Software Tools and Analysis Methods for the Use of Electromagnetic Articulography Data in Speech Research

Andrew Kolb
Marquette University

SOFTWARE TOOLS AND ANALYSIS METHODS FOR THE USE OF ELECTROMAGNETIC ARTICULOGRAPHY DATA IN SPEECH RESEARCH

by

Andrew J. Kolb, B.S.

A Thesis submitted to the Faculty of the Graduate School,
Marquette University,
in Partial Fulfillment of the Requirements for
the Degree of Master of Science

Milwaukee, Wisconsin

May 2015

ABSTRACT

SOFTWARE TOOLS AND ANALYSIS METHODS FOR THE USE OF ELECTROMAGNETIC ARTICULOGRAPHY DATA IN SPEECH RESEARCH

Andrew J. Kolb, B.S.

Marquette University, 2015

Recent work with Electromagnetic Articulography (EMA) has shown it to be an excellent tool for characterizing speech kinematics. By tracking the position and orientation of sensors placed on the jaws, lips, teeth and tongue as they move in an electromagnetic field, information about movement and coordination of the articulators can be obtained with great time resolution. This technique has far-reaching applications for advancing fields related to speech articulation, including recognition, synthesis, motor learning, and clinical assessments.

As more EMA data becomes widely available, a growing need exists for software that performs basic processing and analysis functions. The objective of this work is to create and demonstrate the use of new software tools that make full use of the information provided in EMA datasets, with a goal of maximizing the impact of EMA research. A new method for biteplate-correcting orientation data is presented, allowing orientation data to be used for articulatory analysis. Two examples of applications using orientation data are presented: a tool for jaw-angle measurement using a single EMA sensor, and a tongue interpolation tool based on three EMA sensors attached to the tongue. The results demonstrate that combined position and orientation data give a more complete picture of articulation than position data alone, and that orientation data should be incorporated in future work with EMA.

A new standalone, GUI-based software tool is also presented for visualization of EMA data. It includes simultaneous real-time playback of kinematic and acoustic data, as well as basic analysis capabilities for both types of data. A comparison of the visualization tool to existing EMA software shows that it provides superior visualization and comparable analysis features to existing software. The tool will be included with the Marquette University EMA-MAE database to aid researchers working with this dataset.

In providing tools for increased research productivity, solutions to complex questions in speech production can be found more quickly, with major benefits for individuals requiring speech training and rehabilitation.

ACKNOWLEDGEMENTS

Andrew J. Kolb, B.S.

I have many people to thank for their help with this work, and more broadly, the work throughout my Marquette years.

I thank my research advisor, Dr. Michael Johnson, for recruiting me to work on the Senior Design team and giving me the opportunity to work on this project. His advice and leadership have been instrumental in my development as a young researcher.

I thank Dr. Jeff Berry, for sharing his enthusiasm, expertise, and most importantly, life experience. The breadth and depth of knowledge I have gained in his lab are truly invaluable.

I thank my academic advisor, Dr. John LaDisa, who has fostered my growth as an individual in so many ways throughout my five years at Marquette. I could not have achieved my goals here without his wisdom and guidance.

I thank my fellow lab members, who have provided a great deal of knowledge and much needed company during long hours of work.

I thank my close friends and family, each of whom have played an integral role in my development as a student and as a man.

And most importantly, I thank my wife Katie, who is my biggest critic, my strongest supporter, and my best friend. I appreciate her patience and understanding during all my long nights of writing. This work would not exist without her.

TABLE OF CONTENTS

1. INTRODUCTION AND RESEARCH OBJECTIVES

1.1 General Background

Over the past few decades, electromagnetic articulography (EMA) has emerged as a valuable tool in speech research applications, ranging from speech modeling and synthesis to recognition and motor learning [1-4]. EMA operates by attaching sensors to the organs used in articulation (tongue, lips, jaw, teeth, etc.), and tracking them in a small electromagnetic field that surrounds an individual's head. At a basic level, EMA provides a means for acquiring kinematic information about these organs, including position, speed, acceleration, and range of motion [5, 6].

This type of information makes EMA ideally suited for several applications, both in and out of the speech clinic. Clinically, EMA presents an opportunity for the creation of objective measurements in assessments of motor speech disorders. Replacing subjective acoustic assessments with objective kinematic measurements from EMA would result in increased accuracy and precision in diagnoses, as well as improve the ability of clinicians to assess degrees of disorder. This principle could also be applied to non-speech related movements, like those in chewing.

Similarly, EMA data provides a means of determining differences in articulation between native and non-native speakers in a population. These differences can be quantified and used to inform pronunciation training for non-native speakers to increase the naturalness of their speech and improve intelligibility.

Though largely experimental at present, EMA also shows promise in speech rehabilitation. The Marquette University Speech and Swallowing Lab is currently experimenting with an EMA-driven speech synthesizer, with a goal of using a virtual vocal tract for speech motor rehabilitation. The system would be useful in a clinical setting for helping retrain speakers with dysarthria, including survivors of traumatic brain injury.

Given the utility of EMA for understanding and improving articulation, several large datasets have been collected from various speaker populations [7-9]. One of the most recently released databases is the Marquette University Electromagnetic Articulography Mandarin Accented English (EMA-MAE) corpus, collected by Ji, Berry and Johnson [10]. The corpus consists of 40 speakers, with 10 representatives each from native English-speaking males, native English-speaking females, native Chinese-speaking males, and native Chinese-speaking females. All subjects were recorded for roughly 30 minutes of speech, performing various speech tasks in English. The tasks included individual word repetitions, individual sentences, and full paragraphs and passages.

The final corpus includes the resulting kinematic data, acoustic data, phonetic transcriptions of all speech, pause labels, and associated kinematic data needed for subject-specific calibration. This comprehensive collection of data associated with subject speech is the database's greatest strength, as it facilitates research in a diverse group of areas related to accents and word pronunciation.

However, because of its comprehensive nature, the EMA-MAE database is also large (50+ gigabytes) and creates a challenge for many researchers to extract and analyze the data that is relevant to their work. The large group of speech researchers who could benefit from the data come from widely varying backgrounds and have differing levels of technical skill. There is thus a great challenge to make the database accessible to researchers of varying technical abilities and interests.

The complexities of EMA processing are not just a problem for the EMA-MAE database, but for EMA data generally. After acquisition, a series of complicated post-processing tasks are required to make the data more physiologically meaningful. Segments of speech that are relevant to a research question (often only hundreds of milliseconds) must be precisely labeled and extracted from entire records (usually minutes long). Analyses require at least a basic knowledge of computer programming. This process makes EMA-based research complicated and challenging, especially for those without software proficiency.

1.2 Objective of Research

The objective of this thesis is to create and demonstrate the use of new software tools that make full use of the information provided in EMA datasets. Using EMA data to its fullest extent includes the development of new methods for analyzing or visualizing EMA data. The tools created as part of this work aid in the post-processing of EMA data, use EMA data in new articulatory representations, and allow for simple visualization and data processing. These tools will be released along with the EMA-MAE database so researchers who use the database will have the ability to process and analyze a large amount of data in an efficient manner.

In addition to developing and distributing software tools, the thesis introduces and demonstrates the utility of orientation data in EMA. Alongside basic position data, modern EMA systems provide information about the orientation of a sensor in the electromagnetic field. In a typical speech kinematic study, the position data are used to support the research findings, but the orientation data are not included in any meaningful way. One reason this might be is that orientation data are presented in an unfamiliar format, using quaternion notation. To facilitate more work with the orientation data, quaternions are elucidated with a brief tutorial and put to use in several applications. It is demonstrated that orientation data can be used in tandem with the position data to give a more complete characterization of articulation than through position data alone.

1.3 Organization of Thesis

The thesis consists of five chapters and two appendices. Chapter 2 provides an overview of EMA systems, including the operating principle behind EMA, a brief history of EMA technologies, and a description of the data that EMA sensors provide. It also includes some of the basic details of EMA data post-processing (including biteplate correction), while introducing a new method for correcting quaternion orientation data. Chapter 3 provides two applications that use EMA position and orientation data for articulatory representations. Section 1 of Chapter 3 demonstrates EMA's utility in jaw-angle tracking, and Section 2 details the creation of a tongue mesh using three EMA sensors. Chapter 4 presents a new software tool for visualizing and analyzing EMA data. This includes a complete feature description, as well as comparisons to existing EMA processing tools. Chapter 5 summarizes the thesis and places the results of each of the chapters in a broader context. The appendices contain a description of the code used for implementing each of the software tools, and give a description of other related tools that are used in MU Speechlab experiments with EMA data.

2. OVERVIEW OF EMA SYSTEMS AND DATA STRUCTURE

2.1 EMA Systems Overview

Electromagnetic articulography (EMA) provides a method of tracking the movement and orientation of sensors in 3-dimensions as they move through an electromagnetic field. To study positioning and coordination of movements during speech, these sensors can be rigidly attached to the articulators and their position and orientation can be reported over time.

2.1.1 EMA History

In its original form, EMA provided data comparable to earlier, existing x-ray microbeam systems [11], which utilized gold pellets attached to the articulators to provide 2D midsagittal position data over time. By placing a subject's articulators and the attached pellets between an x-ray generator with a pinhole and an x-ray detector, the trajectory of the pellets could be tracked by finding local minima in x-ray intensity on the 2D plane [12]. The x-ray microbeam system allowed for some of the first large-scale data collection projects (e.g. Westbury [13]), which provided speech kinematic information across a wide variety of speakers and speech contexts. However, because the method required exposure to x-ray radiation, there were risks to subject health that had to be weighed against the benefits of the research. Because EMA poses no proven risk from radiation, it emerged after x-ray microbeam as the best, most similar alternative for articulatory position tracking.

One of the first major EMA systems developed and described in the literature is the EMMA (ElectroMagnetic Midsagittal Articulometer) system, created by Perkell *et al.*[14] at Massachusetts Institute of Technology. The apparatus consisted of three magnetic-field transmitter coils capable of tracking solenoidal receiver coils placed along the midsagittal plane of the vocal tract. With the subject's head fixed in place and the receiver coils oriented parallel to the transmitter coils, the induced voltage in a given coil varies inversely with the cubed-root of the distance between the transmitter and receiver [14]. This system was capable of resolutions of between 0.5 and 1 mm, but had major limitations, as the sensors had to be placed and oriented carefully in the midsagittal plane to avoid rotational misalignment, which caused faulty position measurements. This problem in turn meant any movement by the subject's articulators perpendicular to the midsagittal plane resulted in erroneous positional measurements. The system was also large, complicated and immobile, making it difficult to emulate experiments with a large number of subjects, or in other laboratories.

2.1.2 Modern (3D) EMA

However, the experiments done with the EMMA system laid the groundwork for future development of modern day 3D EMA. To move from a 2D system to a 3D system, Zierdt, Hoole and Tillmann created a setup with 6 transmitter coils positioned on a sphere around a subject's head [15]. The total induced voltage in a single receiver coil is a function of its x, y, and z position, as well as its 2D orientation (as shown with rotational misalignment error), with respect to each transmitter. The result is 6 equations (one for each transmitter) and 5 unknowns (one for each position/orientation variable of the receiver), which can be solved numerically to determine the position and orientation of the sensor in the field. Thus, in moving from 2D positional data to 3D positional data, two additional rotational degrees of freedom were added to the known information about the receivers. The transmitters-on-a-sphere technique formed the basis for the creation of one of the two major EMA systems widely used today, the Carstens AG-series of articulographs (Carstens Medizinelectronik, Lenglern, Germany).

Modern commercial EMA systems include the NDI Wave Speech Research System (NDI; Waterloo, Ontario, Canada) and the Carstens AG500 and AG501 (Carsens Medizinelectronik, Lenglern, Germany). As described, both systems are capable of reporting positional data in 3 dimensions, as well as rotation about the transverse axis and anterior-posterior axis. The Carstens AG-500 system (described and characterized in detail by Yunusova *et al.* [16]) is a project whose design and measurement principles are publicly available. The system consists of 6 transmitter coils attached along the edges of plexiglass cube, with the subject and receiver coils positioned inside of the cube during data collection. The Carstens AG500 system can track up to 12 sensors at 200 Hz.

The NDI Wave, in contrast to the Carstens system, is a proprietary EMA system (detailed and characterized more completely in Berry [17]) consisting of a data collection unit and a small box containing the transmitter coils. The standard unit allows for tracking of 8 sensors in one of two available electromagnetic field sizes (300 mm^3 or 500 mm^3) at 100 Hz. The Wave specifications state that it is accurate within 0.5 mm, which is within the acceptable tolerance for meaningful analysis of speech kinematics [18]. The standard unit can be upgraded to increase the sampling rate to up to 400 Hz, along with a second data collection unit capable of obtaining data from an additional 8 sensors. The NDI Wave with this upgrade is the EMA unit used by the Marquette University Speech and Swallowing Lab.

2.2 EMA Data Format and Quaternions

As described, when data are reported from the Wave, each sensor provides information about its position in 3D space, as well as its 2D orientation. The position data comes in a familiar X, Y, Z format, but the orientation is given in the form of a quaternion. Because quaternions are a rarely used data type outside of computer graphics, aviation and controls, few speech researchers are familiar with what they are and how to adequately make use of them. The result is a dearth of papers and findings relating how orientation can be used to more completely characterize articulator movements. Hoole, Zierdt and Geng [19] noted that using two tongue sensors that provide orientation and position data provides the equivalent amount of information as four sensors reporting only 2D (x and y) position data. This means in many cases where orientation data is neglected, researchers are discarding a significant portion of the information provided by each sensor. With this in mind, a tutorial to gain a basic understanding of quaternions is provided. For a more detailed, rigorous description (including proofs for a number of the equations presented), see Hamilton's original work on quaternions [20], or [21].

2.2.1 Quaternions Explained

Quaternions are 4-dimensional numbers used to represent rotations and orientations, often used because of their compactness compared to Euler matrices and avoidance of gimbal lock [22]. Quaternions, along with their discoverer, Sir William Rowan Hamilton, have a fascinating history, described in [23] and in great detail in [24]. In the 1830's, Hamilton began working with complex numbers and in particular their applications toward vector algebra. At this time, vectors in the modern sense did not exist, and it was actually in Hamilton's quaternion-product formulation that dot products and cross products were first described.

By the early 1840's, Hamilton believed he found a method for multiplying and dividing vectors using quaternions. His original discovery, however, was filled with misconceptions and errors about how quaternions represented and operated on vectors, a problem which helped guide quaternions into their modern mathematical obscurity. Another mathematician at the time, Olinde Rodrigues, arrived at a similar (and actually more conceptually correct) formulation of quaternions almost simultaneously with Hamilton, but his work was largely ignored by mathematicians at the time.

Before diving into quaternions, a brief mathematical review of complex numbers is required. A complex number in algebra is any number:

$$c = a + b\boldsymbol{i} \tag{2.2.1}$$

where, a and b are real numbers and $\boldsymbol{i} = \sqrt{-1}$. Graphically, complex numbers are often represented in the complex plane. The example of $c = 3 + 4\boldsymbol{i}$ is plotted in Figure 2.1.

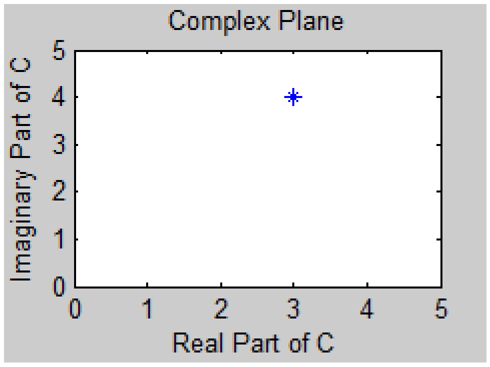


Figure 2.1- *Representation of a complex number in the complex plane.*

Complex numbers behave intuitively for addition and subtraction:

$$(a + b\boldsymbol{i}) + (c + d\boldsymbol{i}) = (a + c) + (b + d)\boldsymbol{i} \tag{2.2.2}$$

And follow distributive laws for multiplication:

$$(a + b\boldsymbol{i})(c + d\boldsymbol{i}) = (ac - bd) + (ad + bc)\boldsymbol{i} \tag{2.2.3}$$

The conjugate of a complex number is found by simply changing the sign of the imaginary component:

$$c^* = a - b\boldsymbol{i} \tag{2.2.4}$$

The notion of rotors in the complex plane provides a simple introduction to the idea of using complex numbers to rotate vectors, as is done with quaternions. A rotor is a complex number that rotates another complex number in the complex plane by an angle θ about the origin. It is represented generally as:

$$r = \cos(\theta) + \sin(\theta)\boldsymbol{i} \tag{2.2.5}$$

As an example, imagine the complex number plotted in Figure 2.1 as a vector pointing from the origin to the point $3 + 4\boldsymbol{i}$. This vector can be rotated counter-clockwise 30 degrees by multiplying by:

$$\cos(30)+\sin(30)\boldsymbol{i}=\frac{\sqrt{3}}{2}+\frac{1}{2}\boldsymbol{i} \tag{2.2.6}$$

yielding:

$$(3+4\boldsymbol{i})(\frac{\sqrt{3}}{2}+\frac{1}{2}\boldsymbol{i})=0.60+4.96\boldsymbol{i} \tag{2.2.7}$$

Plotting the two vectors demonstrates the 30 degree rotation:

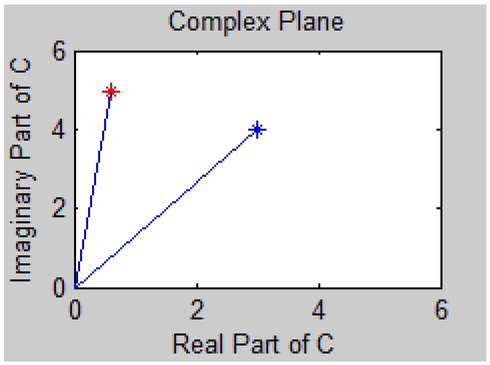


Figure 2.2- *Demonstration of 30 degree rotation by a rotor.*

With this basic understanding of rotations in the complex plane, the same concept can be generalized to higher dimensions, leading to the development of the quaternion.

Formally defined, a quaternion is a complex number of the form:

$$q=w+x\boldsymbol{i}+y\boldsymbol{j}+z\boldsymbol{k} \tag{2.2.8}$$

where w,x,y and z are real numbers, and $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ are imaginary numbers satisfying the equation:

$$\boldsymbol{i}^2=\boldsymbol{j}^2=\boldsymbol{k}^2=\boldsymbol{ijk}=-1 \tag{2.2.9}$$

The parallel to ordinary complex numbers and rotors in the complex plane can be seen clearly in this form. The large difference is that quaternions add two extra imaginary dimensions, allowing for rotation in 3D space and increasing the complexity of their behavior.

