Scientific Visualization, 2024, volume 16, number 5, pages 151 - 163, DOI: 10.26583/sv.16.5.10

Additive Manufacturing of Personalized Brain-Computer Interface Headsets Reinforced by Scientific Visualization

D.A. Chiruhin[1,A], K.V. Ryabinin[2,B]

[A] Perm State University, Perm, Russia
[B] Astronomisches Rechen-Institut, Centre for Astronomy of Heidelberg University, Heidelberg, Germany

[1] ORCID: 0009-0008-7683-6978, chiruhind@gmail.com
[2] ORCID: 0000-0002-8353-7641, kostya.ryabinin@gmail.com

Abstract

Recently, a large attention has been attracted to the brain-computer human-machine interfaces (BCI) based on electroencephalography (EEG). This emerging technology allows touchless control over digital systems, in which the commands are based on human brain activity. In the ideal case, it means controlling the systems virtually by thoughts, but in reality, also simpler approaches are highly demanded like reacting to concentration, relaxation, or specific emotions. Modern BCIs are based on detecting so-called brain waves, the electromagnetic field oscillations induced by brain neurons. These waves are captured by electrodes either intruded into the brain or placed on top of the head. Obviously, placing electrodes on the head is more demanded for non-medical applications of BCI because it is absolutely harmless for the person. To achieve this, special headsets are needed which can be put on the head like a helmet and ensure the correct positions for the electrodes mounted on them. In this regard, wearing comfort and anatomical accuracy of headsets play an important role in ensuring both ergonomics and precision of BCI. This paper focuses on automation of the personalized EEG headset manufacturing for BCI. The technological chain is proposed and corresponding software tools are developed to foster the complete cycle of BCI headset production for a particular person. The production steps include 3D scanning of the head, interactive editing of the electrodes' location system, and automatic generation of a collapsible head cap model with sockets for EEG electrodes optimized for 3D printing. The performance of the pipeline has been validated in practice. The accuracy of electrodes' placement has been evaluated by comparison with the head cap from professional medical equipment and is established as sufficient for BCI. The headset model editing and customizing tools are powered with scientific visualization and cognitive graphics techniques to be friendly for a wide range of users including those with no dedicated IT skills.

Keywords: Brain-Computer Interface, 3D Scanning, 3D Printing, Dry Electrodes, Electroencephalography, Cognitive graphics, Scientific Visualization.

1. Introduction

Brain-computer interfaces (BCIs) are an emerging technology providing a completely new way of interacting with digital equipment including computing systems, Internet of Things (IoT) devices, medical apparatuses, entertainment gadgets, etc. Being a young research field, BCIs still lack well-established consumer-grade tools but attract a lot of attention from neurophysiologists, and computer scientists, as well as software and hardware developers.

Modern BCI works by detecting and decoding so-called brain waves (electroencephalography, EEG), electromagnetic oscillations induced by human brain activity. Based on physical implementation, BCI can be divided into invasive and non-invasive. Invasive BCI involves electrodes, which intrude the brain through the skull and receive the electrical signal from direct contact with the brain cells. This allows for a high sensitivity and high precision of signal localization but requires complicated surgery, which makes this kind of BCI applicable for medical purposes only. Non-invasive BCI utilizes electrodes placed on the scalp. Although the precision of brain waves detection is much lower, this approach is much more attractive to researchers, especially those outside of clinical studies.

While conducting research on non-invasive BCI, it is convenient to use so-called dry electrodes, which are placed directly on the scalp without the need for conductive gel [1-2]. These electrodes require a rigid headset that fits on the head, similar to a helmet. There are standard models of such headsets, but they are designed for an average head size and often do not fit an individual properly, failing to secure the electrodes reliably in the required locations (according to a specified electrode layout). In this regard, producing the individualized headsets becomes highly demanded.

In our previous study [3], a prototype of a computer-aided design (CAD) system for creating dry electrode headsets was developed. However, it had several limitations: the resulting headset could only accommodate the 10-20 electrode layout (also known as the 10-20 system) [4], and the process of constructing a model of the head was only partially automated. In this work, we propose improvements to the developed system, including software tools for describing electrode layouts, managing their use in headset generation, and automating the process of constructing a head model based on a 3D scan of the user's head. The headset model generated by the developed system has been verified by comparison with a factory-made fabric cap included in professional medical EEG equipment.

2. Background and Related Work

Many studies use standard headsets. One such headset is the Ultracortex Mark IV [5]. This headset is not designed to be customized for a particular person, it has only a few fixed sizes from which the most suitable one can be chosen. Popular solutions with the ability to customize the headset to a person include WalkEEG [6] and Spiderclaw [7]. They allow customization of the headset, but this customization still has limitations. In addition, these headsets are not suitable for 3D printing due to complex and large parts, and WalkEEG does not support the 10-20 system. To address this issue, a prototype was developed for generating a headset model with the following capabilities:

1. Create an ellipsoidal model of the human head based on distances between key points: Nasion (the deepest point of the nasal bridge), Inion (the external occipital protuberance), Ear points, or based on head circumferences along three axes.
2. Calculate the coordinates of electrodes according to the 10-20 placement system for the given head model.
3. Place models of electrode sockets, forming a headset.
4. Connect the socket models with bridges.
5. Segment the bridges with "dovetail" connections for later assembly of the headset from separate small parts, optimized for 3D printing with minimal supports.
6. Place text labels on the bridges to simplify the headset assembly process.
7. Export parts into separate STL files to facilitate printing.

