy: A report from the mcdonnell infant methodology consortium», *Developmental Cognitive Neuroscience*, vol. 1, no. 1, pp. 22–46, Jan. 2011. DOI: 10.1016/j.dcn.2010.07.004.

[12] A. Benitez-Andonegui, M. Lührs, L. Nagels-Coune, D. Ivanov, R. Goebel, and B. Sorger, «Guiding functional near-infrared spectroscopy optode-layout design using individual (f)mri data: Effects on signal strength», *Neurophotonics*, vol. 8, no. 2, p. 025 012, Apr. 2021. DOI: 10.1117/1.NPh.8.2.025012.

[13] R. K. Almajidy, K. Mankodiya, M. Abtahi, and U. G. Hofmann, «A newcomer's guide to functional near infrared spectroscopy experiments», *IEEE Reviews in Biomedical Engineering*, vol. 13, pp. 292–308, Oct. 2019. DOI: 10.1109/RBME.2019.2944351.

[14] J. Meek, «Basic principles of optical imaging and application to the study of infant development», *Developmental Science*, vol. 5, no. 3, pp. 371–380, Jul. 2002. DOI: 10.1111/1467-7687.00376.

[15] S. R. Arridge, «Photon-measurement density functions. part i: Analytical forms», *Appl. Opt.*, vol. 34, no. 31, pp. 7395–7409, Nov. 1995. DOI: 10.1364/AO.34.007395.

[16] S. R. Arridge and M. Schweiger, «Photon-measurement density functions. part ii: Finite-element-method», *Appl. Opt.*, vol. 34, no. 34, pp. 8026–8037, Dec. 1995. DOI: 10.1364/AO.34.008026.

[17] L. Wei et al., «Transcranial brain atlas based on photon measurement density function in a triple-parameter standard channel space», *NeuroImage*, vol. 307, no. 121026, Feb. 2025. DOI: 10.1016/j.neuroimage.2025.121026.

[18] D. Tsuzuki and I. Dan, «Spatial registration for functional near-infrared spectroscopy: From channel position on the scalp to cortical location in individual and group analyses», *NeuroImage*, vol. 85, pp. 92–103, Jan. 2014, Celebrating 20 Years of Functional Near Infrared Spectroscopy (fNIRS). DOI: 10.1016/j.neuroimage.2013.07.025.

[19] M. Brett, I. S. Johnsrude, and A. M. Owen, «The problem of functional localization in the human brain», *Nature Reviews Neuroscience*, vol. 3, no. 3, pp. 243–249, Mar. 2002. DOI: 10.1038/nrn756.

[20] H. Gray, *Gray's Anatomy*, 20th. Lea & Febiger, 1918.

[21] A. C. Evans, D. L. Collins, and B. Milner, «An mri-based stereotaxic atlas from 250 young normal subjects», in *Proceedings of the 22nd Annual Symposium, Society for Neuroscience*, vol. 18, 1992, p. 408.

[22] A. C. Evans et al., «Anatomical mapping of functional activation in stereotactic coordinate space», *NeuroImage*, vol. 1, no. 1, pp. 43–63, 1992.

[23] D. L. Collins, P. Neelin, T. M. Peters, and A. C. Evans, «Automatic 3-d intersubject registration of mr volumetric data in standardized talairach space», *Journal of Computer Assisted Tomography*, vol. 18, no. 2, pp. 192–205, 1994.

[24] A. C. Evans, D. L. Collins, S. R. Mills, E. D. Brown, R. L. Kelly, and T. M. Peters, «3d statistical neuroanatomical models from 305 mri volumes», in *Proceedings of the IEEE Nuclear Science Symposium and Medical Imaging Conference*, 1993, pp. 1813–1817.

[25] MRC CBU Wiki. «The MNI brain and the Talairach atlas». Accessed: Apr. 9, 2025. [Online]. Available: `https://imaging.mrc-cbu.cam.ac.uk/imaging/MniTalairach`.