Another way of conceptualizing quaternions is as an ordered pair, consisting of a real number portion and a vector portion, as follows:

$$q=[w,\boldsymbol{v}] \tag{2.2.10}$$

where w is the real portion and $\boldsymbol{v}$ is a vector with $\boldsymbol{i}$, $\boldsymbol{j}$, and $\boldsymbol{k}$ components. This conception is useful when thinking about rotations. Similar to rotors in the complex plane, a quaternion can be viewed as an angle-axis pair, where the imaginary quaternion components define an axis about which another vector or point will be rotated. Specifically, the equation is:

$$q = [\cos(\frac{\theta}{2}), \sin(\frac{\theta}{2})(x\boldsymbol{i} + y\boldsymbol{j} + z\boldsymbol{k})] \tag{2.2.11}$$

where θ is the angle of desired rotation, and x, y and z define the axis about which to rotate. Before establishing how the rotation is done, however, quaternion mathematics must be further explained.

Quaternions add and subtract in the same way as ordinary complex numbers, but there is a major difference when it comes to quaternion multiplication. First, quaternion multiplication is not commutative; when multiplying out quaternion components, order matters. The quaternion multiplication rules are shown below:

$$\begin{aligned} ij &= k \\ jk &= i \\ ki &= j \\ ji &= -k \\ kj &= -i \\ ik &= -j \end{aligned} \tag{2.2.12}$$

It should be noted that these rules are the same as the vector cross product, which is often used when working with vectors. Using these same rules, distributing over all parts of the quaternion, and rearranging to combine components, the quaternion product can be defined as follows:

$$q_1q_2 = [w_1w_2 - \boldsymbol{v}_1 \bullet \boldsymbol{v}_2, w_1\boldsymbol{v}_2 + w_2\boldsymbol{v}_1 + \boldsymbol{v}_1 \times \boldsymbol{v}_2] \tag{2.2.13}$$

Beyond addition, subtraction and multiplication, three other basic calculations are required for working with quaternions, the conjugate, the norm, and the inverse. A quaternion conjugate is similar to the conjugate of an ordinary complex number. However, instead of negating only a single imaginary number, all three imaginary components must be negated:

$$q^* = w - x\boldsymbol{i} - y\boldsymbol{j} - z\boldsymbol{k} \tag{2.2.14}$$

Another important definition is the norm of a quaternion. The norm is calculated in the same way as the L^2 norm for any 4-dimensional vector, as follows:

$$norm(q) = \sqrt{w^2 + x^2 + y^2 + z^2} \tag{2.2.15}$$

It is desirable to work with normalized quaternions, as they have many characteristics that make them easy to work with. As an example, here is the definition of the inverse of a quaternion:

$$q^{-1} = \frac{q^*}{norm(q)} \tag{2.2.16}$$

For a normalized quaternion, the norm is magnitude 1, and the quaternion inverse is the same as the quaternion conjugate [22]. Because quaternion inverses are ubiquitous when doing rotation calculations, working with normalized quaternions reduces computational complexity to a large degree. Also, when rotating a vector, the magnitude of the vector being rotated should be maintained after rotation by a quaternion. Using a unit-length (normalized) quaternion ensures that vector magnitude is left untouched during the rotation. As a rule, normalized quaternions should be used exclusively to do rotations.

2.2.2 Quaternion Rotation Example

With the basic tools for quaternion rotation in place, the mechanism for quaternion rotation of vectors can be shown through an illustrative example. Imagine a unit-vector that points directly down the x-axis, $[1,0,0]$. To rotate to the y-axis, $[0,1,0]$, a positive 90 degree rotation is required about the positive z-axis. Recall that to build a quaternion that will rotate a vector by θ about an axis requires:

$$q = [\cos(\frac{90}{2}), \sin(\frac{90}{2})(0\boldsymbol{i} + 0\boldsymbol{j} + 1\boldsymbol{k})] = [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\boldsymbol{k}] \tag{2.2.17}$$

To rotate a vector, a quaternion must be multiplied by the vector, using the sandwich product:

$$\boldsymbol{v}_{rotated} = q\boldsymbol{v}q^{-1} \tag{2.2.18}$$

where q is the quaternion representing the desired rotation, q^{-1} is the quaternion inverse, and $\boldsymbol{v}$ is the vector represented as a pure quaternion. Representing a vector as a pure quaternion requires that the vector's x, y and z components are represented as the quaternion $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ components. Using the previous example, the sandwich product becomes:

$$\begin{aligned} \boldsymbol{v}_{rotated} &= [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\boldsymbol{k}][0, 1\boldsymbol{i} + 0\boldsymbol{j} + 0\boldsymbol{k}][\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}\boldsymbol{k}] \\ \boldsymbol{v}_{rotated} &= [0, 1, 0] \end{aligned} \tag{2.2.19}$$

Note how the vector being rotated is shown as a pure quaternion, and how the inverse of the quaternion was simply formed by taking the conjugate of the original quaternion. There are, however, a few nuances of the quaternion rotation that merit further explanation. First, the rotation from one vector to another is not unique. In the example shown, the x-axis was rotated to the y-axis using a 90-degree rotation. A 270-degree rotation in the opposite direction would have resulted in the same net rotation, as would a 180-degree rotation about the axis $[1,1,1]$. In fact, there are an infinite number of ways to represent the rotation between two points. Secondly, the rotation from this example is called a point rotation. In this kind of rotation, the axes of the coordinate system, or the frame of the coordinate space, remain stationary and the vector rotates from one orientation to another. If the quaternions in the sandwich product are switched, the result is a frame rotation:

$$\boldsymbol{v}_{rotated} = q^{-1}\boldsymbol{v}q \tag{2.2.20}$$

$$\begin{aligned} \boldsymbol{v}_{rotated} &= [\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}\boldsymbol{k}][0, 1\boldsymbol{i} + 0\boldsymbol{j} + 0\boldsymbol{k}][\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\boldsymbol{k}] \\ \boldsymbol{v}_{rotated} &= [0, -1, 0] \end{aligned} \tag{2.2.21}$$

The effect is a rotation in the opposite direction, which acts as if the vector was held still while the frame of the coordinate space rotated around it. This can be a useful tool for rotating coordinate axes.

2.2.3 Quaternions to Represent Orientations

The use of quaternions in representing the orientation of an EMA sensor can now be better understood. Quaternions, as demonstrated, represent rotations between two vectors. As such, a quaternion on its own cannot represent an absolute orientation of a sensor. Using quaternions to represent sensor orientation means they actually represent the rotation required to produce a given sensor's current orientation from a baseline orientation.

In data provided by the NDI Wave, the quaternion data represent the rotation required to rotate a vector that is normal to the XY plane (pointing along the positive z-axis), to the current sensor norm vector orientation. Thus, by taking the vector $[0, 0, 1]$ and using the sandwich product to multiply it by the sensor's quaternion data, the current sensor norm vector orientation can be found (Figure 2.3).

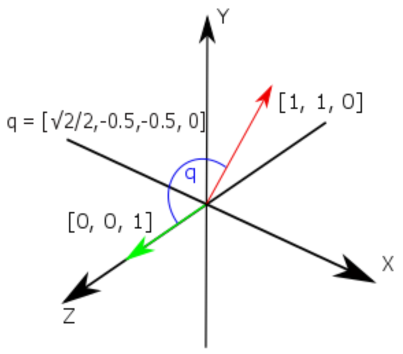


Figure 2.3- *Illustration representing the orientation of the [1, 1, 0] vector via a quaternion. The [0, 0, 1] vector is used as a baseline in data provided by the NDI Wave. Multiplying q by [0, 0, 1] results in the direction given by [1, 1, 0].*

2.3 EMA Data Post-Processing

While providing the position and quaternion data, the Wave provides two different reference spaces for the data to be presented in. The first is in a global coordinate system, where the x, y and z position are provided with respect to the face of the transmitter box. The origin of the space is located at the center of the transmitter box, with the x direction pointing up, the y direction pointing forward, and the z direction going into the transmitter. A data record is first collected with sensors reporting positions and orientations in these global coordinates (see Figure 2.3).

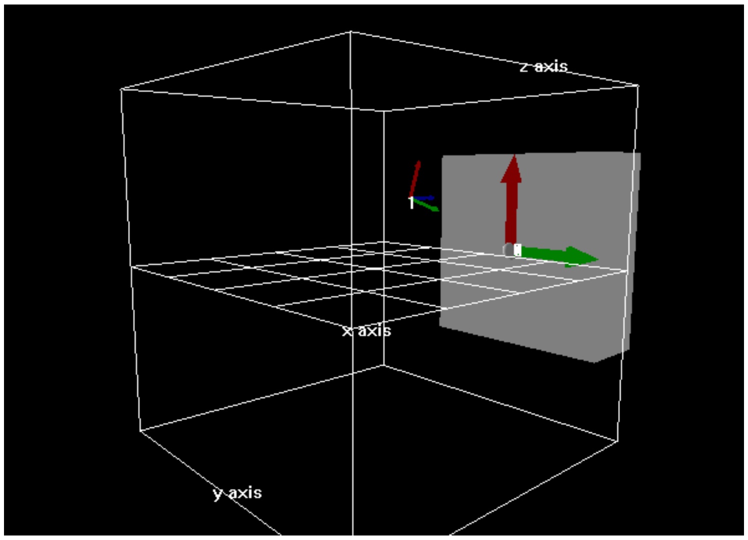


Figure 2.4- *Depiction of the global coordinate space for the NDI Wave. The red arrow represents the x-direction, and the green arrow represents the y-direction. Note the 6-DOF sensor present in the space to set the origin for the head-corrected space.*

Generally, global coordinates are not useful for speech analysis, and the data must be transformed into a head-corrected space. If a subject's head nods up and down during a data recording, their tongue is moving along with the head, but not in a way that is pertinent to their speech. Thus, by attaching a 6-DOF (degree of freedom) reference sensor to a fixed point on the subject's head, any head movement can be calculated using the following equation:

$$P_{HC}(x, y, \mathrm{z}, \mathrm{t}) = P_{ABS}(x, y, z, t) - P_{REF}(x, y, z, t) \qquad (2.3.1)$$

where P_{HC} is the head corrected position of a given sensor, P_{ABS} is the absolute position of the same sensor, and P_{REF} is the position of the 6-DOF reference sensor. A 6-DOF sensor is formed by combining two 5-DOF sensors so that rotations about the inferior-superior axis can be measured, in addition to rotation about the two other axes. This reference sensor establishes a new origin and new axes for all of the other sensors throughout the record. In the EMA-MAE data collection protocol, the reference sensor is placed on the bridge of a pair of glasses worn by a subject to maintain a consistent set of head corrected axes throughout a given recording session (Figure 2.5).

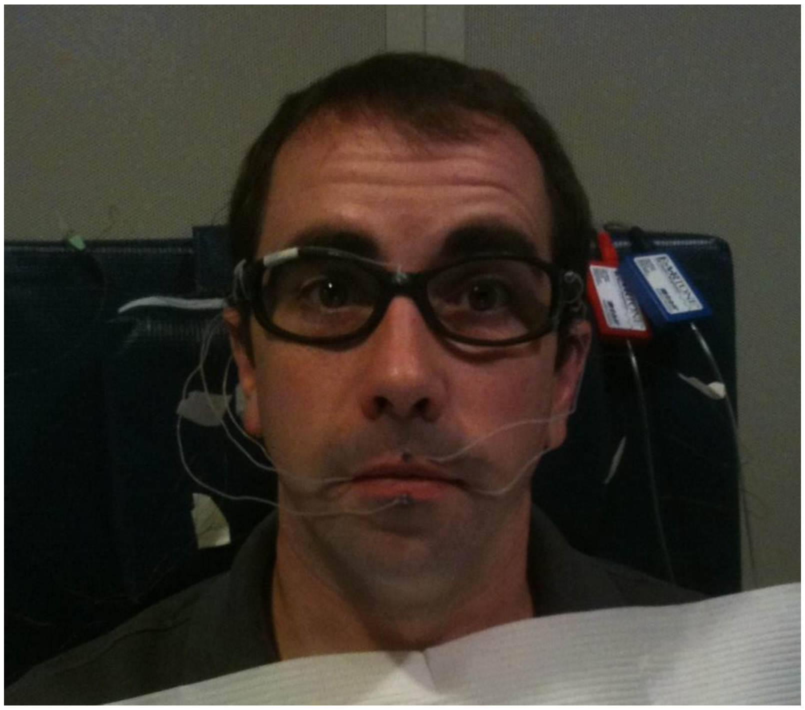


Figure 2.5- *A subject with sensors attached to the lips and tongue.*

After head correcting a given data record, the data are in a more useful form than when in the global coordinate space. However, a head corrected coordinate space with an origin between the subject's eyes, coupled with axes that do not point in a well-defined direction, still does not allow for intuitive interpretation of the collected data. The distance of a sensor from the point directly between a subject's eyes has no physiological meaning, and provides a poor landmark with respect to the articulators. Distances to the tongue surface, lips or hard palate from this point vary across subjects, meaning little comparison of speech kinematics can be done across subjects using the head corrected data. For these reasons, it is necessary to convert the data from the head corrected reference space to a spatially normalized coordinate space that controls for variations in subject anatomy. This coordinate transformation is called biteplate correction.

2.3.1 Biteplate Correction of Position Data

Biteplate correction has been widely used to aid in the analysis of speech data in both x-ray microbeam[13] and EMA [25] datasets. Biteplate correction requires a new choice of origin and axes for the data, as well as a method to convert the data from the old coordinate space to the new one. The choice of origin and axes used for the EMA data produced in the MU Speechlab emulates the maxillary reference frame described in Perkell *et al.* and the reference space used in the X-ray microbeam speech production database [25, 26] This reference frame places the origin of the biteplate-corrected space at the intersection of the maxillary occlusal plane and midsagittal plane, just anterior to central maxillary incisors. The x-axis points in the anterior direction along the midsagittal plane, with the y-axis pointing superior and the z-axis to the subject's right. Thus the goal of the biteplate correction process is to find and implement the necessary translation and rotation components that will move the origin to the central maxillary incisors and orient the space so that the X-Y plane is the midsagittal plane and the X-Z plane is the maxillary occlusal plane.Biteplate correction is so-named because it requires the placement of a plate in the subject's jaws to bite down on during a recording. An example of the type of biteplate used in the EMA-MAE database is shown in Figure 2.6.

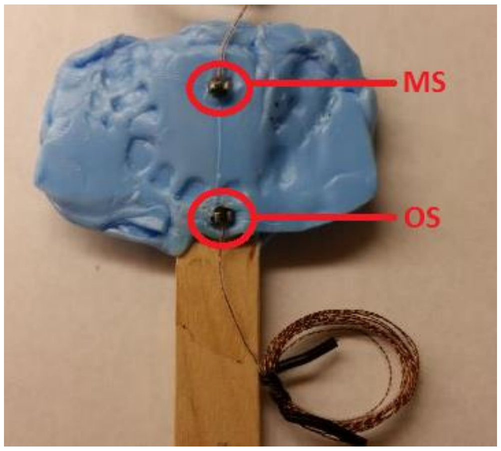


Figure 2.6- *Biteplate for a sample subject. The OS sensor defines the new origin of the space, and the line from MS toward OS defines the positive x-axis.*

In this version of biteplate correction, two sensors are placed along the midsagittal line on the biteplate, OS and MS (with the REF sensor placed between the subject's eyes, as is typical). When the biteplate is put into the subject's mouth, the OS sensor rests just anterior to the central maxillary incisors and defines the new origin for the biteplate corrected coordinate space. The OS and MS sensors together are used to define the x-axis of the new coordinate space, with the perpendicular component of the line from the REF sensor to the OS sensor defining the y-axis. The z-axis is found from the cross product of these two vectors.

Transforming from the head-corrected coordinate space to the biteplate-corrected space can be described in terms of an axes switch, a translation and a rotation (Figure 2.7). In the head-corrected space, the superior direction is the x-axis, and the anterior direction is the y-axis. These two must be swapped to obtain the desired axes for the biteplate-corrected space, which also requires that the z-axis be negated (to maintain a right-handed coordinate system). The translation is then simple, as each sensor can be corrected with respect to the origin by subtracting off the value of the OS sensor during the biteplate record from all other sensor values in a given record, much like head-correction by Equation(2.3.1). Table 2.1 gives a description of each of the four coordinate spaces described.

SPACE NAME	ACHIEVED BY…	POSITIVE X-DIRECTION	POSITIVE Y-DIRECTION	POSITIVE Z-DIRECTION
GLOBAL COORDINATE SPACE	Default NDI Wave data recording	Up	Forward	Into the face of the transmitter
HEAD CORRECTED SPACE	Equation (2.6.1) applied to Global Coordinate Space	REF's x-axis direction, roughly anterior	REF's y-axis direction, roughly superior	REF's z-axis direction, roughly subject's left
AXES-FLIPPED SPACE	Swapping x and y axes, negating z-axis in Head Corrected Space	REF's y-axis direction, roughly anterior	REF's x-axis direction, roughly superior	REF's negative z-axis direction, roughly subject's right
BITEPLATE-CORRECTED SPACE	Translating and rotating Axes-Flipped Space	Intersection of maxillary occlusal plane and midsagittal plane, pointing anterior	Perpendicular to maxillary occlusal plane, pointing superior	Perpendicular to midsagittal plane, pointing to the subject's right

Table 2.1- *Description of each of the coordinate spaces used with the NDI Wave data*

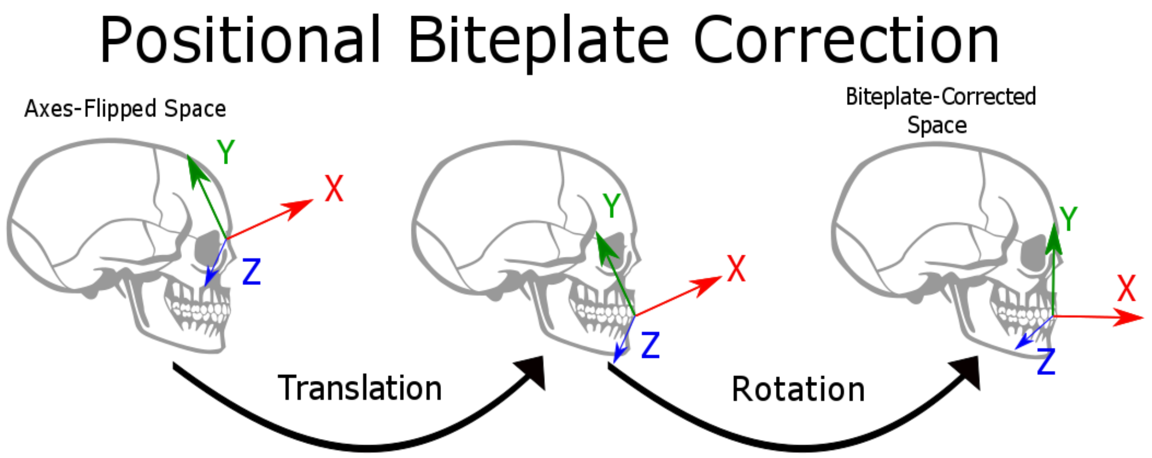


Figure 2.7- *Two steps describing positional biteplate correction. The axes are translated to the OS location, and then rotated such that the REF-OS-MS triangle defines the midsagittal (X-Y) plane.*

The rotation to transform the orientation of the axes-flipped space to the orientation of the biteplate-corrected space following translation can be described using a quaternion. Determining the quaternion required to rotate the head-corrected coordinate space into the biteplate corrected space requires selecting an axis of rotation and an angle to rotate about the selected axis. As previously noted, the rotation between two vectors can be described by an infinite number of quaternion rotations. This makes finding the required quaternion somewhat complicated.