The prototype had some limitations, namely:

1. Inability to change the electrode placement system.
2. Requirement for manual measurement of distances between key points, even when a 3D scan of the head was available.

The prototype was developed in Python using FreeCAD's geometric kernel [8].

3. Software Architecture

To overcome the limitations of the first prototype, the corresponding software system architecture is proposed. It is demonstrated in Fig. 1. The Head modelling module gets a 3D head scan and creates a head model for further headset generation. The Placement system creating a module is an editor that allows you to describe any, both standard and modified, electrode placement systems. Finally, the Headset model generating module automatically creates a set of STL models ready to manufacture on a 3D printer. The proposed pipeline of modules overcomes the weaknesses of the first prototype described earlier.

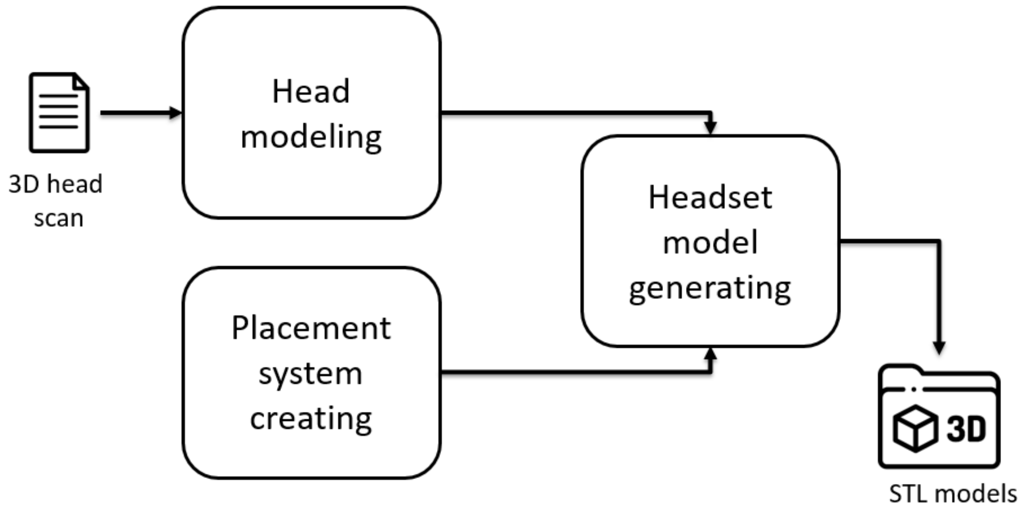

Fig. 1. Proposed architecture of the headset model generation software system

4. Head Modelling

The previous version of the system used an ellipsoidal head model. It is known that such a head model is well-suited for electrode placement [9], and it also simplifies many calculations. For generating the ellipsoidal model, two options were proposed: measuring the head circumference along three axes or performing a 3D scan of the head and measuring the distances between key points directly on it. In the first case, it was necessary to solve a system of ellipse perimeter equations (1) to find the lengths of the ellipsoid's semi-axes:

$$L \approx \pi \left[3(a+b) - \sqrt{(3a+b)(a+3b)}\right] \tag{1}$$

where a, b are the lengths of the semi-axes.

In the second approach, the input values were used directly as the lengths of the ellipsoid's axes. The first approach did not require additional equipment, but its accuracy largely depended on the precision of circumference measurements, and the approximate formula for the ellipse perimeter introduced a small error. The second approach was more accurate, but its precision still depended on the accuracy of measurements; additionally, even using a 3D scan for measuring distances digitally, measurements had to be taken manually using interactive tools in the CAD system.

This work presents an advancement of the second approach, where the model is constructed directly from the 3D scan without intermediate steps from the user, thereby increasing the level of automation and reducing measurement errors.

The following algorithm for model construction is proposed:

1. Load the 3D scan.
2. Identify the positions of key points on the scan.
3. Construct a coordinate system based on the positions of the key points.
4. Cluster points on the 3D scan.
5. Filter out unnecessary clusters.
6. Construct an ellipsoid based on the point cloud.

This algorithm can be divided into three main stages: identifying key points on the scan and constructing the coordinate system (1–3), identifying electrode positions on the scan (4–

5), and finding the lengths of the ellipsoid's semi-axes as the principal components of the resulting point cloud (6).

The first stage currently relies entirely on a third-party software system [10–12]. This system is implemented in MATLAB and offers a set of functions for loading the 3D scan model, obtaining two-dimensional projections of all key points on the scan, determining their coordinates, transforming the coordinates of the projected points into the global coordinate system of the scan, and constructing a new head coordinate system, where the X and Y axes pass through the key points, and the Z axis is perpendicular to them, pointing to the top of the head. The software system provides tools for visualizing the results, and a visualization of the constructed head coordinate system can be seen in Fig. 2.