[26] Q. Wang, W. Yu, K. Chen, Z. Zhang, and R. A. M. Asad, «Brain cognitive comparison of fabric touch on human glabrous and hairy skin», *Textile Research Journal*, vol. 86, no. 3, pp. 318–324, Feb. 2016. DOI: `10.1177/0040517515591785`.

[27] C. J. Holmes et al., «Enhancement of mr images using registration for signal averaging», *Journal of Computer Assisted Tomography*, vol. 22, no. 2, pp. 324–333, Mar. 1998. DOI: `10.1097/00004728-199803000-00032`.

[28] F. Klein, «Optimizing spatial specificity and signal quality in fnirs: An overview of potential challenges and possible options for improving the reliability of real-time applications», *Frontiers in Neuroergonomics*, vol. 5, no. 1286586, Jun. 2024, Rev. DOI: `10.3389/fnrgo.2024.1286586`.

[29] T. Liu, «A few thoughts on brain rois», *Brain imaging and behavior*, vol. 5, no. 3, pp. 189–202, Sep. 2011. DOI: `10.1007/s11682-011-9123-6`.

[30] A. L. Manera, M. Dadar, V. Fonov, and D. L. Collins, «Cerebra, registration and manual label correction of mindboggle-101 atlas for mni-icbm152 template», *Scientific Data*, vol. 7, no. 1, p. 237, Dec. 2020. DOI: `10.1038/s41597-020-0557-9`.

[31] S. Brigadoi, D. Salvagnin, M. Fischetti, and R. J. Cooper, «Array Designer: automated optimized array design for functional near-infrared spectroscopy», *Neurophotonics*, vol. 5, no. 3, May 2018. DOI: `10.1117/1.NPh.5.3.035010`.