Because several sensor positions are involved however, these can be used to constrain the set of possible rotations. Recall from Equation (2.2.11) that any quaternion can be thought of as an angle-axis pair. Using the non-rotated position of the REF and MS sensors and their rotated counterparts, all possible rotation axes about which each sensor can be possibly rotated lie on two separate planes. Finding these planes and using the cross-product of their normal vectors defines a single rotation axis suitable for rotating both sensors from their non-rotated position, to the final rotated position. Once the axis is defined, geometry can be used to find the angle between the non-rotate and final positions. Plugging these values into Equation

(2.2.11) yields a quaternion which will rotate the old sensor position values into the biteplate-corrected coordinate system.

In theory the derived translation and rotation derived to implement bite-plate correction apply to both the sensor position data and the sensor orientation data. For the position data, the implementation of this is straightforward, in that the translation is applied first as a direct subtraction, and then the rotation around the origin is applied to the translated point to compute the final position value of the sensor in the bite-plate corrected space. However, due to the way in which the orientation data are encoded, as a quaternion rotation with respect to a baseline spatial orientation, correction of the orientation component of the sensor data is slightly more complex, as described in more detail in the next section.

2.3.2 New Method for Biteplate Correction of Quaternion Orientation Data

While the process for positional biteplate correction has been implemented in many studies in the literature, there has been no process described for biteplate correction of the orientation data. Because the quaternions are referenced to the positive z-axis in the head-corrected coordinate space, if they remain unchanged when the position data is moved to the biteplate corrected space, the orientation represented by the quaternions will be ambiguous. By "correcting" the quaternions, they can be re-referenced to the biteplate-corrected positive z-axis and used to accurately represent sensor orientations in the biteplate corrected coordinate

The goal is to make the quaternion data represent the same relative sensor orientation in both the head-corrected and biteplate-corrected coordinate spaces. Because the quaternion data represents the rotation required to obtain a sensor's norm vector from a baseline norm vector, the process of biteplate-correcting the quaternions amounts to re-referencing them to the biteplate-corrected z-axis from the head-corrected z-axis. Choosing the biteplate-corrected z-axis as the new baseline norm vector makes the quaternions represent information in a similar manner as in the head-corrected space.

As with the position data, correcting the quaternions begins with swapping the x and y components and negating the z component. A quaternion $q = [q0, qx, qy, qz]$ becomes:

$$q = [q0, qy, qx, -qz] \tag{2.4.1}$$

Because the quaternion can be thought of as an angle/axis pair, when switching the head-corrected axes, the quaternion's axis in the angle/axis pair must be represented differently. In this case, when the axes are flipped, flipping the axis representation in the quaternion results in representing the same rotation before and after the change of axes. The angle thus remains the same.

Following the axes switch, the quaternions need to be re-referenced to the biteplate corrected z-axis. This re-referencing is a three step process:

1. Calculate the current sensor norm vector orientation represented in the axes-flipped coordinate space.

2. Represent the biteplate-corrected z-axis in the axes-flipped coordinate space.

3. Get the new quaternion value as the rotation required to achieve the current sensor norm vector orientation from the biteplate-corrected z-axis.

In step one, the current sensor norm vector can be calculated easily using Equation 2.4.18:

$$v_{current} = q(-v_z)q^* \tag{2.4.2}$$

where the head-corrected quaternion value ($v_{rotated}$) is used to rotate the head-corrected z-axis, $-v_z$ (the vector $[0,0,-1]$). The z-value of the head-corrected z-axis is negative because of the axes switch from Equation(2.4.1). This means the quaternion baseline orientation is $-v_z$.

In step two, the biteplate-corrected z-axis is more difficult to represent in the axes-flipped space. Beginning with the axes-flipped z-axis (following the axes switch, $[0,0,1]$), the biteplate-corrected z-axis can be found through a frame rotation. The amount of rotation is the same amount used to achieve the biteplate-corrected axes from the head-corrected axes (which is determined during positional biteplate correction). A frame rotation is used instead of a point rotation because the axis is being rotated (the frame of the coordinate space), rather than a point (like a sensor position value). Mathematically, this involves the use of Equation(2.2.20):

$$\boldsymbol{z}_{BPC} = q^*(\boldsymbol{z}_{HC})q \tag{2.4.3}$$

where $\boldsymbol{z}_{BPC}$ is the biteplate-corrected z-axis, $\boldsymbol{z}_{HC}$ is the head-corrected z-axis, and q is the quaternion used to rotate from the axes-flipped space to the biteplate-corrected space.

With both the current sensor norm vector and biteplate-corrected z-axis calculated, all that remains is to determine the rotation required to produce the former from the latter, as in step three. Getting the quaternion q between two vectors $\boldsymbol{v}_{init}$ and $\boldsymbol{v}_{final}$ requires the following equation [20]:

$$\begin{aligned}
&q = [\mathrm{w}, \boldsymbol{v}] \\
&w = \sqrt{\mathrm{norm}(\boldsymbol{v}_{init})^2\ \mathrm{norm}(\boldsymbol{v}_{final})^2} + \boldsymbol{v}_{init} \bullet \boldsymbol{v}_{final} \\
&\boldsymbol{v} = \mathrm{cross}(\boldsymbol{v}_{init}, \boldsymbol{v}_{final})
\end{aligned} \tag{2.4.4}$$

While there is no unique rotation between the two vectors, this equation gives a simple way of calculating one such rotation. Note that Equation (2.4.4) fails when the two vectors are exactly 180 degrees apart (yielding $q = [0,0,0,0]$), so a special exception must be made. See Appendix A for the code which handles the exception. In practice, this is rarely a problem, since vectors are rarely exactly 180 degrees apart.

Substituting $\boldsymbol{z}_{BPC}$ and $v_{current}$ for $\boldsymbol{v}_{init}$ and $\boldsymbol{v}_{final}$ in Equation (2.4.4) gives the new quaternion which will accurately represent the sensor orientation in the biteplate-corrected space.

2.3.3 Methods for Incorporating Orientation Data in Interpolation

Sensor orientation data represents valuable additional information about articulator position that is often under-utilized. In addition to providing directly relevant articulatory measures, such as jaw angle or tongue blade surface orientation, the orientation data can be combined with position data to more precisely interpolate additional articulatory position points, increasing the limited spatial resolution of the EMA modality. This will be illustrated using a tongue interpolation tool presented in Chapter 3. Some introductory background on interpolation methods that integrate both location and orientation information are presented here. For simplicity, the methods presented here are 2D methods of interpolation, but these can be generalized easily to three dimensions.

Hermite interpolation makes use of Hermite basis polynomials to interpolate between two points. Hermite interpolation makes use of both position and first derivative information, similar to that provided by EMA sensors. It has been used to interpolate missing data points in EMA data previously [27], and is a natural choice for the type of data provided in EMA. The four Hermite basis polynomials are:

$$\begin{aligned} h_1(s) &= 2s^3 - 3s^2 + 1 \\ h_2(s) &= -2s^3 + 3s^2 \\ h_3(s) &= s^3 - 2s^2 + s \\ h_4(s) &= s^3 - s^2 \end{aligned} \tag{2.5.1}$$

The first two polynomials are applied to the points that require interpolation, and the second two polynomials are applied to the normal vectors. In matrix form, interpolating a point R between two points p_1 and p_2 with tangent vectors t_1 and t_2 is:

$$R = \begin{bmatrix} s^3 \\ s^2 \\ s \\ 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ t_1 \\ t_2 \end{bmatrix} \tag{2.5.2}$$

Thus, provided C_1 data, Hermite interpolation produces C_0 continuous curves. It is somewhat computationally expensive because it requires N interpolation calculations for N points.

Bezier curves are another form of interpolation, often used in graphic design tools. They work very similarly to Hermite interpolation, with the exception that the derivative constraints are implicitly defined, rather than explicitly defined. Control points are used to define the curvature of the interpolation between the two points, as shown in Figure 2.8.

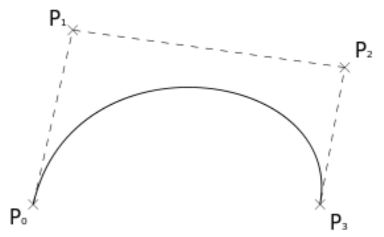


Figure 2.8- *Depiction of implicit control points used in Bezier interpolation.*

The basis polynomials for Bezier curves on the range [0, 1] are:

$$\begin{aligned} b_0(s) &= (1-s)^3 \\ b_1(s) &= 3(1-s)^2 \\ b_2(s) &= 3(1-s)s^2 \\ b_3(s) &= s^3 \end{aligned} \tag{2.5.3}$$

The quaternion data can be used in a manner similar to Hermite interpolation to define the control points and interpolate between two points. Bezier surfaces are used in three dimensions to complete the Clough-Tocher algorithm for surface interpolation, an element of which is used in Chapter 3.

2.4 Summary

When both the position data and quaternion data have been biteplate corrected, the data are in a physiologically meaningful coordinate space and can be more easily interpreted across subjects. The process begins with data collection in the global coordinate space, where data are referenced to the face of the electromagnetic field generator. The data are then converted into the head-corrected space through the use of a reference sensor placed at a fixed point on a subject, on the bridge of the nose in the EMA-MAE protocol. Following the head correction process, the data must be transferred to the biteplate-corrected coordinate space through use of a biteplate record and the biteplate correction method.

While the biteplate correction method has been previously implemented for changing the position data on a per-subject basis, no method previously existed for correcting the quaternion orientation data. To facilitate an understanding of the quaternion data and quaternion manipulations, a brief introduction to quaternions was given. Then, a process was presented for changing the quaternion data such that the orientations given in the head corrected space could be re-expressed in the biteplate-corrected space. Software implementations of each of these portions of the biteplate correction process have been created and are given in Appendix A.

Given the complexity of EMA data pre-processing, the software tools for biteplate correction stand as an important contribution toward expediting future data collection and analysis. Providing these tools and methods along with the EMA-MAE database give an easily usable, standardized method for processing the data, making EMA data generally more accessible for a broader group of speech researchers.

3. USE OF QUATERNION ORIENTATION DATA FOR ARTICULATORY REPRESENTATIONS

When both the position and orientation data are properly corrected and set in the subject's biteplate-corrected reference space, the two components can be used in concert to provide a more complete picture of articulator movement than position data alone. One example of this is in jaw movement, where a single, well-placed sensor's orientation data can provide the same information as two sensors' position data. Similarly, though a tongue sensor might remain relatively stationary for a portion of speech, the added orientation data can provide information about how tongue concavity changes over time [19]. With these ideas in mind, a pair of tools –one for jaw movement analysis, one for tongue movement analysis—is presented to demonstrate how added orientation data might be utilized in articulatory representations.

3.1 Single Sensor Jaw Tracking

3.1.1 Experimental Setup and Data Collection

To demonstrate and evaluate the use of quaternions in jaw tracking, existing EMA sensor data were used in an experimental analysis, where position-derived and orientation-derived jaw angles were compared for their correlation and noise levels. The EMA data were collected using the NDI Wave Speech Research System. As described in Chapter 2, a single 6-DOF (Degree of Freedom) sensor was attached to glasses worn by the subject for head correction purposes. In addition, two 5-DOF sensors were attached to the subject: one to the buccal surface of the right second molars (MM) and one to the labial surface of the junction of the lower incisors (MI). Sensors were attached at the gum line (using an oral surgery adhesive), since adhesion was poor at the dental surface alone. The sensor placement is shown in Figure

3.1. A biteplate record (see Chapter 2) was also taken so the data could be transformed into the articulatory workspace during post-processing.

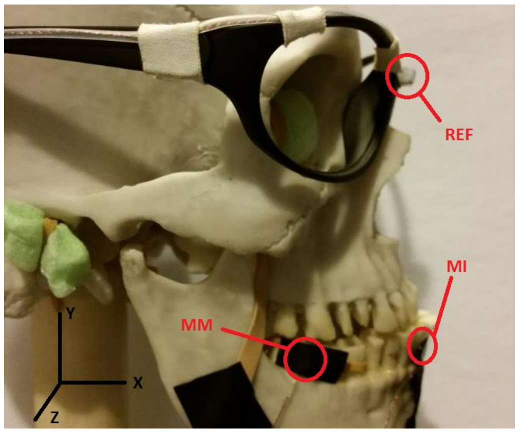


Figure 3.1- *Subject sensor placements (REF, MI and MM) during the jaw sensor tracking experiment.*

Sixteen normal-functioning, American English-speaking, female subjects repeated two speaking tasks, "buttercup" and "oo-ee-ow-ee" (in ARPABET, [UWIYAOIY]) as quickly as possible, for as long as possible. All data were collected at a sample frequency of 100 Hz. All procedures and documentation were approved by the Institutional Review Board of Marquette University

A separate recording was also taken to assess the amount of noise introduced into the jaw angle measurement. This recording was taken from a skull model with the REF, MI and MM sensors attached. The skull's jaw was kept still during the recording, so no change in jaw angle would occur. Because there is no actual movement, any variance in the angle measurements would be due exclusively to measurement error and noise, thus allowing for direct comparison of the orientation and position based jaw angle computation.

3.1.2 Jaw Angle Measurement Method # 1: Two-Sensor Position Method

Using the 5-DOF kinematic data acquired from the subject, the jaw angle can be calculated in two ways, one from positional data and one from orientation data. The first is using a simple angle calculation between the X-axis (the [1 0] vector) and the vector pointing from the MM sensor to the MI sensor, v_{MI-MM}. Because the angle of interest is defined in the XY-plane, all vectors were projected onto the XY-plane. The X-axis vector represents the upper jaw, and remains fixed at all times during the experiment. The vector v_{MI-MM} represents the movement of the lower jaw, and will vary as the subject's mandible moves up and down (Figure 3.2).

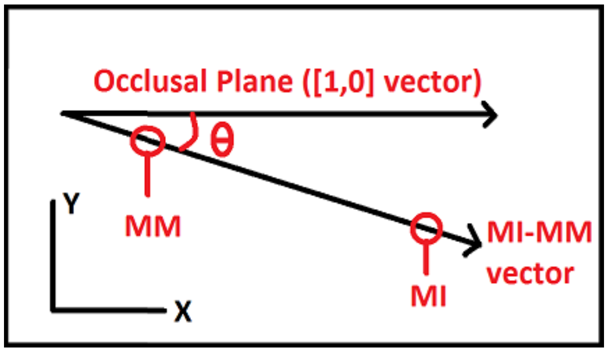


Figure 3.2- *Positional method of jaw-angle calculation.*

Using the two vectors, and defining opening of the jaws as a negative angle, the angle can be calculated as follows:

$$\theta = -\cos^{-1}\left(\frac{x \bullet v_{MI-MM}}{\|x\|\|v_{MI-MM}\|}\right) \tag{3.1.1}$$

where x is the X-axis vector, and θ is the jaw-opening angle. Because the MM and MI sensors were difficult to place exactly parallel to the X-Z plane, the "zero" jaw angle would actually be registered as a small offset from zero. To correct for the offset and report an absolute jaw angle, the position-derived angles were corrected using:

$$\theta_p' = \theta_p - \max(\theta_p) \tag{3.1.2}$$

where θ_p' is the absolute jaw angle. This method assumes that at some point during the record the subject closed their jaws completely. If the subject did not close their jaws during the record, the final jaw angle would be an underestimate of the true jaw angle.

3.1.3 Jaw Angle Measurement Method 2: Single-Sensor Quaternion Method

The second method of calculating the jaw angle for a subject requires only one sensor, MI. Using the MI quaternion data, the orientation of the MI sensor can be used to track the changes in the jaw angle over time. The same upper jaw vector (the [1 0] vector), is used as in the first method, but the lower jaw vector is calculated differently. It is obtained using (2.2.18) along with the quaternion data from the MI sensor:

$$v_L = q_{MI} v_b q_{MI}^* \tag{3.2.1}$$

where v_L is the lower jaw vector, q_{MI} is the quaternion obtained from the MI sensor, and v_b is baseline orientation vector. During biteplate correction, the baseline orientation vector is enforced to be along the z-axis ([0 0 1], or pointing toward the subject's right side). Using (2.3.1), the lower jaw vector is calculated, and the final jaw angle is solved for using:

$$\theta = -\cos^{-1}\left(\frac{x \bullet v_L}{\| x \| \| v_L \|}\right) \tag{3.2.2}$$

Ideally, if the sensor is affixed such that its orientation norm vector is perpendicular to the lower incisors, the lower jaw vector produced by quaternion multiplication will exactly match the vector of the lower jaw. However, in practice, affixing a sensor this precisely is not feasible, and therefore subtracting a baseline offset (similar to the two-sensor method) from the quaternion data becomes necessary.

As done with the position-derived angles, the orientation-derived angle offset was chosen to be the greatest recorded angle in a given record. The new angles were:

$$\theta_o' = \theta_o - \max(\theta_o) \tag{3.2.3}$$

where θ_o' represents the absolute jaw angle.

3.1.4 Results: Angle Comparison

Figure 3.3 depicts a speech record for a single female subject repeating “oo-ee-ow-ee” for 10 seconds. The position- and orientation-derived measurements produce similar values over the course of the speech sample, with only a slight offset between them.