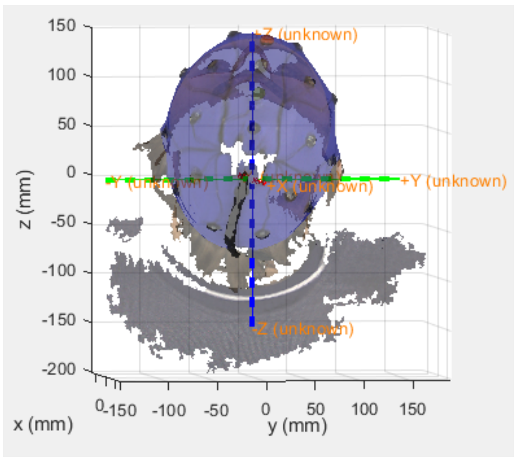

Fig. 2. Coordinate system on a 3D head scan

MATLAB has a Python API, which allows us to integrate this software system into the developed prototype. However, the issue is that MATLAB is not freely available, which may complicate the distribution of our system. In the future, we consider implementing the needed mathematical methods on the side of our software to get rid of the MATLAB dependency.

The second stage is based on the research of Chen S. et al. [12], Shirazi S. et al. [13], and Mazzonetto I. et al. [14], where electrode localization on a scan was performed through clustering. In this work, clustering of the vertices of the 3D scan model is performed using the DBSCAN method [15], which is based on point density. After applying the method, the cluster centres indicate the electrode locations. This method is not ideal for layouts with high electrode density, but for our purposes in BCI, we plan mainly to use the standard 10-20 layout and its derivatives, so this is not an issue.

Next, we transform the coordinates of the obtained points to the coordinate system identified in the first stage, which will simplify the process of finding the ellipsoid in the third stage of the algorithm. From the obtained points, we discard those that fall outside the boundaries defined by the key points, retaining only those within the region between the key points. To achieve this, we create simple clipping planes that define the area between the key points, keeping only points located within this region. The primary purpose of this clipping is to remove clusters corresponding to extraneous objects captured in the scan. Additionally, we filter out outliers: clusters that are either too large or too small. One standard deviation from the mean cluster size is used as the range. The main goal of this stage is to remove clusters representing extraneous objects within the clipping area (such as "tails" of wires from electrodes) and clusters indicating scan artefacts. The result is a point cloud describing the position of the electrodes on the head surface in the scan.

In the third stage, an ellipsoid needs to be found that best approximates the resulting point cloud. For this task, we use the Principal Component Analysis (PCA [16]). This method identifies the directions of maximum data variance. We project the points onto these directions and identify the principal components and their corresponding variances. The directions of the ellipsoid’s axes correspond to the directions of the principal components, and the axes’ lengths are calculated as the square root of the respective variances.

5. Electrode Location Systems Editor

Depending on the objectives of creating a BCI, different headsets with various electrode location systems may be required. There are many standard location layouts, but sometimes also hybrid or unique layouts may be needed. Consequently, the modelling system under development should include an editor that allows users to specify any electrode layout interactively. Essential requirements for such an editor are ergonomic and cognitively accessible graphics based on scientific visualization, ensuring usability by people who are not IT specialists (the target audience for the system includes neurophysiologists interested in conducting BCI research using EEG).

In order to achieve this goal, scientific visualization techniques play a crucial role, allowing for a clear and intuitive representation of the placement and connection of each part of the location system, which contributes to a better understanding and usability. To ensure the editor's graphical elements are cognitively accessible, a conceptual model of the layout must first be developed, identifying components and their interrelations. A convenient way to represent the conceptual model is through an application ontology [17].

The following concepts are defined and described for building the editor ontology:

1. **Point:** An abstract object with a name.
2. **Electrode:** A point where an electrode socket will be placed during frame generation.
3. **Fastener:** A headset element enabling the assembly of the headset from parts. It can be one of two types: socket-to-socket and socket-to-bridge-to-socket.
4. **Bridge:** A headset element with a relative length connecting two points. It can be real (with a fastener) or imaginary (without a fastener).
5. **Arch:** A list of bridges.

The following types of relationships are identified among these concepts:

1. **is_a:** A relationship linking a child concept to a parent concept.
2. **a_part_of:** A relationship between a “part” and the “whole.”
3. **has:** A relationship between an “owner” and the “entity” belonging to it.

The “a_part_of” relationship links a point to a bridge and a bridge to an arch. A bridge can only have two points associated with it. The “has” relationship connects a bridge to a fastener. The “is_a” relationship links the concept of Electrode to the concept of Point.

Fig. 3 shows the application ontology describing relationships between key concepts, created in the Ontolis software system [18].