Appendix A

MATLAB code

A.1 Function SelectROIorCoordinatesButtonPushed

Code A.1: Function SelectROIorCoordinatesButtonPushed, case 'MNI coordinates'.

```
% User has selected a file with xyz coordinates
case 'MNI coordinates'
    [filename, pathname, ~] = uigetfile(fullfile(app.repoPath, '
       ArrayDesigner', '*.txt;*.csv'), 'Select MNI coordinates');
    if filename == 0
        return
    end
    fullpath = [pathname filename];
    [pathname,filename,ext] = fileparts(fullpath);
    app.pathnameMNI = fullfile(pathname,[filename ext]);
    app.CurrentROIorMNIText.Value = [filename ext];

    % Update text field
    app.HeadModelPathText.Value = app.pathnameHeadModel;

    % Nearest nodes for MNI coordinate
    upldCoor = readmatrix(fullpath); % Add the xyz coordinate file in a
                                     % matrix
    if size(upldCoor,1)<size(upldCoor,2) % If is 3xM
        upldCoor = upldCoor';
    end

    nodes = app.gmSurfaceMesh.node;

    % Check that the file contains transformed coordinates
    distcnmin = zeros(size(upldCoor, 1), 1);
```

```
for i = 1:size(upldCoor, 1) % For each uploaded coordinate
    distcn = zeros(size(nodes, 1), 1); % Initialize a vector for
                                       % distances
    for j = 1:size(nodes, 1) % For each node
        % Calculation of 3D distances
        distx = abs(upldCoor(i, 1) - nodes(j, 1));
        disty = abs(upldCoor(i, 2) - nodes(j, 2));
        distz = abs(upldCoor(i, 3) - nodes(j, 3));
        norm = sqrt((distx)^2 + (disty)^2 + (distz)^2);
        distcn(j) = norm;
    end
    distcnmin(i) = min(distcn);
end

% File with already transformed coordinates
if isempty(find(distcnmin > 1)) % If there are no distances that do not
                                % exceed the threshold of 1mm (each
                                % coordinate is almost coincident with
                                % the point on gmSurfaceMesh)
    app.MNIcenters = zeros(size(upldCoor,1),size(upldCoor,2));
        % Empty matrix initialization where the centers will go to
        % calculate the spheres
    for i= 1:size(upldCoor,1)
        for j= 1:size(upldCoor,2)
            app.MNIcenters(i,j) = upldCoor(i,j);
        end
    end
    firstColumnROI(app) % Calculation of column 1 of the ROI file

% File with coordinates to be transformed
else
    app.MNIcenters = zeros(size(upldCoor,1),size(upldCoor,2));
        % Empty matrix initialization where the centers will go to
        % calculate the spheres
    for i=1:size(upldCoor,1) % For each xyz value, find the
                             % corresponding center
        puntoMNI = upldCoor(i,:);
        [nnode, app.nnode_ind(i,1), ~] = DOTHUB_nearestNode(puntoMNI,
           nodes); % Find the nearest node
        app.MNIcenters(i, :) = nnode; % Add the closest node to the
                                      % matrix
    end

    % Check if the new points are too distant from the position in the
```

```
% uploaded file
exceedsThreshold =[];
for i = 1:size(app.MNIcenters,1)
    distx = abs(app.MNIcenters(i,1) - upldCoor(i,1));
    disty = abs(app.MNIcenters(i,2) - upldCoor(i,2));
    distz = abs(app.MNIcenters(i,3) - upldCoor(i,3));
    norm = sqrt((distx)^2 + (disty)^2 + (distz)^2);
    dist(i,:) = [norm, distx, disty, distz];

    if norm > 10 % If the distance of each point exceed 1cm
        exceedsThreshold = [exceedsThreshold; i, norm, upldCoor(i,
            :), distx, disty, distz]; % Add index and coordinates to
                        % the results vector
    end
end

if ~isempty(exceedsThreshold) % Check if some coordinates are too
                              % far apart
    disp('The following points exceed the threshold of 10 mm! ');
    for j = 1:size(exceedsThreshold, 1)
        % Check if one or more coordinates exceed the threshold
        exceedingCoordinates = [];
        if exceedsThreshold(j,6) >= 10
            exceedingCoordinates = [exceedingCoordinates, "X"];
        end
        if exceedsThreshold(j,7) >= 10
            exceedingCoordinates = [exceedingCoordinates, "Y"];
        end
        if exceedsThreshold(j,8) >= 10
            exceedingCoordinates = [exceedingCoordinates, "Z"];
        end

        if ~isempty(exceedingCoordinates)
            % If at least one coordinate exceeds threshold
            fprintf('Index: %d | Coordinate (x, y, z): (%.2f, %.2f,
                %.2f) | 3D distance: %.2f mm \n', ...
                exceedsThreshold(j, 1), exceedsThreshold(j, 3), ...
                exceedsThreshold(j, 4), exceedsThreshold(j, 5), ...
                exceedsThreshold(j, 2));
            fprintf('Coordinates exceeding the 10 mm threshold:
                %s\n', strjoin(exceedingCoordinates, ', '));

        else  % If threshold is exceeded but none of the
              % coordinates exceed it
```

```
                fprintf(['Index: %d | Coordinate (x, y, z): (%.2f,
                    %.2f, %.2f), 3D distance 3D: %.2f mm, \n', ...
                    'No specific coordinate exceeds the threshold of 10
                    mm. Provide a better point. \n'], ...
                    exceedsThreshold(j, 1), exceedsThreshold(j, 3), ...
                    exceedsThreshold(j, 4), exceedsThreshold(j, 5), ...
                    exceedsThreshold(j, 2));
            end
            fprintf('---\n');
        end
    else
        % File saving and plotting
        % Window creation
        app.save_menu = uifigure('Name', 'Save File', 'Position', [500,
            300, 250, 100], 'Resize', 'off');

        % Creating a grid layout with 3 rows and 2 columns
        grid = uigridlayout(app.save_menu, [3, 2]);
        grid.RowHeight = {'fit', 'fit', 'fit'};
        grid.ColumnWidth = {'6.5x', '9x'};
        grid.Padding = [10, 10, 10, 10];
        grid.RowSpacing = 5;
        grid.ColumnSpacing = 10;

        % File name label
        nameLabel = uilabel(grid, 'Text', 'Insert file name:', ...
            'HorizontalAlignment', 'center', ...
            'FontSize', 12);
        nameLabel.Layout.Row = 1; nameLabel.Layout.Column = 1;

        % Filename
        app.tmp_savedfile = uieditfield(grid, 'text', ...
            'Value', 'MNI_salvate', ...
            'HorizontalAlignment', 'left', ...
            'FontSize', 11);
         app.tmp_savedfile.Layout.Row = 1;  app.tmp_savedfile.Layout.
            Column = 2;

        % Extension label
        extLabel = uilabel(grid, 'Text', 'Choose format:', ...
            'HorizontalAlignment', 'center', ...
            'FontSize', 12);
        extLabel.Layout.Row = 2; extLabel.Layout.Column = 1;
```

```
            % Extension dropdown
            app.tmp_savedext = uidropdown(grid, ...
                'Items', {'.txt', '.csv'}, ...
                'Value', '.txt', ...
                'FontSize', 11);
            app.tmp_savedext.Layout.Row = 2; app.tmp_savedext.Layout.Column
                = 2;

            % Save button
            saveButton = uibutton(grid, ...
                'Text', 'Save', ...
                'ButtonPushedFcn', @(btn, event) saveCoordinates(app), ...
                'FontSize', 12);
            saveButton.Layout.Row = 3; saveButton.Layout.Column = 1;

            % Cancel button
            cancelButton = uibutton(grid, ...
                'Text', 'Cancel', ...
                'ButtonPushedFcn', @(btn, event) cancelSaving(app), ...
                'FontSize', 12);
            cancelButton.Layout.Row = 3; cancelButton.Layout.Column = 2;

            firstColumnROI(app)
        end
    end
```