Correlation between the orientation-derived and position-derived jaw angle calculation for the record in Figure 3 is 0.995. Over all 16 subjects, the mean correlation value was 0.981, with a standard deviation of 0.019. The high correlation values indicate strong agreement between the two methods, and validate that reasonably similar information can be obtained from one or two sensors.

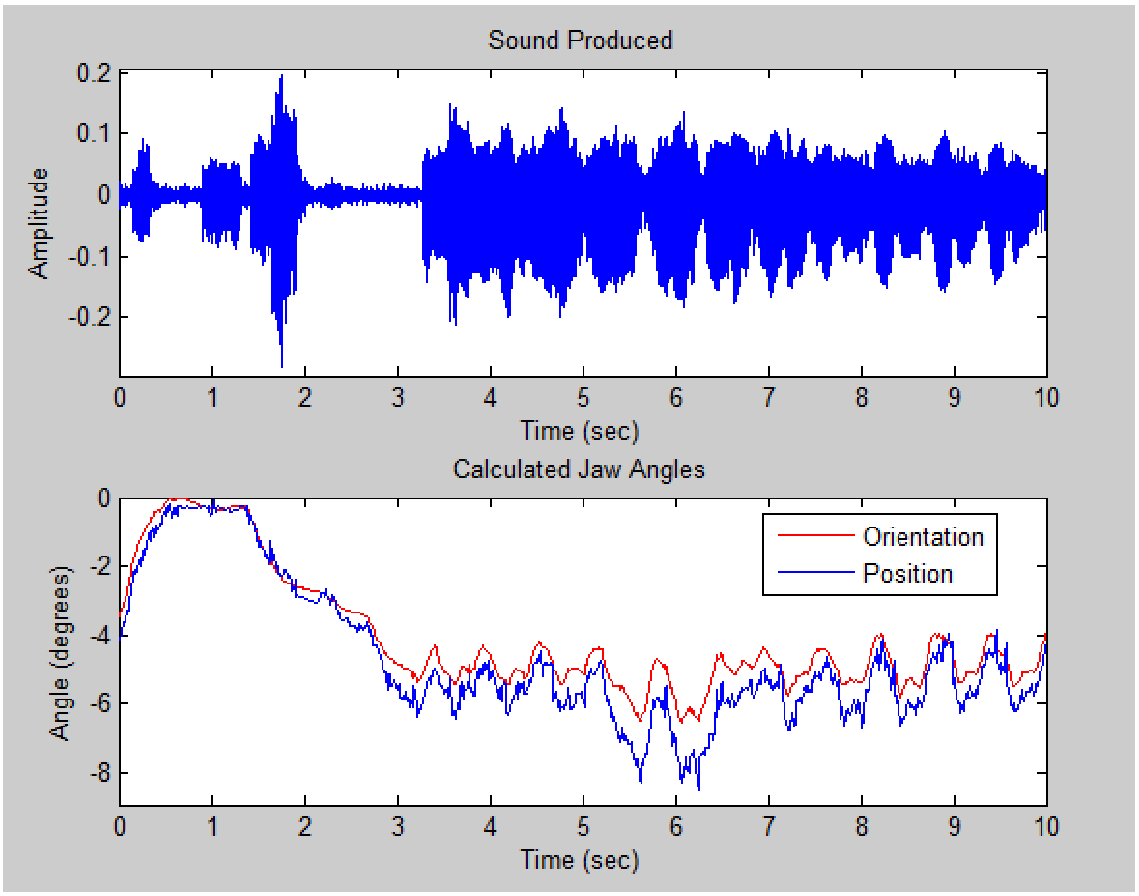


Figure 3.3-*Plot of subject saying "oo-ee-ow-ee" repeatedly from 4-10 seconds. The sound at 0-2 seconds is the test administrator prompting the subject. Note the similar shape of the position- and orientation-derived angle measurements, with a slight offset between the two.*

There are, however, several noticeable differences between the orientation and position-derived angle values. There are several sources for these differences. There may be a slight constant offset since the max jaw angles are necessarily based on two different computation methods; however, the similarity during non-speaking periods (e.g. during the prompt period in the first few seconds), suggests this is minimal. Additional sources of difference include possible sensor movement in any of the three sensors. Although all three sensors are rigidly attached, poor adhesion can sometimes cause slight movements. Specifically, poor adhesion of the MM sensor to the molar (an occasional problem during data collection) can cause it to rock and change position with the cheek tissue sliding across it, which can artificially inflate jaw angles, as the MM position vector would be at a greater y-position. The MI sensor tended to be more well-adhered to the teeth (as the front of the mouth is much easier to access), and thus is less likely to suffer from the same problem. Note that sensor movement has differential impact on position-derived and orientation-derived computation methods, with potential for much higher impact on the orientation-derived angle.

3.1.5 Results: Variance Comparison

Figure 3.4 shows the calculated angles for a rigid jaw model, along with measurement variance. The position-derived angles show a variance that is an order of magnitude larger than the orientation-derived angles. The higher variance for position-derived calculation was also indicated by the speech record shown previously in Figure 3.3.

From a signal processing perspective, the position-derived angle is based on the difference of two vectors to be calculated, acting as a differentiator and amplifying high frequency noise. The quaternion-derived angle does not require such a difference to be calculated, although as discussed previously it may be more sensitive to sensor movements during the record. It should also be noted that since EMA data can be collected at 400 Hz, which is higher than expected velocities for jaw motion, it would be possible to reduce variance of the angle calculation through simple smoothing or other tracking methods, which have not been implemented for this evaluation.

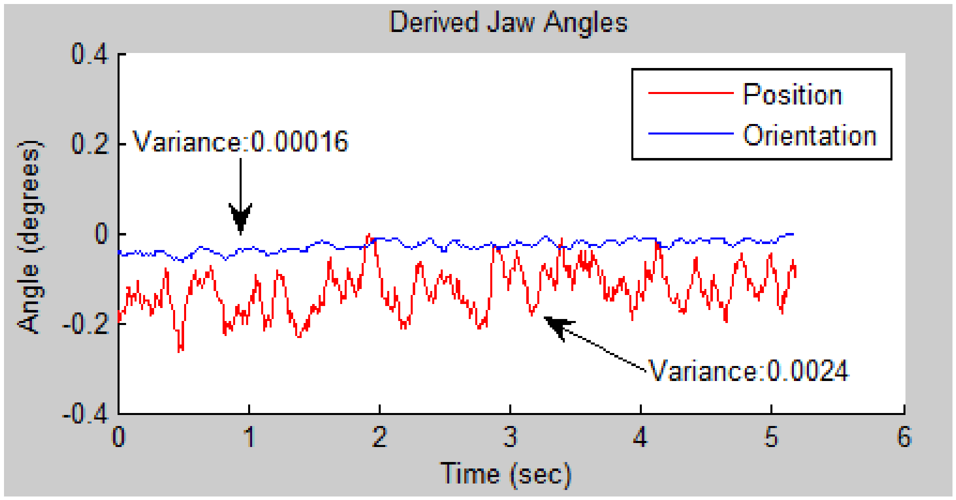


Figure 3.4- *Jaw angles calculated from a record where the sensors were kept completely still (relative to one another and the REF sensor).*

3.1.6 Conclusions and Future Directions

This experiment demonstrated the use of EMA quaternion orientation data for determining articulatory angles, and compared orientation-derived jaw angle measures with position-derived jaw measures. Applications of this technique include tongue modeling and mesh construction, as well as clinical applications for assessing jaw and speech motor kinematics.

Given the accuracy and ease of attachment of a single sensor, there are numerous applications for which single-sensor jaw-angle tracking would be useful. One use would be in articulatory component analysis, where contributions of the jaw need to be separated from simultaneous movements of the lower lip and tongue [28]. The method could also allow for EMA-based studies on the kinematics of chewing, as the single sensor does not interfere with normal chewing or swallowing.

The method as described also has room for future improvements. The jaw model used here is a simple temporomandibular hinge model assuming a rotational axis perpendicular to the midsagittal plane with no joint skewness. A more complex 3-D jaw model reflecting the true dimensionality of jaw movement would also allow measurements of joint looseness or jaw movement skewness [29] with correct sensor selection and placement.

Overall, results indicate that even small measurement variance is magnified by using a position-derived angle calculation. Provided that the sensor can be reliably attached, an orientation-based method showed more than a factor of ten reduction in computed angle variance.

3.2 Tongue Analysis Tool

Aside from opening and closing of the jaws, tongue positioning and shape have the largest impact on acoustic filtering by the vocal tract [30]. With this in mind, using EMA sensors to accurately estimate the behavior of the surface of the tongue during speech could have broad applications in analysis and synthesis of speech.

Applying similar principles to those used to calculate the jaw angles, a mesh estimating tongue shape and position can be created. While the tongue sensor positions give some information about the tongue shape, on their own they do not provide much insight about what is going on at other positions nearby. Coupling the position data with quaternion orientation data provides information about both the tongue at the sensor position, as well as nearby that position. Given this added information, interpolation of tongue fleshpoints can be made more accurate.

3.2.1 Tongue Interpolation Background

Because of the added information given by sensor orientation data, these data have proven useful in applications such as head stabilization [31], and jaw angle measurement [28]. Orientation data have also been suggested for reconstructing tongue shape in the coronal plane [32], and have aided in the control of a skeletal animation of a tongue surface that was extracted from MRI data [33]. Yet no general method currently exists for interpolating tongue fleshpoint positions between EMA sensors using combined position and orientation data.

Other modalities for representation of tongue shape and movement include ultrasound [34, 35] at rates as high as 90Hz, and cine-MRI and real-time MRI [36, 37] at rates sample rates as high as 33 Hz [38]. EMA offers superior temporal resolution (up to 400 Hz), but somewhat lower spatial resolution because of the limited number of EMA sensors attached to the tongue surface. While EMA can be combined with MRI data to more completely characterize the moving tongue [39], such multi-modality approaches are cost-prohibitive and less practical for clinical assessment of tongue movement. Since increasing the number of EMA sensors is almost certain to influence movement patterns, reducing the validity of data for kinematic assessment, a method for interpolating tongue fleshpoint positions from relatively scant EMA sensor data would be very useful for both research and clinical applications. The goal of this work is to develop and evaluate such an interpolation method based on data from a three sensor EMA configuration.

3.2.2 Tongue Interpolation Methods

The three points on the tongue used for interpolation in the current experiment included: a sensor placed on the tongue blade, about 1 cm back from the tongue tip along the midsagittal line (TB); a sensor placed on the tongue dorsum, as far back on the tongue as possible along the midsagittal line (TD); and a sensor placed on the left side of the tongue, one centimeter from the tongue's edge at the midpoint between TB and TD (TL). Each of these was a 5 Degree of Freedom sensor that included 3D position information and 2D orientation information that represented the plane of the sensor. With sensors oriented roughly parallel to the tongue, the data provided thus includes both position and gradient information at the vertices of a triangle on one half of the tongue.

Given data that has both position and derivative information, several different existing interpolation schemes are possible. The simplest is inverse-distance weighting, which expresses each interpolated point as a combination of the other points, weighted by how close the interpolated point is to each of the known points [40]. This is computationally simple and intuitive, but does not make use of the gradient information provided by the sensors. Other methods for surface fitting utilizing derivative information include Bezier patches, Hermite surface interpolation, and Clough-Tocher interpolation. A thorough summary of each of these techniques can be found in [41].

To interpolate inside the triangle formed by the sensors, a hybrid technique was chosen, drawing from techniques used in both Clough-Tocher interpolation and inverse-distance weighting. This approach makes the resulting algorithm computationally easy, while making adequate use of the derivative information at each of the known sensor points.

Prior to introducing the algorithm, there are a few important points to be made about the use of orientation data for obtaining gradient data on the tongue surface. Since sensors are initially only roughly positioned, the first step in obtaining accurate derivative information and calculating the tongue mesh is to establish a baseline orientation, without which it is not possible to infer a gradient from the sensor orientation data.

There are several techniques that could be used to define the tongue baseline. A natural way is to find some time-point where the tongue is flat, and define the orientation for each of the sensors at this point to be completely in the superior direction. This could be done with a bite-plate or tongue depressor to hold the tongue flat in the mouth with the sensors attached. Another means would be to use the recorded sensor position data to determine a time-point when the three sensors formed a plane whose normal vector was parallel to a vector pointing in the superior direction. It is very unlikely that the normal vector given by this plane will actually be parallel, so choosing the time-point where the angle between the vector pointing in the superior direction and the normal vector to the plane of three sensors is at a minimum is the closest approximation. A final method would include simple averaging of the orientation data over a long dataset, on the grounds that the long-term temporal average of tongue orientation will be roughly in the superior direction. Each of these methods has clear strengths and weaknesses. For the current work, the quaternion averaging method and plane definition methods (referred to as the positional method) are used to assess the effects of baseline quaternion definition on overall accuracy.

For each individual record, the average orientation of the sensors was calculated and assigned a value that represents the orientation pointing in the superior direction. Because orientation data are provided in quaternion format [42], the averaging technique employed the eigenanalysis method described in [43]. Defining this average quaternion value as the superior direction, an adjustment quaternion to be applied as a correction to all other data can be calculated as follows:

$$q_{adj} = q_{avg}^{*} q_{superior} \tag{3.4.1}$$

where q_{avg}^{*} is the conjugate of the average quaternion, and $q_{superior}$ is the quaternion whose orientation represents the superior direction. Right-multiplying the adjustment quaternion with all quaternions in a given record results in adjusted orientations that point in a direction roughly normal to the tongue surface:

$$q_{corrected} = q_{raw} q_{adj} \tag{3.4.2}$$

With quaternions that more accurately provide gradient information about the tongue surface, virtual positions along the tongue can be calculated from the position and derivatives. The method begins by breaking up the macro-triangle formed by TD, TL, and TB into sub-triangles, in a Clough-Tocher split [44]. This split breaks a single triangle into three new triangles, by connecting the three original vertices to the triangle's centroid. This process can be repeated with the sub-triangles as many times as needed to obtain the desired number of points inside the macro-triangle. For the purposes of this description, the triangle points will be described as points in the X-Z plane, with the Y-value at each XZ position representing the tongue height that needs to be found.

Using this split, once the Y-position and gradient at the centroid of the macro-triangle are found as described in the following sections, the process can be repeated to determine the position and gradient values at the centroids of each of the sub-triangles.

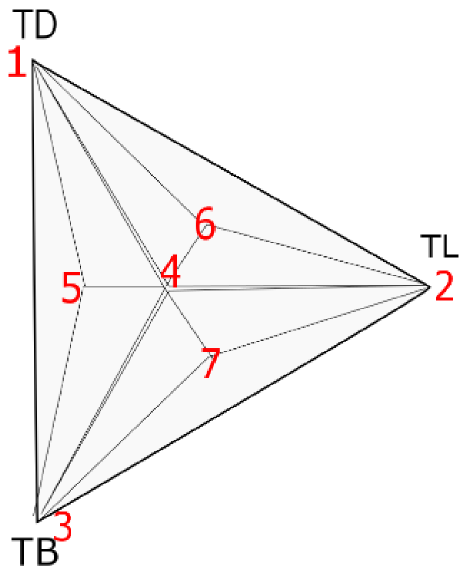


Figure 3.5- *First seven points found using a Clough-Tocher split. Point 4 is the centroid of the macro-triangle, with points 5, 6, and 7 found from the centroids of the sub-triangles.*

Given the point, (b_x, b_y, b_z) and normal vector, $(B_x^{'}, B_y^{'}, B_z^{'})$ represented by sensor TB, an equation for a plane can be written, representing a tangent surface on the tongue:

$$B_x^{'}(x-b_x)+B_y^{'}(y-b_y)+B_z^{'}(z-b_z)=0 \tag{3.4.3}$$

This plane can be used to project the tongue height *y* at some point in the X-Z plane by rearranging and solving for *y*:

$$y = P_B(x,z) = b_y + \frac{B_x^{'}(x-b_x) + B_z^{'}(z-b_z)}{-B_y^{'}} \tag{3.4.4}$$

This process can be repeated for each of the sensors, resulting in three planes, $P_B(x,z), P_L(x,z), P_D(x,z)$ which are capable of providing projections at some point in the X-Z plane (Figure 2.2).

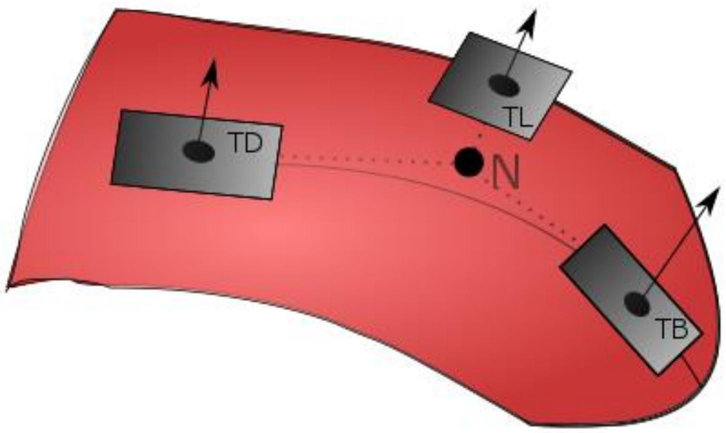


Figure 3.6- *Planes created from sensor position and orientation, used to project the value at a new point N.*

Inverse-distance-weighting was used to combine projections for the *y*-value at the centroid from each sensor into a single *y* estimate, with the weight given to a sensor's projection assigned to be the inverse of the distance between the sensor and the desired point. For example, the weight given to a projection from TB is:

$$w_B(x,z) = \frac{1}{\sqrt{(b_x - x)^2 + (b_z - z)^2}} \tag{3.4.5}$$

Others weights are determined similarly.

Combining equations (4) and (5) for each sensor yields the y-value for a new point (n_x, n_y, n_z):

$$n_y = \frac{w_B(n_x,n_z)P_B(n_x,n_z) + w_L(n_x,n_z)P_L(n_x,n_z) + w_D(n_x,n_z)P_D(n_x,n_z)}{w_B(n_x,n_z) + w_L(n_x,n_z) + w_D(n_x,n_z)} \tag{3.4.6}$$

The normal vector for the new point can be calculated analogously, as the inverse-distance-weighted average of the normal vectors at TB, TL, and TD.

With the macro-triangle's centroid position and derivative information calculated, this process can be repeated for other sub-triangle centroids on the tongue. It should be noted that to complete the interpolation, the calculated tongue heights are reflected across the midsagittal plane, which assumes that the tongue behaves approximately symmetrically (generally the case for normal speakers). To allow for visualization, the points were connected to form a triangular mesh.