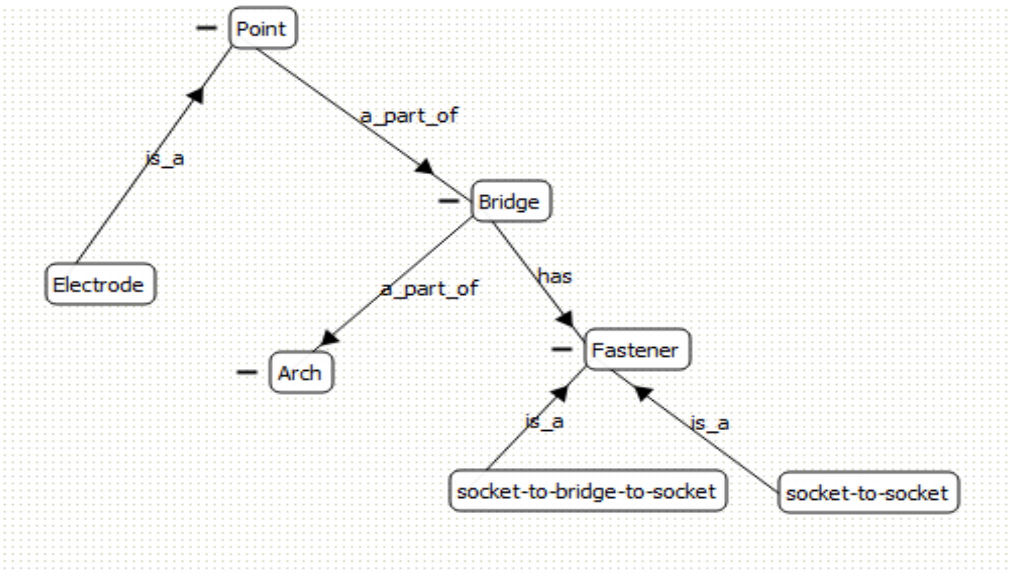

Fig. 3. Ontology of concepts for electrode placement system

To implement the editor, it is necessary to define the appropriate software tools. Python is chosen as the language since the existing system is already written in it. For usability, the editor should provide a graphical user interface, so it is necessary to select an appropriate framework for organizing the interfaces. Table 1 shows a comparison of popular frameworks for developing desktop applications in Python.

Table 1 Comparison of Python frameworks for graphical applications

Framework	Included in the standard library	Well documented	Powerful canvas painting capabilities	Easy to use	Free license
TKinter	+	+	+	+	+
Kivy	-	+	-	+	+
PyQT	-	+	+	-	+/-
wxPython	-	+	+	-	+
PySide	-	+	+	+	+

The TKinter framework was chosen to implement the editor, as it is included in the standard library, is well documented, has powerful features for using the Canvas widget, is easy to use and is freely distributed.

During implementation, three main classes are defined: Point, Electrode, and Bridge. These are used to create the schematic model. The Point class has fields *x*, *y* for placement on the canvas and *radius* to set the display radius of the point, *ID* for identification, *name* as a descriptive field containing the point's name on the schematic, and *base* as a boolean field indicating if the object is a base point. The Electrode class inherits from the Point class but adds no new fields. Objects of this class have a different radius value and do not have the base field. These objects will display differently on the canvas but are only needed logically to distinguish between points and electrodes. The Bridge class includes fields *start_point* and *end_point* as references to the points/electrodes that the bridge connects; *bridge_id* for object identification; *length* to indicate the bridge's length relative to the arch; *arch* to specify which arch the bridge belongs to; *connection_type* to specify the connection type (socket-to-socket, socket-bridge-socket); and *imaginary* flag to indicate if the bridge is real or imaginary. The *length*, *arch*, *connection_type*, and *imaginary* fields are used for calculating socket coordinates, connecting sockets with bridges, and generating fasteners.

Objects of these classes must have certain constraints.

The Point/Electrode classes have the following constraints:

1. The names of points and electrodes must be unique.
2. The system must always include points named Nasion, Inion, LeftEar, and RightEar. They are created with a new document and cannot be deleted or renamed.

The Bridge class has the following constraints:

1. A bridge may or may not have an arch.
2. If a bridge has an arch, the *length* field must contain a value in the range [0, 100].
3. The sum of the *length* values of all bridges in any arch must equal to 100.
4. A bridge cannot belong to an arch unless it has common endpoints with other bridges unless it is the only bridge.
5. If the *imaginary* field of a bridge is *False*, then its *connection_type* field must not be *None*.

The corresponding program module is a graphical editor for electrode location schematics. The main part of the interface is a canvas for drawing the schematic. To the right of the canvas (see Fig. 4), there is a properties panel for the selected element, with a list of tools below: "Select," "Point," "Electrode," and "Bridge." Shortcut keys 1-4 are set up for these tools, respectively.

When selecting the "Point" or "Electrode" tools and clicking on the canvas, a large white circle (when selecting an electrode) or a small black circle (when selecting a point) appears at the click location. When selecting the “Bridge” tool and clicking on any of the circles, it will highlight in blue; then, clicking on any other circle not yet connected to it will create a bridge between them.

When using the “Select” tool and clicking on any circle, it will be highlighted in red, and a text field for the “Name” property will appear in the properties panel. Entering a name in this field will immediately display it on the canvas. Circles can be moved on the canvas using a drag-and-drop gesture.