A.2 Function RadiusmmEditFieldValueChanged

Code A.2: Function RadiusmmEditFieldValueChanged.

```
% Value changed function: RadiusmmEditField
function RadiusmmEditFieldValueChanged(app, event)
    app.ROI = struct('gmNodeList',[],'scalpPosList',[]);
    updateFigures(app);
    firstColumnROI(app)
end
```

A.3 Max Rho input

Code A.3: Max Rho input added to already existing code.

```
% Create solution space subset associated with this ROI
scalpSolSpace = load([app.pathnameHeadModel '/Utils/
    scalpSolutionSpace_10_2p5.txt']);
maxRho = app.MaxgoodSDSmmEditField.Value; % user input for MaxRho
[~, solSpaceSubsetInd] = getScalpSolSpaceSubset(app.gmSurfaceMesh,
    scalpSolSpace,app.ROI.gmNodeList,maxRho);
app.ROI.solutionSpaceNodeList = solSpaceSubsetInd;
```

Appendix B

Dataset

B.1 `test_coordinates.txt` file

```
33 40 22 -1 4 -30 2 -24 -42 38 -42 40 -47 -52 46 38 50
-38 -20 -40 7 10 -65 -75 -64 -51 -44 -40 -44 4 6 -30 -22 -36
52 48 72 55 46 41 53 52 45 66 50 48 38 22 44 60 44
```

B.2 `test_coordinates.csv` file

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
x	33	40	22	-1	4	-30	2	-24	-42	38	-42	40	-47	-52	46	38	50
y	-38	-20	-40	7	10	-65	-75	-64	-51	-44	-40	-44	4	6	-30	-22	-36
z	52	48	72	55	46	41	53	52	45	66	50	48	38	22	44	60	44