To assess interpolation accuracy, six sensors were placed on the tongue, with the original three (TD, TL, TB) used to create the projected tongue values, and three additional sensors (labeled S1, S2, S3) served as ground truth against which the projected tongue values could be compared, as shown in Figure 2.3.

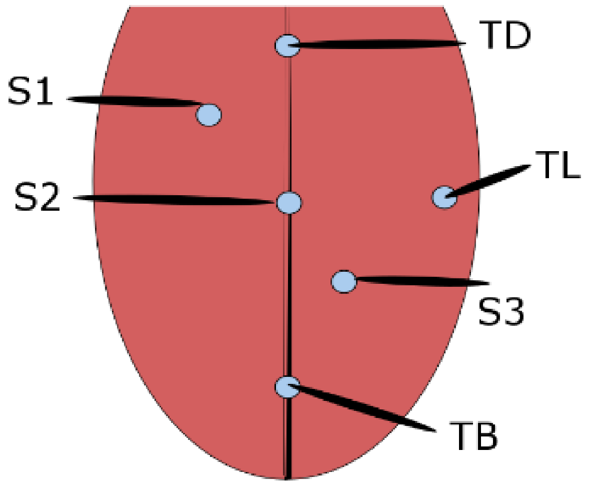


Figure 3.7-*Tongue sensor placement for evaluation*

The single subject used to conduct the experiment was a 22-year-old female, whose native language was American English. The data were collected at 400 Hz using the NDI Wave Speech Research System. All procedures and documentation were approved by the Institutional

Review Board of Marquette University. The data were corrected for head movement using a reference sensor attached to glasses worn by the subject throughout all recordings. Bite-plate correction was also used to place the data in a physiologically defined coordinate space, with the X-Z plane corresponding to the maxillary occlusal plane and the X-Y plane corresponding to the midsagittal plane following bite-plate correction. The origin of the space was set at the tip of the central maxillary incisors.

Six speech records were taken from the participant. The caterpillar script [45] was read twice, both as an acclimation record and a typical connected speech record. The remaining four records consisted of consonant-vowel-consonant (CVC) words or pseudo-words with consonants /k/, /t/, and /p/ flanking vowels /i/, /e/, /æ/,/ɚ/, /ʌ/,/u/,/o/, and /ɑ/ (or /IY/, /EY/,/AE/, /ER/, /AH/, /UW/, and /OW/ in ARPABET). Three of the records consisted of five repetitions of each word or pseudo-word, embedded inside a carrier phrase (for 40 repetitions in each record). The carrier phrase used was “It’ll say (/kik/) again.” This allowed for more natural use of the words in context, including more natural variability. The other single record was simply all 24 pseudo-words repeated without the carrier phrase. For all records, excluding the reading of the caterpillar script, the participant was provided a verbal prompt from demonstrating each target utterance.

For each of the six records, a basic error analysis was done to evaluate the predictions made by the interpolation method. This analysis included calculations of mean absolute error, standard deviation of absolute error, maximum overestimate, and maximum underestimate for each sensor. The errors were calculated by measuring the difference between the projected tongue height and actual tongue height within a sub-triangle. Additionally, these same metrics were calculated within only the vowels in each of the CVC records. Combined, these metrics give a picture of overall errors for the algorithm, as well as errors for a variety of vowels in certain consonant contexts.

3.2.3 Tongue Interpolation Results and Discussion

The results of the evaluation are shown in Tables 3.1 and 3.2. For this first set of results, the quaternion averaging method was used to calculate the quaternion baseline orientation. Generally, S3 was the most accurately predicted from the three other points, but all sensors showed mean absolute errors below 3 mm, with standard deviations near 1.5 mm. The maximum errors averaged across all records for sensors 1, 2 and 3 were 10.09 mm, 12.46 mm, and 8.81 mm respectively. The vowel segments in the CVC records did not differ significantly in accuracy from all of the records as a whole. The total accuracy of the EMA sensors is 0.5 mm (with an unknown orientation resolution) [46], which places an upper limit on the accuracy achievable through interpolation.

Error Table (Averaging)	*Mean Error (mm)*	*Dynamic Range (mm)*
S1	2.56 ± 1.72	23.82
S2	2.28 ± 1.45	28.23
S3	0.90 ± 0.93	36.29

Table 3.1-*Mean absolute error and standard deviation across recordings for the quaternion averaging quaternion correction method, compared to sensor dynamic range.*

Expanded Sensor Error Table (Averaging)

Filename	*S1 Mean Error*	*S1 Error SD*	*S2 Error*	*S2 Error SD*	*S3 Error*	*S3 Error SD*
KVK	2.56	1.67	2.83	1.42	0.98	0.94
TVT	2.96	1.66	2.64	1.55	0.92	1.01
PVP	2.86	1.81	2.13	1.43	1.00	1.00
Caterpillar	2.56	1.73	2.06	1.43	0.86	0.88
Caterpillar (acclimation)	2.53	1.81	1.95	1.29	0.76	0.75
All Words	2.24	1.54	2.13	1.67	0.79	0.80
Overall	2.65	1.72	2.28	1.45	0.90	0.93
Vowel Context Data						
Filename						
KVK	1.85	1.76	3.23	1.50	0.98	0.75
TVT	2.35	1.75	1.88	1.06	0.66	0.45
PVP	2.78	2.28	1.82	1.17	0.85	0.60
Overall	**2.33**	**1.93**	**2.31**	**1.24**	**0.83**	**0.60**

Table 3.2- *Expanded results of the error, broken down by file. The errors were relatively consistent across files, including the vowel segments alone.*

In a typical case, the interpolation method overestimates the value of S1, while underestimating the value of S2 and S3. The larger, overestimated error in S1 is reasonable, as by using only linear, first derivative components to interpolate the points, changes in tongue concavity near TD on either side of the medial sulcus will be smoothed over. S2 errors arise from only two sensors being used along the medial sulcus, which are inadequate for fully capturing midsagittal curvature. S3 was consistently the most accurate. Taken together, the accuracy of the sensors suggests that areas with less change in curvature can be more accurately interpolated.

The predictions were accurate enough that estimated gross changes in tongue position and shape (e.g. changes in concavity) matched the actual changes when viewed as a mesh. However, because the three data points contain only first derivative information, no changes in tongue concavity could be predicted within the interpolated area. A coronal cross-section of the interpolated tongue will always appear as a single concave or convex curve (u-or n-shaped), when in reality the tongue surface can contain multiple changes in concavity (allowing for an m-shape).

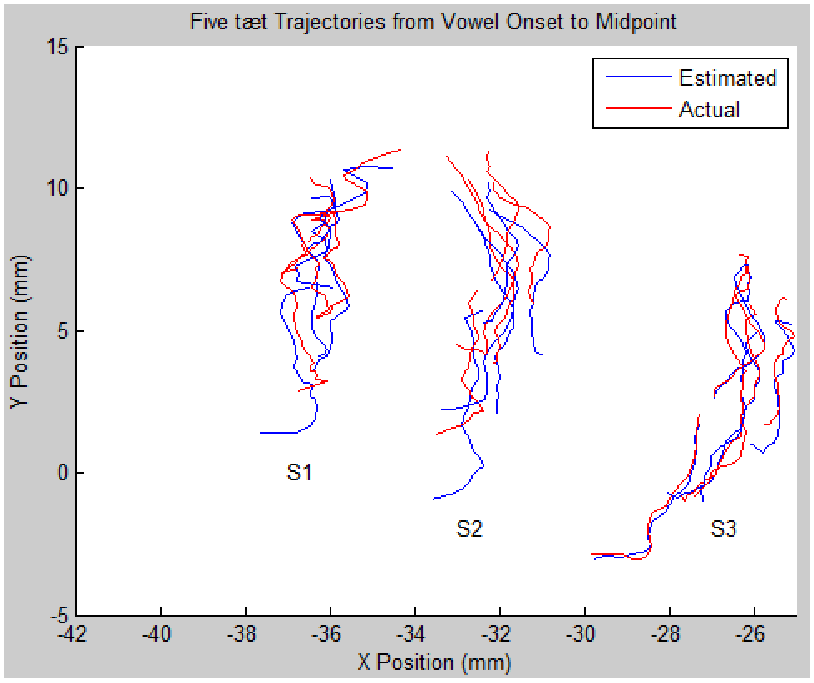


Figure 3.8- *Estimated and actual smoothed vowel trajectories for an example word (tæt).*

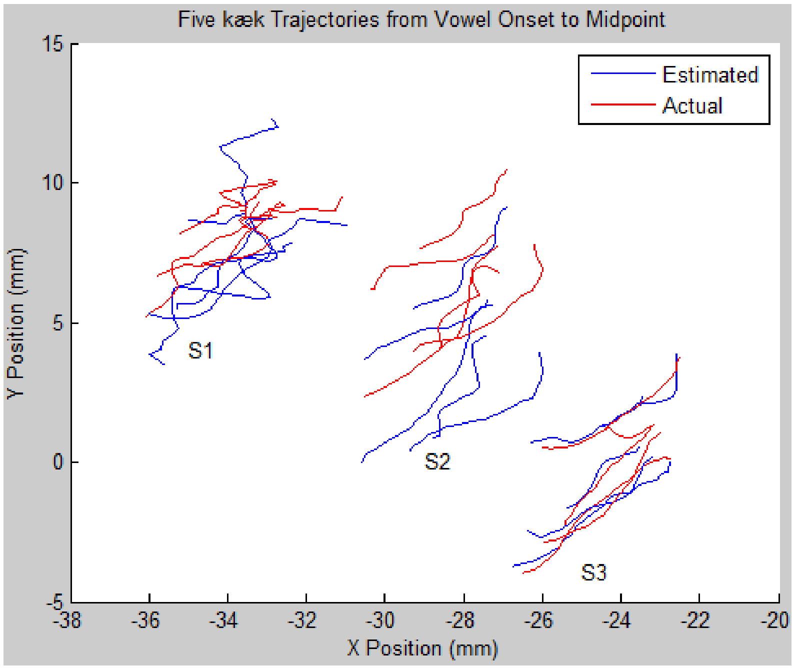


Figure 3.9- *Estimated and actual smoothed vowel trajectories for an example word (kæk).*

An example of actual versus estimated trajectories for the interpolated tongue points are shown in Figures 3.8 and 3.9, for two different consonant contexts. The trajectories have been smoothed using a moving average filter to remove jagged movements in the trajectories that are due to sensor noise and do not represent a natural movement. These trajectories illustrate the scale and nature of the interpolation accuracy, showing that the proposed interpolation method is able to characterize tongue motion throughout an articulatory movement. While the absolute range and positions show noticeable errors, the paths traced by the virtual and actual sensors appear very similar, suggesting measures like speed and distance travelled could be obtained via interpolation. Similarly, the variance in the actual and estimated trajectories is close to the natural variance in the real trajectories alone.

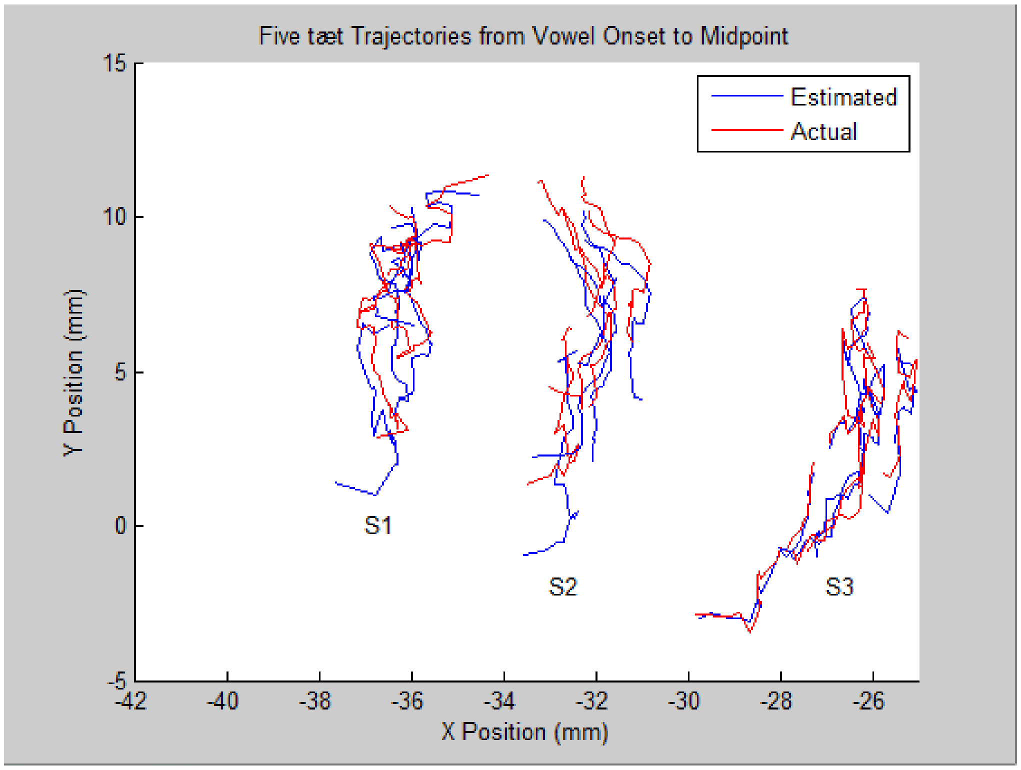


Figure 3.10- *Estimated and actual unsmoothed vowel trajectories for an example word (tæt).*

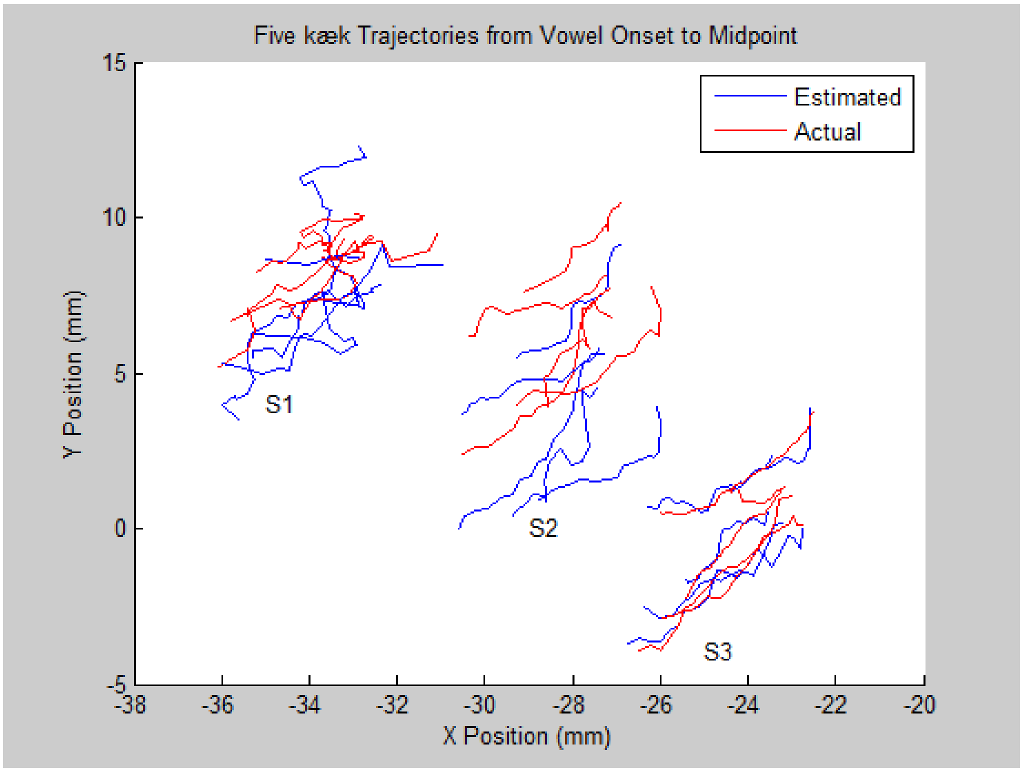


Figure 3.11- *Estimated and actual unsmoothed vowel trajectories for an example word (kæk).*

Unsmoothed trajectories are shown in Figure 3.10 and 3.11. When compared with Figures 3.8 and 3.9, they contain similar information, but appear to be too noisy to be considered truly "natural" movement. Smoothing of articulatory trajectories is common practice for this reason, although it might obscure higher-frequency content in the signal. This tactic can be justified, however, as long as the lower frequencies that are important for production of a given phoneme are captured.

Error Table (Positional)	*Mean Error (mm)*	*Dynamic Range (mm)*
S1	3.68 ± 3.55	23.82
S2	2.92 ± 5.12	28.23
S3	1.73 ± 2.48	36.29

Table 3.3- *Mean absolute error and standard deviation across recordings for the positional quaternion correction method, compared to sensor dynamic range.*

The results of using the positional quaternion correction method are shown in Table 3.3. These results are clearly inferior to the accuracy provided using the quaternion averaging method, suggesting the averaging method more adequately captures the flat-tongue position than does the positional method. From a tongue physiology perspective, the positional method falsely assumes that when the tongue dorsum, tongue blade, and lateral portion of the tongue form a flat plane, the entire tongue is flat. One can easily picture a situation where the tongue is concave up or concave down, but the three sensor points chosen form a plane that is parallel to the X-Z plane.

Future work includes a number of improvements to increase the accuracy of the interpolation method and extend it beyond its current limitations. The development of a more reliable baseline-orientation protocol would provide more accurate gradient information. Because the quaternion averaging method was shown to be superior to the positional method, this method will be used as a point of comparison for future quaternion baseline definition algorithms.

Additionally, it is possible to add more sensors to the tongue that can be used in the interpolation, which would add additional information and improve the overall accuracy. Adding a sensor outside of the macro-triangle used in the current work would allow for some extrapolation outside of the main portion of the tongue body. The current focus was on simple tongue configurations used in vowel articulation, but additional sensors could provide information about more complicated changes in concavity and curvature seen during liquids and consonant clusters. However, since adding sensors may affect sensor adhesion reliability and decrease the naturalness of articulation, increased accuracy must be weighed against these considerations.

By incorporating both orientation and position data from the EMA sensors, this approach substantially increases the overall amount of information and representative capability of EMA. In particular, the method provides a mechanism to estimate tongue position at untracked locations, both laterally and midsagittally. Combined with additional reference information such as palate height, this approach enables the creation of much more accurate articulatory features that directly correspond to vocal tract cross-section and shape. Applications of this work include clinical assessment of articulation, studies of pronunciation variation and accent, as well as speech processing technologies such as acoustic-to-articulatory inversion and articulatory audio and audio-video synthesis.