Clicking on a bridge will also highlight it in red and display its properties. The text field for the “Length” property is restricted to numeric input only, within the range from 0 to 100. If the input does not match this restriction, the value will be reverted to its original state. The “Arch” property is set using a combobox that allows list additions, with neighbouring bridges’ arches suggested as options. If a bridge has no arch, its length is not set. If the bridge’s length is not set or is zero, it will not display on the canvas. The “Imaginary” property is set using a checkbox. The initial value is *False*; if set to *False*, an additional field for “Connection Type” appears among the properties. This is a non-expandable list with “Socket-to-Socket” and “Socket-Bridge-Socket” options. Imaginary bridges are displayed with a dashed line, while real bridges have a solid line.

Pressing the Delete key removes the selected object. If an electrode or point is deleted, all bridges connected to it are also deleted. The top menu includes buttons for “Create New Schematic,” “Load,” “Save,” and “Save As.” Scrolling the mouse wheel zooms the canvas, while holding down the wheel and moving the mouse pans (moves across) the canvas. The schematic is saved to a file in the JSON format.

Fig. 4 shows the editor interface after creating a new placement system.

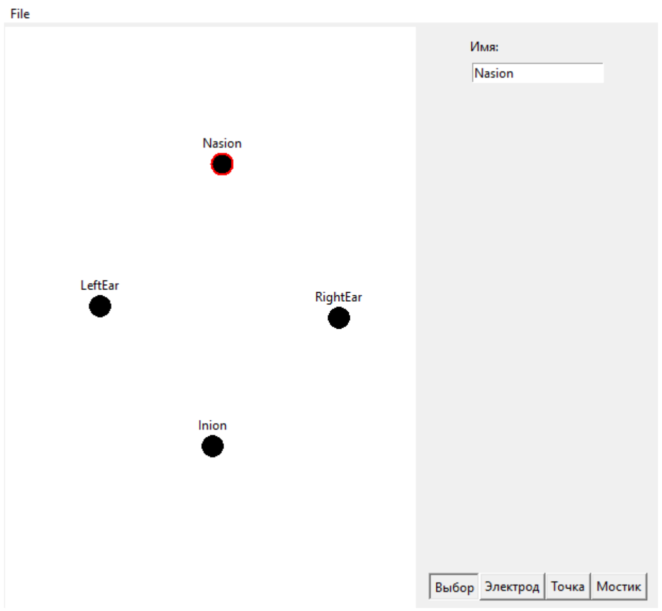

Fig. 4. Interface of the location systems editor

Using the developed editor, the system 10-20 and its modifications are created, which are shown in Fig. 5–7.

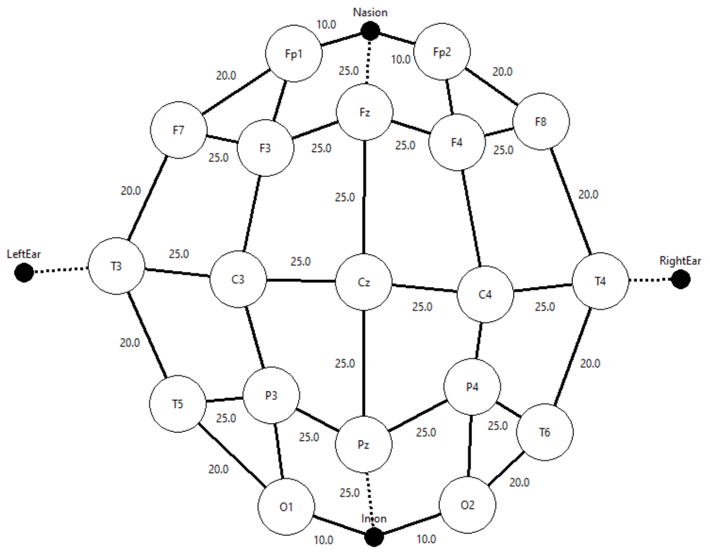

Fig. 5. System 10-20 in the developed editor environment

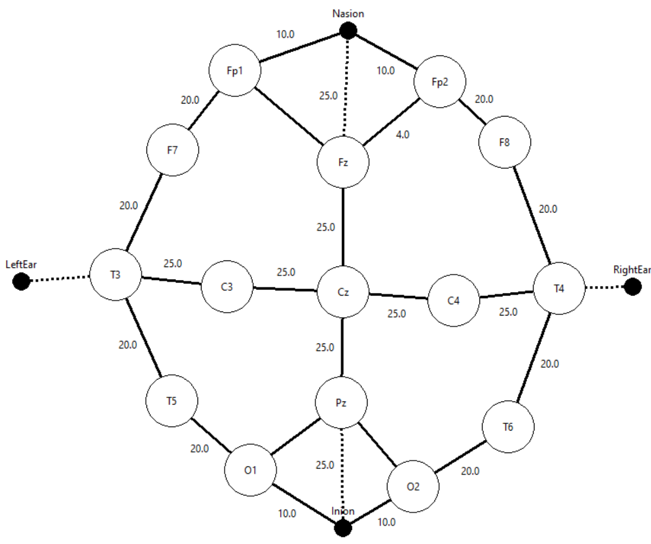

Fig. 6. System 10-20 without the electrodes F3, F4, P3, P4 in the developed editor environment

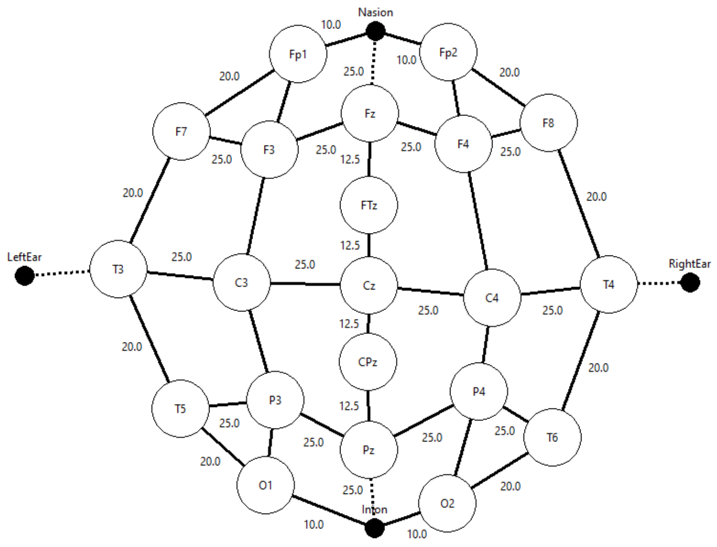

Fig. 7. System 10-20 with the FTz and CPz electrodes in the developed editor environment

6. Headset generation

In the previous version of the headset generation system, the generation of the headset model was based on the 10-20 system only, which was directly implemented in the program code. Now, the generation must be based on a user-defined scheme. The generation process can be divided into the following main stages: calculating the coordinates of electrode sockets, placing them, connecting sockets with bridges, categorizing bridges by fastener type, and placing text labels on the bridges to foster easy assembly. The placement of socket models, the addition of fasteners, and text labels are independent of the location system and therefore will not change compared to the previous headset generation system's implementation.

The coordinate calculation process is based on arcs containing key points. Any location scheme includes four key arches: a longitudinal arch from Nasion to Inion through the top of the head, a perpendicular arch from LeftEar to RightEar (points in front of the left and right ear lobes), and two other arches encircling the head from Nasion to Inion via LeftEar and RightEar. These arches can be identified by the following conditions:

1. The arch contains the points Nasion and Inion and does not contain LeftEar or RightEar.
2. The arch contains LeftEar and RightEar and does not contain Nasion or Inion.
3. The arch contains Nasion, Inion, and LeftEar.
4. The arch contains Nasion, Inion, and RightEar.

Key points can serve as markers if they are connected to an electrode by a bridge of zero length. In such cases, a key arch may contain not the key point itself but an electrode for which this point is a marker. In the created location systems, LeftEar and RightEar act as markers since they are connected to electrodes T3 and T4 by zero-length bridges. For electrodes located on these four arches, their positions are calculated using the parametric equation of an ellipse (2) as points on ellipses in the principal sections of an ellipsoid.

$$\begin{cases} x = \ a \cos(t) \\ y = \ b \sin(t) \end{cases} \tag{2}$$

where a, b are the semi-axis lengths, and t is the parameter.

The coordinates of points located on the other arcs can be calculated based on already determined points. For each of these, an arc is identified among the incident bridges where points with known coordinates are found on either side of the current point. If no such points exist, an error is generated explaining for which point no reference points could be found. If such points are found, the relative distances from the current point to the known points are calculated, and the proportion in which the current point divides the section of the arc between the two known points is determined. Then, the coordinates of this point can be found by interpolation on the surface of the ellipsoid.

To solve the interpolation problem, the minimize method from the scipy.optimize package is used [19].

The overall coordinate calculation algorithm is presented in Listing 1 using pseudocode.

```
def calculate_coordinates(scheme, ellipsoid):
    nasion, inion, left_ear, right_ear = get_key_points(scheme)
    key_arcs = get_key_arcs(scheme)
    coordinates = {}
    for arc in arcs:
        for point, length in arc.enumerate_points():
            coordinates[point] = arc.get_coordinates(ellipsoid, length)
    for point in scheme.get_points_not_in_list(coordinates):
        coordinate_calculated = False
        for arc in point.get_arcs():
            point1, length1 = arc.get_known_neighbour_1(point)
            point2, length2 = arc.get_known_neighbour_2(point)
            if point1 and point2:
                coordinate_calculated = True
                coordinates[point] = interpolate(ellipsoid, coordinates[point1], \
                                     coordinates[point2], length1 / (length1 + length2))
                break
        if not coordinate_calculated:
            raise Exception()
    return coordinates
```

Listing 1. Algorithm for calculating sockets' coordinates

The previous version of the prototype relied on a fixed list of bridges to generate them. In the current version, instead of a fixed list of bridges, it is sufficient to use a list obtained from the location system file to implement different location systems.