4. EMA VISUALIZATION TOOL

After EMA data have been collected and processed, the data require some validation. During recordings, a variety of problems can arise. Sensors fall off of the articulators, fail to report a position sample, become disconnected from the receiver, or break. Simple, automated data validation code can count NaN (not a number) instances and provide some assurance of data quality. However, without actually visualizing the data, there is no way to validate whether or not the data represent meaningful information that would be useful for analysis.

If the data are determined to be useful, analysis can begin utilizing the sensor data in combination with the acoustic data. The analysis will vary widely by discipline, but often includes indexing the speech into segments, tracking sensor trajectories, determining sensor speeds, and viewing the patterns and coordination between the acoustic and kinematic data. As an aid to individuals making use of the EMA-MAE database, a set of software tools will be included to view and process the dataset. The goal is to include software that provides capabilities for both data validation and analysis, while being accessible to the greatest number of users possible.

4.1 Review of Existing EMA Visualization Software

Several pieces of software have been developed to work with EMA data over the past few years. The main examples include EMATOOLS [47], MView [48], and VisArtico [49]. Each of these has their strengths and weaknesses, which will be discussed in some detail.

4.1.1 EMATOOLS

EMATOOLS was developed and distributed by Noël Nguyen from the University of Provence in France in 2000. It is a collection of MATLAB scripts that are capable of displaying and indexing various forms of articulatory data, including EMA, electropalatography (EPG), and even x-ray microbeam data. The scripts can be used together to create user interfaces. Each of the user interface components provides a different display capability, showing either the audio and selected 1D kinematic data, the EPG data, or the midsagittal trajectories of the sensors. The scripts and functions can also be pulled out of the package to be used on their own, providing the ability to load various file types, calculate sensor velocities and accelerations, and plot the variety of data types in many different forms.

However, for current EMA data visualization and processing, EMATOOLS has serious flaws. Because it was developed 15 years ago, EMATOOLS is no longer supported or maintained. Any problems with the software cannot be reported or fixed because of this, and the user left to make improvements or adjustments as needed. Also, because it is built as a part of MATLAB, the group of researchers who can easily make use of it must have technical skills, familiarity with MATLAB, and a MATLAB license (which can be expensive). Finally, the analysis tools allow for 1D or 2D analysis, with no 3D capabilities in the software. Though it has many desirable features, EMATOOLS has too many drawbacks to be used as a tool for analysis and visualization of the EMA-MAE dataset.

4.1.2 MView

MView is similar to EMATOOLS in that it is a set of MATLAB scripts developed for the purpose of EMA analysis. MView was created by Mark Tiede at Haskins Laboratories and released in 2005. It is still currently supported and available by contacting the developer for a copy of the software. MView supports viewing EMA sensor trajectories, sensor positions in a midsagittal view, and palate data in the midsagittal view. It also supports viewing the audio file, both as an oscillogram and a spectrogram. Indexing is allowed through any of these views, so the dataset can be broken up into meaningful segments and batch processing can be done.

One other distinctive feature of MView is its support for finding certain time points where a given articulatory gesture occurs. By defining minimum, maximum and threshold values for sensor values, lip opening, lip protrusion, tongue raising, and tongue lowering can be found in a dataset loaded into MView. Similar functions are built in for velocity-based detection of articulatory gestures. Once the relevant time indices are located, they can be saved to the MATLAB workspace, and eventually saved to file for future analysis. Given its broad feature set, MView is a very nice tool for EMA analysis, and has been recommended for use by Narayanan *et al.* [27] for use with a recent MRI/EMA database.

While MView provides excellent functionality for technical users in a MATLAB environment, it contains the same drawbacks as EMATOOLS in that users without a MATLAB license and significant MATLAB experience will be unable to use it. It also does not offer full 3D visualization of the sensor movements, or any analysis components for the quaternion orientation data. Overall, MView's analysis capabilities are excellent, but its utility for many non-technical speech researchers is limited.

4.1.3 VisArtico

Of the software tools mentioned, VisArtico represents the only cross-platform solution for working with EMA data. Developed in Java, VisArtico only requires that a user has a Java Runtime Environment (JRE) installed on their computer, making it usable for Windows, Mac OSX and Linux operating systems. This makes it an attractive option for researchers without access to a MATLAB distribution.

While providing excellent accessibility, VisArtico features many tools for both visualization and analysis of the data. For visualization, VisArtico contains a 3D spatial view, complete with both sensor position and orientation representation. It also has a 2D midsagittal view for traditional data viewing and a 1D trajectory view, which lets the user view the movement of a sensor in a single dimension over time. All of these can be played back synchronously with the audio, allowing for easy data validation.

VisArtico also has a variety of analysis tools built-in. On loading EMA data, each sensor can be assigned to a certain articulator, and the overall position of the lips and tongue can be interpolated given the sensor positions. The configuration interface is a key step in making the EMA data usable for a variety of researchers who may lack the technical skills needed to process the data on their own. VisArtico also supports adding a palate trace file to a set of subject data, so the sensor movements relative to the palate can be viewed. Along with the palate drawn into the midsagittal space, sensor trajectories can also be shown for an entire data file. Lastly, like other EMA software tools, VisArtico supports indexing of the kinematic and audio data, as well as labeling phonemes for the indexed data. These analysis features are comprehensive and make VisArtico a solid choice for many users for EMA data analysis.

Unfortunately, VisArtico is currently incompatible with the EMA-MAE dataset. It was originally developed for use with the Carstens EMA system, and thus works well with all Carstens generated data. According to VisArtico's documentation, the latest version supports NDI Wave-produced EMA data, but several attempts to load the data into VisArtico from the Wave have resulted in loading errors. It is thought that problems with audio/kinematic synchronization are the problem with the dataset, as there are several issues in the Wave's timestamps when they are written to the kinematic files. One problem is that the timestamps often begin with a non-sense value that is well over one million seconds. Second, the timestamps that follow do not begin exactly at zero, meaning an offset is required when beginning playback of the audio. Lastly, the timestamps are placed in almost exactly 400 Hz increments, but in some cases there are slight fluctuations in the sampling rate (less than 1 Hz) throughout the record. VisArtico likely has not been designed to account for these issues, making the EMA-MAE dataset unusable with VisArtico.

4.2 EMA Software Tool Development

With none of the existing software providing a solution that is completely suitable for distribution with the EMA-MAE database, a custom tool has been developed, to provide a piece of software that can be used for data validation and simple analyses. The goal was to create a tool that is usable across many different computing platforms, by both technical and non-technical users. This goal required creating a standalone executable that could be freely distributed, requiring no special software or licenses for use, and utilizing a graphical user interface (GUI). The final product was required to be usable by non-technical users, but provide some complex functionality for analysis functions (data exporting, segmenting, etc.).

The process of developing the tool will be covered in detail. First, a development platform needed to be selected. This included reviewing various development platforms for their capabilities, including speed of development, support for scientific computing and visualization, support for GUI development, and ease of distribution. MATLAB, Java Swing, C++/Qt, and PyQt were all considered as it relates to these criteria. Then, possible libraries for data plotting and visualization were reviewed. The best visualization library was lightweight (did not require huge libraries to be included), fast (capable of plotting in real-time with the data), and flexible (allowing for both 2D and 3D plotting, with support for representing quaternion orientation data). Using the platform and libraries chosen, the tool will be described from both development and user perspectives. After describing the features, future additions to the tool will be considered.

4.3 GUI Development Platforms

There are a wide variety of development platforms for software tools that could be distributed with the database. The true number of candidates is much smaller, as the choices were limited to those where previous development experience had been obtained.

MATLAB was the first candidate, as it has great built-in support for scientific and mathematical computing, both necessities for a visualization and analysis tool. MATLAB also features the GUIDE (Graphical User Interface Development Environment) tool, which makes development and maintenance of GUI-based tools relatively simple. Using the GUIDE tool, the actual coding is minimized, as MATLAB's system of buttons and callbacks readily connects buttons to their functionality. The built-in plotting and visualization in MATLAB is also a large plus, as native support for data display and visualization does not exist in the other languages.

MATLAB-based tools have several shortcomings though, many of which have been previously noted. Distributing MATLAB tools requires the user have a MATLAB license and technical skills necessary to run MATLAB, which limits overall effectiveness. Also, while MATLAB has built-in plotting capabilities, it is somewhat slow when compared to other languages, which limits the frame rate at which real-time data playback can be viewed. Due to the widely varying technical skills of the intended users of the final software tools, MATLAB may be suitable for certain special analysis tools that will not need to be used by non-technical users, but not for a widely-distributable visualization tool.

Java Swing was another possible choice for development of the visualization tool. Java's main feature is that it is cross-platform, running the same way on every computer that has a JRE installed. Expecting a user to have a JRE is not unreasonable, as a large number of applications utilize Java, and it is likely a typical user will already have a JRE. Swing also makes it relatively easy to develop GUI-based applications, with a lot of built-in classes supporting dialogs, labels, buttons, and other GUI elements. The online support community for Java is large and there are also a great deal of open-source libraries available to be included with Java Swing applications. The data visualization and 3D graphing capabilities in Java exist in several different open source APIs, such as JZY3D.

Even with the large development community around Java however, the support for scientific computing is not as robust as its counterparts. Apache Commons Math is a decent resource for doing complex mathematical calculations in Java, but is nowhere near as complete as MATLAB, and does not have built in support for quaternions in working with the dataset. As long as most of the built-in analysis capabilities of the tool are not mathematically complex, Java would be a fine choice for developing the visualization tool. With more complicated algorithms, however, much more development time would be required to get the desired functionality. VisArtico stands as a good example of what can be done utilizing Java's GUI-development capabilities.

C++/Qt is another excellent GUI development framework. Because it relies on C++, the program speed is faster than other platforms, as C++ code is compiled directly to machine code instead of interpreted (like MATLAB or Python), or converted to intermediate JVM bytecode (as in Java). Qt is a GUI framework (developed by TrollTech, Oslo, Norway) that allows for cross-platform development of GUI applications, where the application will assume a "native" look and feel on any type of major operating system (Linux, OSX, or Windows). Qt provides built-in functionality for creating and customizing dialogs, main windows, buttons, user input, etc. This speeds up development time significantly, as Qt does most of the back-end work, allowing development to be focused on the UI functionality itself. There are also a number of plotting libraries capable of integrating with C++/Qt, with QwtPlot3D and QCustomPlot being good examples.

While using C++ along with Qt has its advantages, they each have notable downsides. Though Qt provides a cross-platform support, C++ must be compiled separately on each of the environments that the program needs to run on. Also, of the development platforms mentioned, C++ requires the more code than MATLAB, Java, or Python to complete the same task, which increases development time and complexity. This includes complexity of learning to use new libraries that would be required for plotting functions and mathematical analysis. With the cost of more development time and added libraries, C++/Qt can produce fast programs that are suitable for cross-platform use.

The final platform considered, PyQt, has many of the same features as C++/Qt, but utilizes Python as the main language. This offers the same benefits as C++/Qt, but many other benefits (and some drawbacks) of Python versus C++. Python provides a great deal of support for scientific computing, with packages like Scipy and Numpy available, and a huge development community behind them. There are also a variety of plotting and visualization tools available as libraries for PyQt (Visvis, Matplotlib, PyQwt, PyQtGraph), which allows for some choice in picking the best visualization library. The main virtue of Python over other programming languages is its easily understandable syntax and built-in abilities to do complicated tasks in relatively small amounts of code. This decreases learning and development time significantly, allowing for rapid development and deployment of applications. Python also features tools that allow it to run across all platforms (PyInstaller, Cx_freeze, Py2exe), which make distribution of the completed product to other platforms relatively simple.

While PyQt's benefits are many, it is not without tradeoffs. Of the development platforms mentioned, Python is probably the slowest for most applications, due to its interpreted, dynamic nature. It thus requires significantly more memory than other development platforms to do comparable tasks. This also means distributed Python programs are typically large (10s of MBs), with many included libraries adding to the size of the distributed file.

Given all of these considerations, PyQt was chosen as the framework for developing the visualization tool. The main benefits are development time and support for scientific computing, as these are as good as or better than the other platforms mentioned. Though Python is relatively slower, in practice, Python is fast enough to do the visualization with the correct libraries (that might rely on C and C++ to improve speed), and its speed of development outweighs the cost of somewhat slower program execution. Large application distribution files might be a problem in a program on its own, but in distributing a program that comes alongside a database that is tens of gigabytes, this really is not much of a limitation.

4.4 Plotting and Visualization Libraries

With PyQt chosen as the platform for development, a suitable visualization library was required for plotting the data in 3D. PyQt, as mentioned, has several options developed by members of the support community, including Visvis, Matplotlib, PyQwt, PyQtGraph. The choice was made to include both Visvis and Matplotlib, as they offer benefits that complement one another, but on their own would not be suitable for the needs of the visualization tool.

Visvis was used for the real-time sensor display, as it is a fast, lightweight means of using Python to make calls to OpenGL. Direct integration of Python with OpenGL maximizes speed and minimizes the complexity of the code required, while retaining the ability to plot in 3D. Visvis also features easy integration with PyQt applications, so the sensor plot could be built into the user interface like any other integrated component. One great feature of Visvis for working with the EMA sensor data is its built in support for quaternions, meaning the sensors can be plotted as cones that convey both position and orientation information. These features, coupled with a logical class structure for handling data and figures, made Visvis the clear choice for real-time data playback in the visualization tool.

While Visvis is excellent for 3D plotting, Matplotlib offers a depth of functionality that far exceeds the other Python libraries. Matplotlib has built in functions for plotting comparable to MATLAB's 2D and 3D plotting capabilities. The one major component of Matplotlib that is useful for the visualization tool is a spectrogram. Spectrograms are a built-in plot type in Matplotlib, and are essential to effectively analyzing or indexing the database's audio data. Like Visvis, Matplotlib plots can be integrated into a PyQt UI window. Including Matplotlib adds another large library to the list of dependencies for the project, but the gain in possible functionality both now and in the future is certainly worth the tradeoff.

4.5 Basic GUI Development with PyQt

Prior to describing the design of the visualization tool, some basic terminology and concepts relating to PyQt development need to be introduced. In PyQt, each component of the GUI is called a widget. Buttons, drop-down menus, user input textboxes, and spin boxes, present in many other GUI are all examples of widgets. On top of a wide variety of built in widgets, PyQt supports creating custom widgets, which can be modified versions of built in widgets, combinations of widgets, or a completely new widget created from scratch. When creating the visualization tool, the plots for displaying the sensor data and spectrogram would each be considered a separate widget.

Applications are created by embedding widgets in a main window or in dialogs. The main window is the central part of any GUI application in PyQt. The main window contains a toolbar with shortcuts and icons for executing various actions in the program. It also has a menu bar, which provides similar functionality, but usually a more complete listing of all available actions in the program. Other main window features include a status display to show the current program state and a title bar to show the name of the application and current open files.

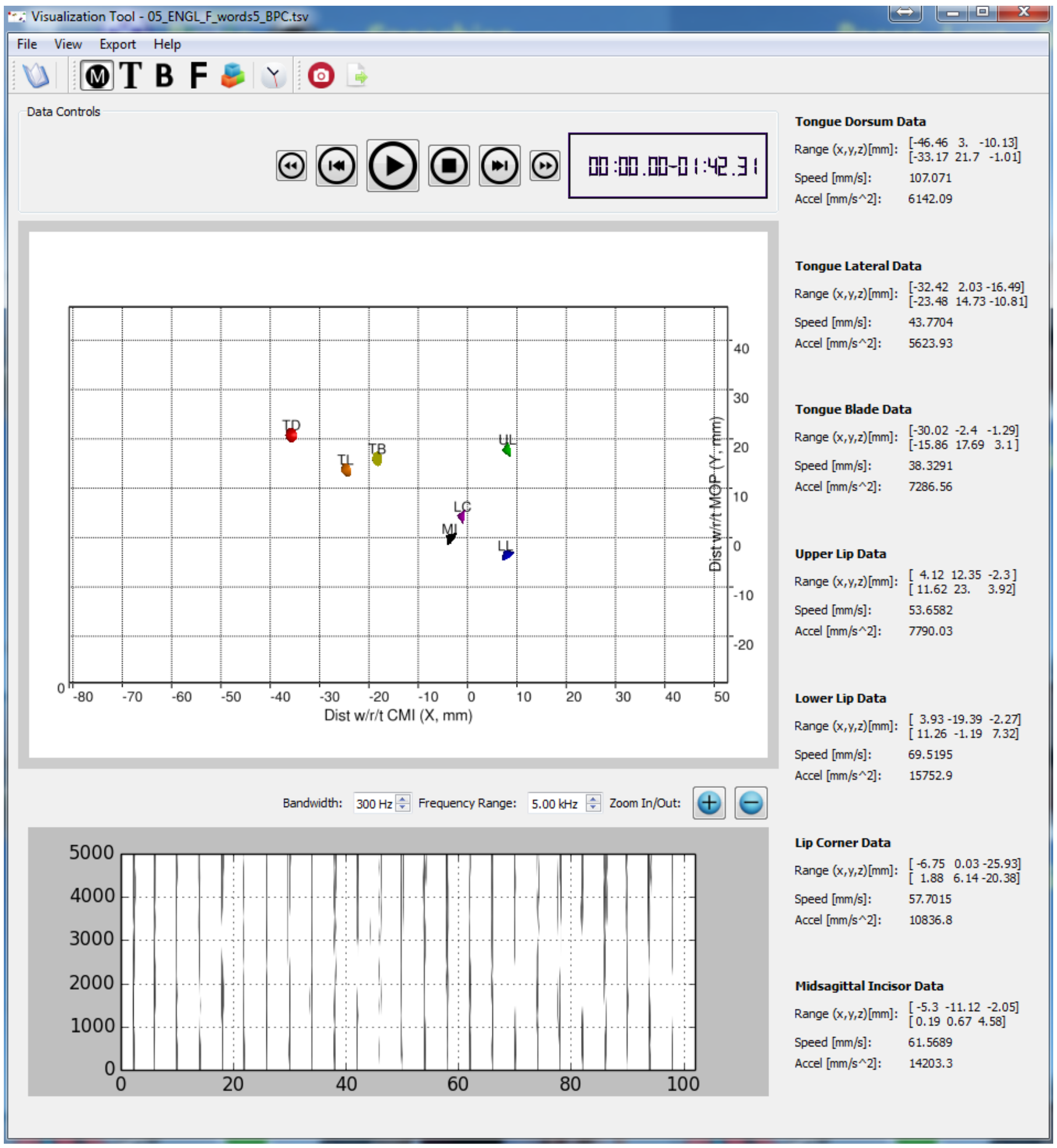
Dialogs are supplementary windows that can be used to alert the user of some information, prompt them to provide input data for the program, or allow the user to load a file, among a host of other tasks. Several built-in dialog types allow for common user actions, like warning them of an error, showing progress of a task, or selecting a file to use. On top of these built-in types, custom dialogs can be created to suit any desired user input.

Widgets interact with the underlying program logic through several mechanisms. The most basic is the use of signals and slots. For example, when a button is clicked, it will emit a “clicked” signal. This signal can be connected to a slot, which will take some action based on the button click. This slot can be a function or method inside the program, which will then be executed. Each built-in widget type will emit a pre-defined signal, per the Qt documentation.

For lower level interactions, an event handler is used instead. These are typically used with either custom widgets, or when special functionality is required. For instance, it often occurs that some action needs to be taken by the program before closing (e.g. file saving or closing). To ensure that these actions take place, the “close” event for a given dialog or window would be implemented, with the relevant file saving or closing calls made within it. Combined together, PyQt’s signals and slots mechanism, along with the event handler allow for ease of implementation, but finer control when needed.

4.6 Visualization Tool Design

The design of the tool was split into four main components. These include the sensor plot, data controller, spectrogram, and data viewer. Each is pictured as part of a main window, shown in Figure 4.1



Bandwidth: 300 Hz
Frequency Range: 5.00 kHz

Tongue Dorsum Data
Range (x,y,z)[mm]: [-46.46 3. -10.13] [-33.17 21.7 -1.01]
Speed [mm/s]: 107.071
Accel [mm/s^2]: 6142.09

Tongue Lateral Data
Range (x,y,z)[mm]: [-32.42 2.03 -16.49] [-23.48 14.73 -10.81]
Speed [mm/s]: 43.7704
Accel [mm/s^2]: 5623.93

Tongue Blade Data
Range (x,y,z)[mm]: [-30.02 -2.4 -1.29] [-15.86 17.69 3.1]
Speed [mm/s]: 38.3291
Accel [mm/s^2]: 7286.56

Upper Lip Data
Range (x,y,z)[mm]: [4.12 12.35 -2.3] [11.62 23. 3.92]
Speed [mm/s]: 53.6582
Accel [mm/s^2]: 7790.03

Lower Lip Data
Range (x,y,z)[mm]: [3.93 -19.39 -2.27] [11.26 -1.19 7.32]
Speed [mm/s]: 69.5195
Accel [mm/s^2]: 15752.9

Lip Corner Data
Range (x,y,z)[mm]: [-6.75 0.03 -25.93] [1.88 6.14 -20.38]
Speed [mm/s]: 57.7015
Accel [mm/s^2]: 10836.8

Midsagittal Incisor Data
Range (x,y,z)[mm]: [-5.3 -11.12 -2.05] [0.19 0.67 4.58]
Speed [mm/s]: 61.5689
Accel [mm/s^2]: 14203.3

Figure 4.1- *Main window for the visualization tool. The data controller, sensor plot, and spectrogram are on the left, from top to bottom. The data viewer is on the right.*

The main window also contains a toolbar and menu bar at the top to allow for manipulations of other GUI components. All components of the GUI will explored in terms of how they can be used for data visualization and analysis.

4.6.1 Feature Description- Visualization

The sensor plot is the main widget involved in kinematic data visualization. To load a file into the program, the user can either select "Open" from the file drop-down menu, or click the icon. A basic file selection dialog is then displayed, where a kinematic data file (*.tsv file) can be selected. Once chosen, the program will automatically find the corresponding audio (*.wav) file based on the file name, and load both files into the program. Following file selection, each of the seven sensors used in the EMA-MAE database collection protocol are shown in the biteplate corrected space (Figure 4.2).

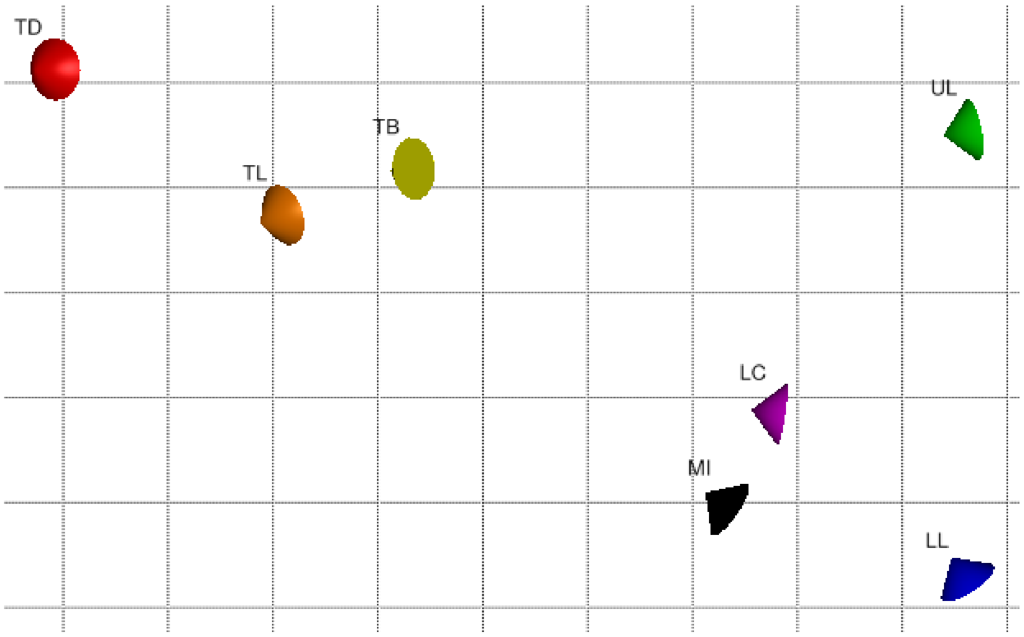


Figure 4.2 - *Sensor cones shown in a midsagittal view. The sensors are UL (upper lip), LL (lower lip), LC (lip corner), MI (medial incisors), TB (tongue blade), TL (tongue lateral), and TD (tongue dorsum).*

The sensors individually are represented by a small, colored cone and a small label placed alongside. The cone shape is ideal for representing the data, as it compactly represents both position and orientation information given by the EMA sensor coils.

The sensor plot can be shown in several different views, using either the "View" dropdown menu, or by clicking one of the four letter icons in the main window's toolbar (Figure 4.3). Examples of each of the views are shown in Figure 4.4.

Figure 4.3- *Possible view selections from the main window toolbar. The views are midsagittal (M), top (T), bottom (B), and front (F).*

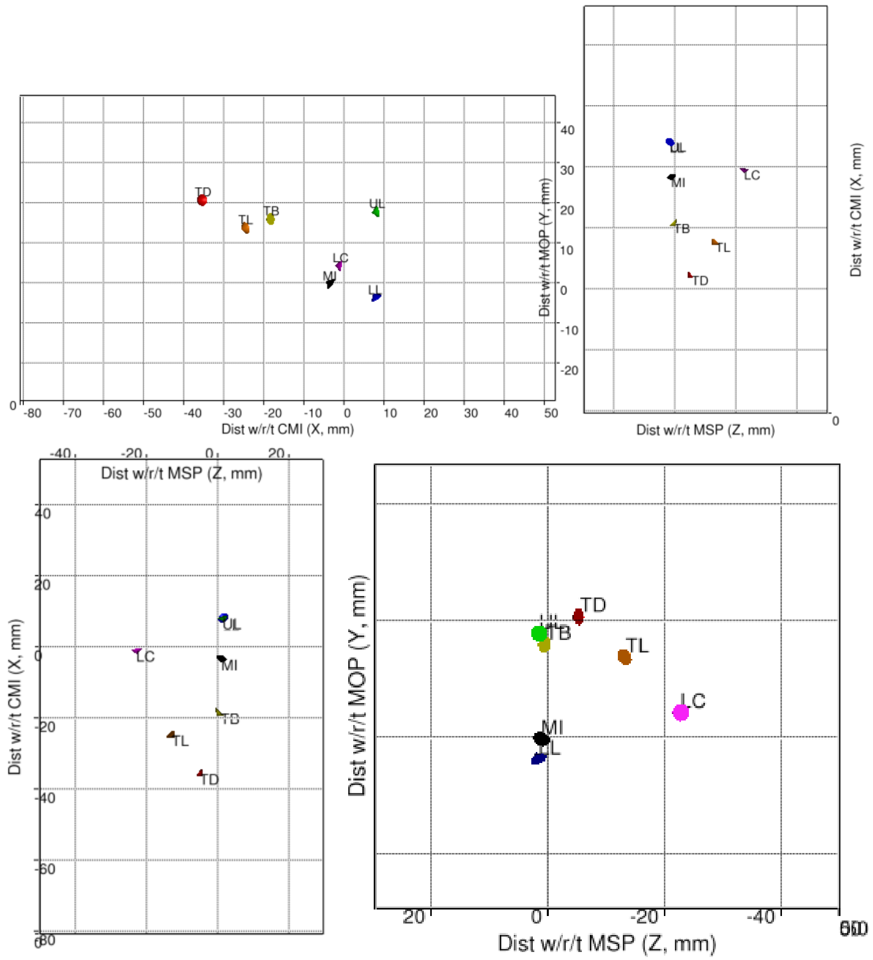


Figure 4.4- *Views of sensors from each of the built-in views: midsagittal (top-left), top (top-right), bottom (bottom- left), and front (bottom-right).*

If none of the default views provide a suitable angle of the sensors, there is also a 3D view, which allows for any arbitrary view of the sensors to be obtained. This 3D interactive mode can be toggled on or off by clicking the icon in the toolbar. In interactive mode, the user can use the mouse's left button to rotate around the center of the axes display. The plot also supports zooming in, either with the mouse's scroll-wheel or holding the right mouse button. An example of the data from an arbitrary view can be seen in Figure 4.5.

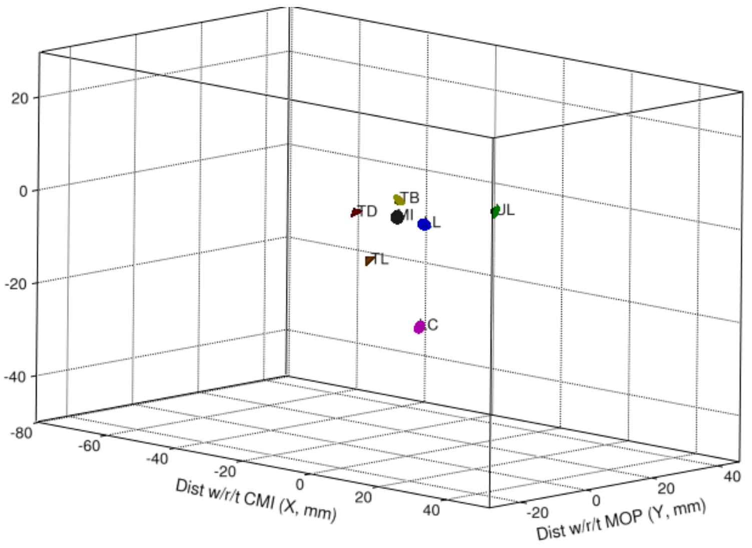


Figure 4.5- *Arbitrary view of the data in the axes.*

When the data are displayed in one of the four default views, the data controller can be used for real-time data playback. The data controller consists of six buttons to control the playback of the audio and sensors, as well as an LCD-style timer that displays the time elapsed and length of the file (Figure 4.6).

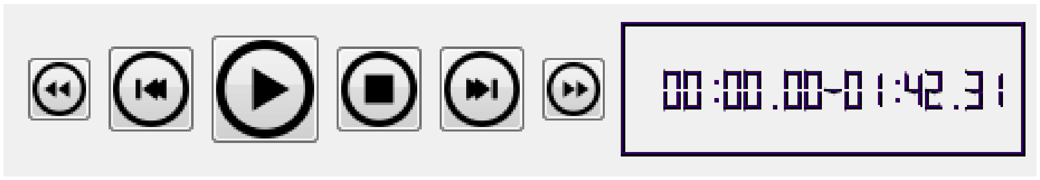


Figure 4.6- *Data controller featuring (from left to right) rewind, step backward, play/pause, stop, step forward, and fast forward buttons.*

The data controller gives the user the ability to sift through the data in either a fine-grained or coarse manner, which makes it excellent for data validation. Errors can be documented by simply playing-back the data record and noting when sensors disappear, fall-off, or otherwise behave irregularly. The data can then be rewound, and the step-forward button can be used to determine the exact time index a problem occurs.

The final major kinematic visualization component is the ability to view sensor trajectories over an interval. After selecting an interval (using the spectrogram widget, discussed shortly), all sensor locations throughout an interval can be viewed by selecting "View trajectories" from the "View" drop-down menu, or by clicking the icon in the toolbar. The trajectories will appear as small, translucent dots in the same color as the sensor whose trajectory they represent (Figure 4.7).

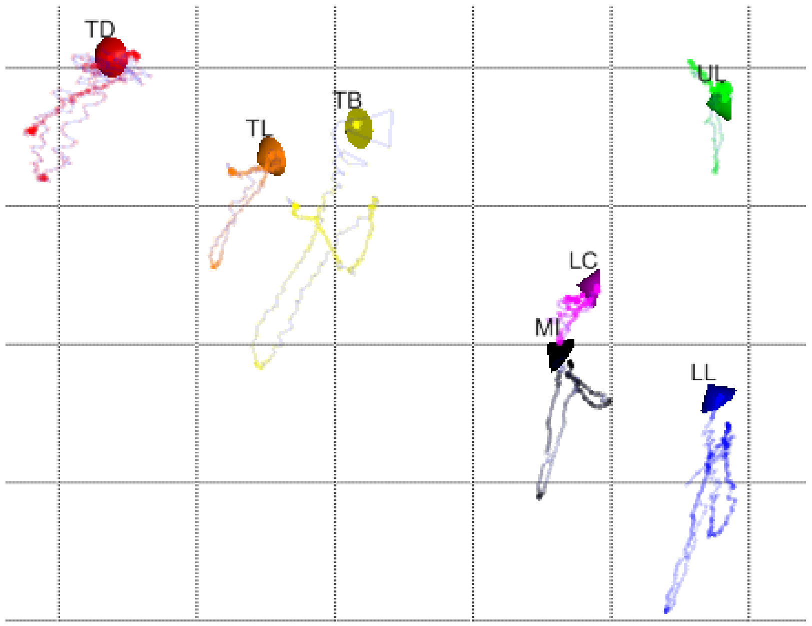


Figure 4.7- *Sensor trajectories for a subject saying the word "copy."*

While the trajectories are displayed, the data can be played back as normal. This includes stepping slowly through the movement of the sensors to watch how they follow the trajectory over the selected interval. In all, the visualization capabilities for observing the kinematic data are many, and provide a means for both data validation and preliminary analysis.

4.6.2 Feature Description- Analysis

The other two components of the visualization tool, the spectrogram and data viewer, provide some additional functionality for visualization, but offer much more in terms of data analysis. The spectrogram shows the frequency content of the record over time, found by taking a discrete Fourier transform (DFT) on each chunk of a signal broken into small (millisecond) windows. The spectrogram on program startup shows the frequency content across the entire record (as in Figure 4.1). Intervals are selectable by clicking and placing a red cursor on either side of the interval and pressing the ⊕ icon. The result is a zoomed-in display of the frequency content of the selected interval (as in Figure 4.8).

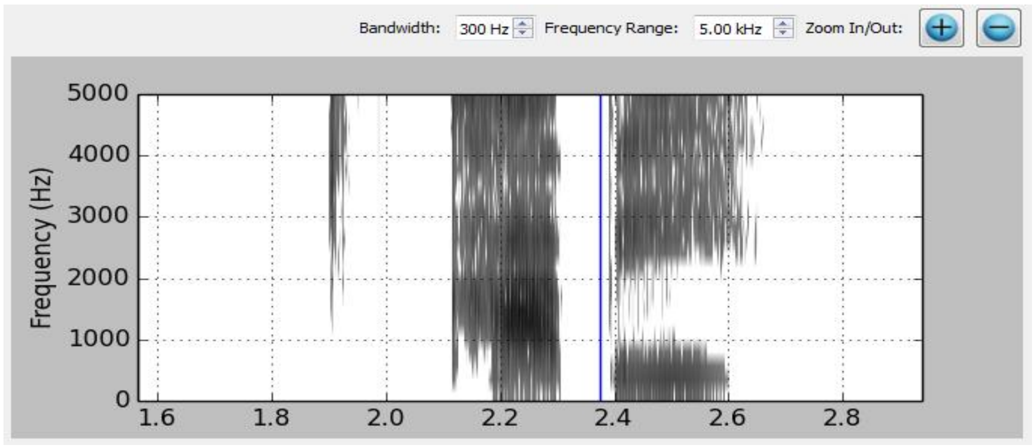

Bandwidth: 300 Hz Frequency Range: 5.00 kHz Zoom In/Out:

Figure 4.8- *Zoomed-in segment of the spectrogram. Note the dark frequency bands which can be used in analysis.*

Upon zooming in, there are several major changes in program operation. The data controller will only play back the portion of the kinematic and audio data selected in the spectrogram, including showing only the trajectories of the sensors within the selected interval. The data can be stepped through slowly or played back in real-time, with a black cursor showing on the spectrogram where the current kinematic and audio data are being played back. If an even closer view is required, an interval within this interval can be selected and zoomed in again. Pressing the ⊖ button will zoom back out to show the previously selected interval.

Aside from these segment selection and playback features, the spectrogram also allows for some configuration of the data shown within it. By default (as shown in Figure 4.8), the spectrogram will show frequencies between DC and 5 kHz, with the bandwidth of the spectrogram set at 300 Hz. By clicking the arrows in the “Frequency Range” box, the range of displayed frequencies can be changed between 0.5 kHz and one-half of the sampling frequency (11.025 kHz for the EMA-MAE database). By clicking the arrows in the “Bandwidth” box, the bandwidth of the spectrogram can be adjusted in intervals of 50 Hz between 50 Hz and 500 Hz. Smaller bandwidths result in a longer DFT and a better frequency resolution. Larger bandwidths result in shorter DFTs, better time resolution and worse frequency resolution. Examples of the same speech segment at 50 Hz bandwidth and 500 Hz bandwidth are shown in Figure 4.9.