7. Results

For the generated electrode placement schemes, frame models were created, which can be seen in Fig. 8–10. These figures show that all scheme features have been accounted for, with the sockets positioned and connected accordingly. The use of scientific visualization techniques allows the generated model to be verified in a CAD system, which facilitates verification and refinement of electrode locations.

Table 2 shows the placement error for the sockets that deviated the most from their positions on the 3D scan when using the old and new versions of our headset generation system. The table demonstrates that the placement error has significantly decreased. It is worth noting that even the errors in the old version were acceptable for BCI tasks. For more accurate testing, it is necessary to collect more 3D scans and conduct additional tests.

Table 2 Electrode placement error

Electrode	Old version deviation, cm	New version deviation, cm
F4	0.5	0.42
Fz	0.6	0.46
T4	0.68	0.5
T8	0.71	0.45

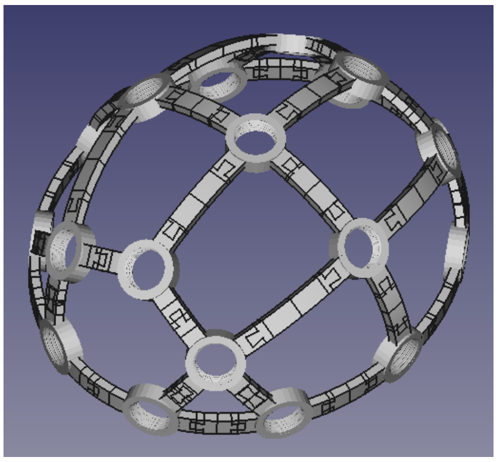

Fig. 8. 3D-model for the system 10-20 in the FreeCAD environment

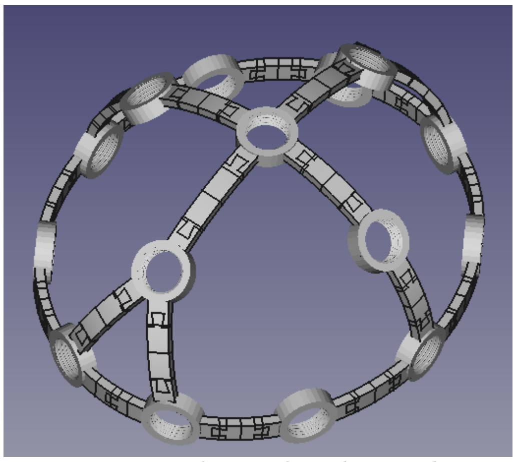

Fig. 9. 3D-model for the system 10-20 without the electrodes F3, F4, P3, P4 in the FreeCAD environment

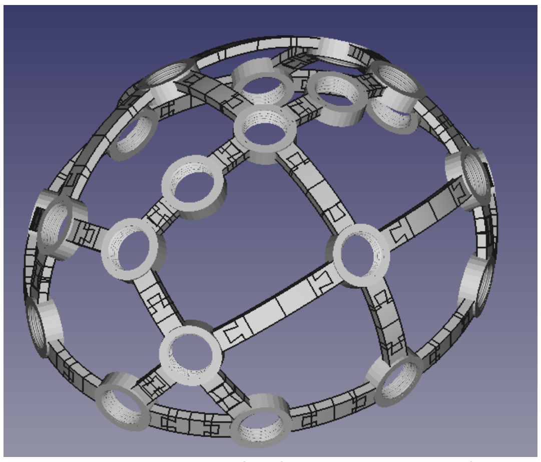
Fig. 10. 3D-model for the system 10-20 with the FTz, CTz electrodes in the FreeCAD environment

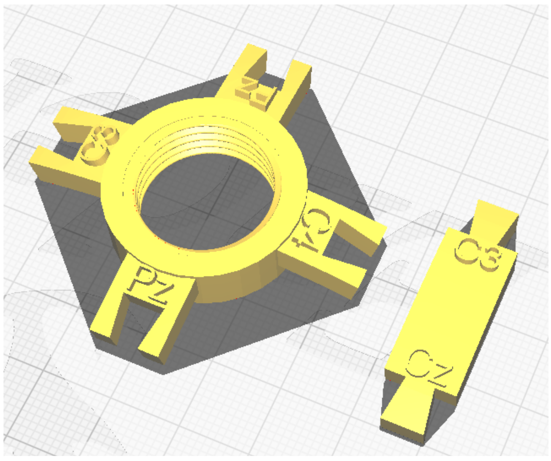

Fig. 11. STL models ready for printing in the UltiMaker Cura environment

The generated models are exported to a set of STL files ready for 3D printing. An example of STL models can be seen in Fig. 11.

8. Conclusion

This work proposes further development of a previously created system for generating dry electrode headsets for EEG powered by scientific visualization. A new method is proposed and implemented to automate the construction of a human head model for electrode placement based on a 3D scan. An ontological model for a visual editor of electrode location systems is introduced; this editor is implemented as a separate software module within the developed system. Based on a formal model of the relevant subject area, the editor's interface incorporates scientific visualization techniques and cognitive graphic properties, making it intuitive for users, even those without IT specialist qualifications. Using this editor, some commonly used electrode location systems are designed, and the corresponding electrode headsets are generated. The output is a set of STL models automatically optimized for 3D printing. The optimization includes the automatic division of the frame, which is ellipsoid-like in shape, into dismantlable elements that are close to planar. This minimizes material consumption by eliminating the need to print large support structures. Thus, a production cycle is established for personalized EEG electrode headsets based on additive manufacturing, efficient in both cost and time.

For further development, more extensive testing with a larger dataset is required to determine electrode placement accuracy, and laboratory testing of the headset for BCI applications is also necessary.

References

1. Mridha M., Chandra S., Moshin M., Akter A., Rashedul M., Watanobe Y. Brain-Computer Interface: Advancement and Challenges // Sensors. 2021. Vol. 17. pp. 1–46.
2. Jamil N., Belkacem A., Ouhbi S., Lakas A. Noninvasive Electroencephalography Equipment for Assistive, Adaptive, and Rehabilitative Brain–Computer Interfaces: A Systematic Literature Review // Sensors. 2021. Vol. 14. pp. 1–31.
3. Chiruhin D. A., Ryabinin K. V. Generaciya 3D-modeli personalizirovannogo karkasa dlya suhih elektrodov elektroencefalografa [Generation of a 3D model of a personalized headset for dry electrode electroencephalograph electrodes] // Trudy konferencii GrafiKon – 2023. 2023. pp. 417–426. [in Russian]
4. Klem G. H., Lüders H. O., Jasper H. H., Elger C. The ten-twenty electrode system of the International Federation // Electroencephalography and Clinical Neurophysiology. 1999. Vol. 52. pp. 3–6.
5. OpenBCI Ultracortex Mark IV. Accessed: July, 22st, 2024. [Online]. Available: https://docs.openbci.com/AddOns/Headwear/MarkIV
6. WalkEEG. Accessed: July, 22st, 2024. [Online]. Available: https://www.stlfinder.com/model/walkeeg-headset-opensource-eeg-hes1eL8Q/343938
7. SpiredClaw. Accessed: July, 22st, 2024. [Online]. Available: https://openbci.com/community/spiderclaw-v2-deprecated
8. FreeCAD (version 0.20.2)[software]. Accessed: July, 22st, 2024. [Online]. Available: https://www.freecad.org/
9. Koessler L., Cecchin T., Eric T., Maillard L. 3D handheld laser scanner based approach for automatic identification and localization of EEG sensors // Annual International Conference of the IEEE Engineering in Medicine and Biology. 2010. pp. 211–252.
10. Martinez E., González A., Garea-Llano E., Bringas-Vega M., Valdes-Sosa P. Automatic Detection of Fiducial Landmarks Toward the Development of an Application for Digitizing the Locations of EEG Electrodes: Occipital Structure Sensor-Based Work // Frontiers in Neuroscience. 2021. Vol. 15. pp. 1–14.
11. Homölle S., Oostenveld R. Using a structured-light 3D scanner to improve EEG source modeling with more accurate electrode positions // Journal of Neuroscience Methods. 2019. Vol. 326. pp. 1–8.
12. Chen S., He Y., Qiu H., Yan X. Spatial localization of eeg electrodes in a TOF+CCD camera system // Frontiers in Neuroinformatics. 2019. Vol. 13. pp. 21–27.
13. Shirazi S., Huang H. More Reliable EEG Electrode Digitizing Methods Can Reduce Source Estimation Uncertainty, but Current Methods Already Accurately Identify Brodmann Areas // Frontiers in Neuroscience. 2019. Vol. 13. pp. 1–14.
14. Mazzonetto I., Castellaro M., Cooper R., Brigadoi S. Smartphone-based photogrammetry provides improved localization and registration of scalp-mounted neuroimaging sensors // Scientific Reports. 2022. Vol. 12. pp. 1–14.
15. Schubert E., Sander J., Ester M., Kriegel H. P., Xu X. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN // ACM Transactions on Database Systems. 2017. Vol 42. pp. 1–21.
16. Gorban A., Kegl B., Wunsch D., Zinovyev A. Principal Manifolds for Data Visualisation and Dimension Reduction. Berlin: Springer, 2008. 300 p.
17. Gruber T. R. The role of common ontology in achieving sharable, reusable knowledge bases // Principles of Knowledge Representation and Reasoning. 1991. pp. 601–602.
18. Chuprina, S. I., Zinenko, D. V. Ontolis: adaptiruemyj vizual'nyj redaktor ontologij [Ontolis: an adaptable visual ontology editor] // Vestnik Permskogo universiteta. Seriya: Matematika. Mekhanika. Informatika. 2013. № 22. pp. 106–110. [in Russian]
19. Virtanen P., Gommers R., Oliphant T. E., Haberland M. et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python // Nature Methods. 2020. Vol 17. pp. 261–272.