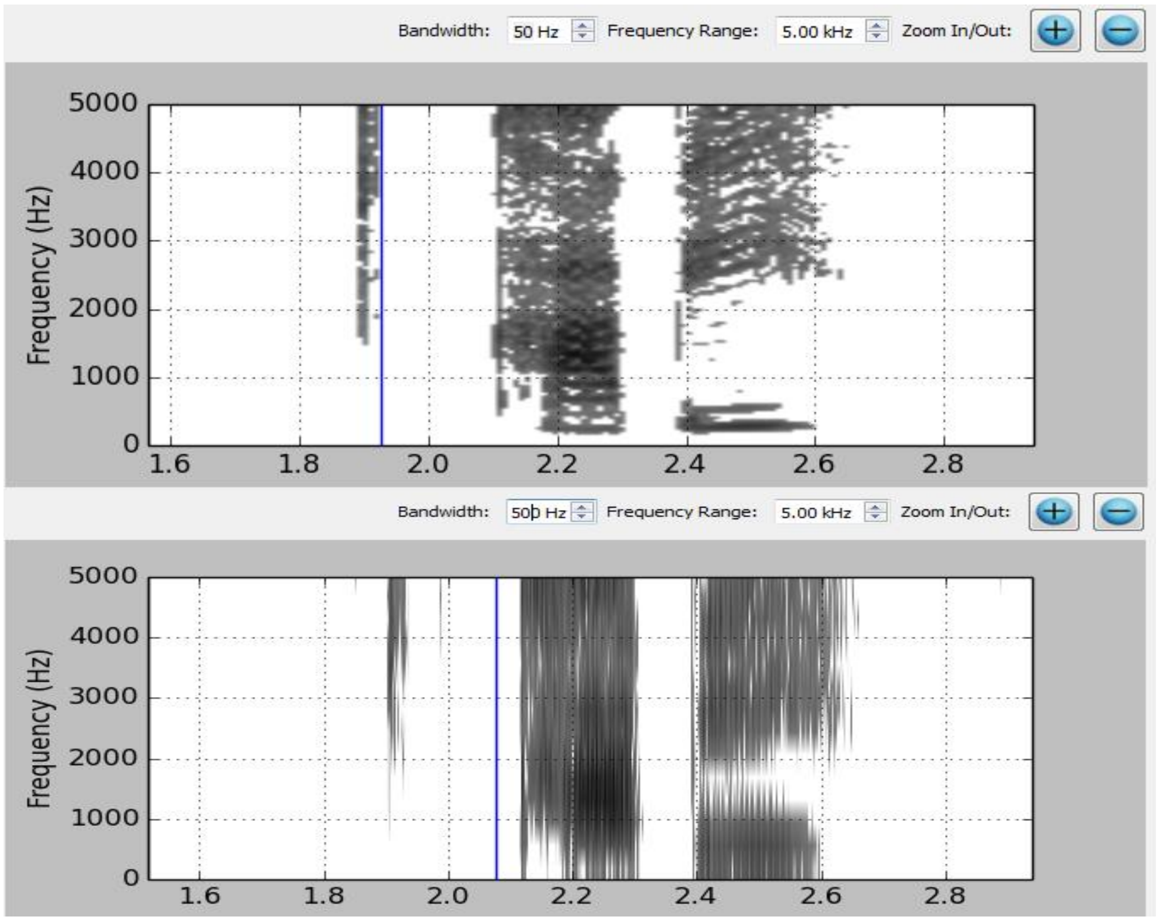
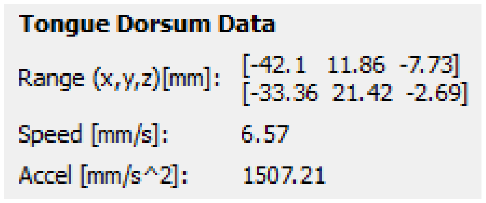


Figure 4.9- *Demonstration of different bandwidths for the spectrogram. The top spectrogram has a bandwidth of 50 Hz, giving great frequency resolution but significant time blurring. The bottom spectrogram was calculated at 500 Hz, giving excellent time resolution, but poor frequency resolution.*

The second major analysis piece is the data viewer, which is on the right side of the main window. Each sensor has its own display section, with the display containing the sensor range, speed, and acceleration (as in Figure 4.10). The speed and acceleration are calculated using the magnitude of the velocity and acceleration components, which are found using the 3-point central difference method.



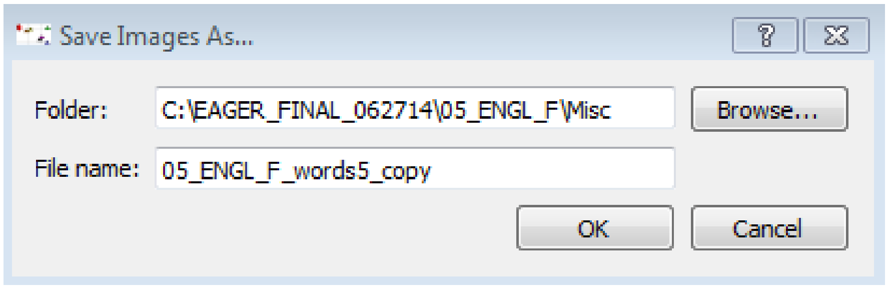
Tongue Dorsum Data

Range (x,y,z)[mm]:	[-42.1 11.86 -7.73] [-33.36 21.42 -2.69]
Speed [mm/s]:	6.57
Accel [mm/s^2]:	1507.21

Figure 4.10- *Data display for the TD (tongue dorsum) sensor. The display includes range, speed, and acceleration.*

As the data plays back, the speed and acceleration values will update rapidly along with the cursor on the spectrogram. This also holds when stepping slowly through the data. The range displayed will be the minimum and maximum values reached during the interval selected in the spectrogram.

If a certain interval is of particular interest, there are two data export functions available for the program. The first type of data export can be done using the icon, which will export the speed and acceleration magnitude values from the selected interval to a text file. The text file provides a header identifying each column, and a time-stamp identifying each data sample. The second type of data export can be done using the icon. This will take a screenshot of both the sensor plot and spectrogram and save them as PNG files. In both types of data export, a file selection dialog will appear, prompting the user for a file name to be used for saving either the images or text file (see Figure 4.11).



Save Images As...

Folder: C:\EAGER_FINAL_062714\05_ENGL_F\Misc

File name: 05_ENGL_F_words5_copy

Figure 4.11- *Dialog for saving the screenshots or text file. The user can select the folder and file name.*

These visualization and analysis features represent the totality of functionality for the current version of the visualization tool, and together allow for in-depth data validation and a fair amount of data analysis.

4.6.3 Building, Distributing and Installing

With the program fully implemented and described, distribution is the next requirement to complete the development cycle. This means building and distributing the program as a standalone executable, capable of being run on computers without a Python interpreter or related software installed. Also, one of the benefits of developing an application with PyQt was the support for a native interface on a variety of operating systems. Making use of this capability requires that the build process for the program be capable of creating distributable application files for any of these systems. To aid in both of these pursuits, the Python community has developed a tool called PyInstaller that is capable of creating standalone applications from the source code originally used to run the program. PyInstaller supports creation of standalone applications for Mac OSX, Linux, or Windows, with usually only one command.

To make use of PyInstaller, the installer requires a list of hooks, which are used to determine the dependencies that are needed for the program to run properly. Hooks for major libraries (like PyQt, Matplotlib, Scipy, and Numpy) come with PyInstaller off-the-shelf, making the process of including these libraries effortless. Visvis, as a less widely-used library, required a custom hook to be created to complete the build process. By embedding the PyInstaller commands and related calls in a single batch script, the program can be built in a single step.

Following the build, the user only requires the “dist” folder created by PyInstaller to run the program. Currently, the dist folder has been placed in the EMA-MAE database software tools folder to be distributed along with the database. To run the program, the user just needs to click the executable in the folder, and an empty main window will appear to begin validation and analysis.

4.6.4 Testing and User Acceptance

Several methods have been used to both verify and validate the visualization tool. Verification includes testing specific functionality and components of the software piece, whereas validation includes determining whether or not specific user needs have been met, and if the tool is easily usable by the intended audience.

Verification was done as components were created and after completion. During development, certain functions were tested individually in their own separate scripts to ensure they were functioning as desired. An example of this is in the file loading section of the program. When a user selects a kinematic file to be played back, the program automatically searches the file path for the corresponding audio file, and selects it for loading once it is found. This functionality was somewhat difficult to achieve, so a separate script was written to test that the method written achieved its aim before being incorporated into the program. This type of verification was done with several different methods, including those used to filter audio signals, retrieve and export data files, and various kinematic calculations.

Another verification component implemented in the software is unit testing, which is done on each class. Every class contains a main method that is executed if the class itself is executed on its own. This main method contains sample data to do basic data loading and display, and allows the user to interact with the specific component for prompt testing of each of the functions. The main method was used to verify each of the individual classes used in creating the program, to ensure that the components of the program were viable on their own prior to integration with other system components.

User validation testing was done without strict protocols, but was conducted using students in the lab. The visualization tool was given to five Speech-Language Pathology students to use as a validation tool for checking for problems in the EMA-MAE database. A document (called Instructions for Project TOAK) was created instructing the users how to load, view, and assess the quality of the data. Several errors and sources of confusion were reported by the students and improvements were made based on this feedback. One example occurred with the file loading functionality described earlier. When the file was not found, users were confused about what happened in the program, as they were used to the audio file being automatically selected. Because of this confusion, a warning dialog was added that informed the user that the audio file could not be found and they would be required to select it manually. This provided an appropriate fail-safe if the audio file could not be found automatically, and reduced confusion about the required user action. Other examples of improvements made after user feedback included automatically setting the axes values, making the step-forward function step forward in smaller time increments, and a few minor aesthetic changes.

Future work with the tool should include stricter validation with use cases that the tool seeks to fulfill. Describing these use-cases in a script and having users assess intuitiveness and ease-of-use of the program would be useful for determining future improvements.

4.6.5 Extensibility and Future Development

Extensibility of the tool is an important consideration, both in terms of expanding functionality and adapting the current functionality to better suit user needs. The program has been built using an object-oriented approach, with a logical class structure that facilitates future development by other programmers. The convention in the program is to make each new user-interface component a new widget, and thus a new class. By organizing the tool in this way, even programmers who are new to PyQt will be able to easily identify which components correspond to which classes. If a problem is found with the sensor data viewer, for example, a developer will be know the problem is within the SensorDataView class, and be able to modify the code accordingly.

Furthermore, the choice to develop using Python and PyQt makes the code readable and compact. Python generally contains far fewer lines of code than similar-functioning C++ or Java code, which does a great deal to aid in the speed of learning about the tool and making future improvements. PyQt features a large amount of built-in widgets that can be combined and modified ot form new widgets that might be desired in a newer version of the visualization tool.

While the visualization tool in its current form provides a core set of features for EMA visualization and analysis, there are a large number of upgrades that could be made to future releases of the tool. The first and most useful upgrade would be to add the ability to index the audio and kinematic files, labeling time segments with start and end times for phonemes. These indices would be labeled within the program and saved to a label file as the user marked them, along with a code number or IPA character identifying the labeled interval.

A few other useful features would involve creation of new dialogs allowing the user to configure various widgets within the program. One example is a dialog for the spectrogram, allowing for more changes than just the frequency range and bandwidth. These could include changes in dynamic range and noise floor to adjust the darkness of the various decibel levels. It could also include turning on a formant tracking algorithm that could plot formant estimates on the spectrogram, be adjusted and confirmed by the user, and exported to a text file.

Another dialog that would increase the flexibility of the tool would be a sensor configuration dialog. Upon opening the tool for the first time (or whenever data is collected in a new format), the tool would prompt the user for the number of sensors in the configuration, as well as sensor names, labels, columns containing the different portions of sensor data in the kinematic file, and even sensor color when plotted on screen. This would make the tool usable for other studies outside of just the EMA-MAE database, and vastly increase its potential impact.

Other additions to the tool could include integration of other components mentioned in previous chapters. An integrated biteplate correction tool, tongue mesh visualization function, or jaw angle tracking function would all be excellent additions to the list of capabilities. The tongue mesh, specifically, if integrated with a palate mesh in the sensor plot, could provide excellent views of the constriction points in the oral cavity that strongly influence articulation. They would also allow for possible numerical calculation of vocal tract cross-sectional area for a sizable portion of the oral cavity, further bolstering analyses related to articulation.

5. CONCLUSIONS

This thesis has presented a set of new software tools and analysis methods for working with EMA data. Beginning with an explanation of EMA's benefits, its operating principle, and its data structure, the basics of EMA were covered in depth. This included a discussion of quaternion data and an associated tutorial on using quaternions to represent rotations and orientations. The concept of biteplate correction, its necessity, and its implementation were briefly discussed. A new biteplate correction method was introduced for placing the quaternion data in a physiologically meaningful coordinate space. Biteplate correcting the quaternion data alongside traditional positional biteplate correction is essential for making use of the orientation data in kinematic analyses.

Applications for quaternion orientation data were shown in the form of two articulatory representations: single-sensor jaw angle measurement, and formation of a tongue surface interpolation algorithm. The single-sensor jaw angle measurement technique produced results that were superior to traditional, position-based measurement with nearly equivalent angles, and a lower variance due to less noise. The tongue mesh showed reasonable accuracy in predicting virtual fleshpoint trajectories and also provided a means of visualizing gross changes in tongue shape and position. Both of these representations showed promise for use in future speech kinematic studies.

Finally, a new EMA visualization tool was presented. The visualization tool allows users to load, observe in real time, and do basic analysis on EMA data over an interval, including calculations of sensor speed, acceleration, and range. Compared to other pieces of software for EMA visualization and analysis, the tool offers similar analysis capabilities, with superior visualization that allows for 3D viewing and couples the position data with orientation information.

In total, the thesis contains both the knowledge and tools to simplify EMA data processing for all researchers that are interested in using the EMA-MAE database for speech kinematic analysis. In explaining and demonstrating the use of quaternion orientation data, it is hoped that future EMA studies will make full use of the orientation data to characterize articulation in the most complete manner possible. The orientation data provided by modern 3D-EMA is an important component of sensor data that has been almost completely neglected thus-far in the literature. By providing others with the means to maximize the knowledge gained from EMA datasets, it is hoped that the contents of this work will help to move the fields related to speech kinematic analysis forward. In providing tools for increased research productivity, solutions to complex questions in speech production can be found more quickly, with major benefits for individuals requiring speech training and rehabilitation.

6. BIBLIOGRAPHY

[1] H. Horn, G. Goz, M. Bacher, M. Mullauer, I. Kretschmer and D. Axmann-Krcmar, "Reliability of electromagnetic articulography recording during speaking sequences," Eur. J. Orthod., vol. 19, pp. 647-655, Dec, 1997.

[2] P. Birkholz, B. J. Kröger and C. Neuschaefer-Rube, "Model-based reproduction of articulatory trajectories for consonant-vowel sequences," IEEE Transactions on Audio, Speech and Language Processing, vol. 19, pp. 1422-1433, 2011.

[3] A. Wrench and K. Richmond, "Continuous speech recognition using articulatory data," 2000.

[4] S. Fuchs, P. Perrier, C. Geng and C. Mooshammer, "What role does the palate play in speech motor control? Insights from tongue kinematics for German alveolar obstruents," Speech Production: Models, Phonetic Processes, and Techniques, pp. 149-164, 2006.

[5] J. R. Green and Y. Wang, "Tongue-surface movement patterns during speech and swallowing," J. Acoust. Soc. Am., vol. 113, pp. 2820-2833, 2003.

[6] J. Berry, A. Kolb, C. North and M. T. Johnson, "Acoustic and kinematic characteristics of vowel production through a virtual vocal tract in dysarthria," in Fifteenth Annual Conference of the International Speech Communication Association, 2014, .

[7] A. Marchal and W. J. Hardcastle, "ACCOR: Instrumentation and database for the cross-language study of coarticulation," Lang. Speech, vol. 36, pp. 137-153, 1993.

[8] S. Narayanan, E. Bresch, P. K. Ghosh, L. Goldstein, A. Katsamanis, Y. Kim, A. C. Lammert, M. I. Proctor, V. Ramanarayanan and Y. Zhu, "A multimodal real-time MRI articulatory corpus for speech research." in Twelfth Annual Conference of the International Speech Communication Association, 2011, pp. 837-840.

[9] J. M. Scobbie, A. Turk, C. Geng, S. King, R. Lickley and K. Richmond, "The Edinburgh Speech Production Facility DoubleTalk Corpus," Proceedings of 14th Interspeech, Lyon, 2013.

[10] A. Ji, J. J. Berry and M. T. Johnson, "The electromagnetic articulography mandarin accented english (EMA-MAE) corpus of acoustic and 3D articulatory kinematic data," in Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference On, 2014, pp. 7719-7723.

[11] D. Byrd, C. P. Browman, L. Goldstein and D. Honorof, "Magnetometer and X-ray microbeam comparison," Proc.14th Int.Congr.Phonetic Sci, pp. 627-630, 1999.

[12] S. Kiritani, K. Itoh and O. Fujimura, "Tongue-pellet tracking by a computer-controlled x-ray microbeam system," J. Acoust. Soc. Am., vol. 57, pp. 1516-1520, 1975.

[13] J. Westbury, P. Milenkovic, G. Weismer and R. Kent, "X-ray microbeam speech production database," J. Acoust. Soc. Am., vol. 88, pp. S56-S56, 1990.

[14] J. S. Perkell, M. H. Cohen, M. A. Svirsky, M. L. Matthies, I. Garabieta and M. T. Jackson, "Electromagnetic midsagittal articulometer systems for transducing speech articulatory movements," J. Acoust. Soc. Am., vol. 92, pp. 3078-3096, 1992.

[15] A. Zierdt, P. Hoole and H. -. Tillmann, "Development of a system for three-dimensional fleshpoint measurement of speech movements," in Proceedings of the XIVth International Congress of Phonetic Sciences, San Francisco, CA, 1999, .

[16] Y. Yunusova, J. R. Green and A. Mefferd, "Accuracy assessment for AG500, electromagnetic articulograph," Journal of Speech, Language, and Hearing Research, vol. 52, pp. 547-555, 2009.

[17] J. J. Berry, "Accuracy of the NDI Wave speech research system," Journal of Speech, Language, and Hearing Research, vol. 54, pp. 1295-1301, 2011.

[18] J. S. Perkell, M. H. Cohen, M. A. Svirsky, M. L. Matthies, I. Garabieta and M. T. Jackson, "Electromagnetic midsagittal articulometer systems for transducing speech articulatory movements," J. Acoust. Soc. Am., vol. 92, pp. 3078-3096, 1992.

[19] P. Hoole, A. Zierdt and C. Geng, "Beyond 2D in articulatory data acquisition and analysis," in Proceedings of the Fifteenth International Congress of Phonetic Sciences, Barcelona, 2003, pp. 265-268.

[20] W. R. Hamilton and W. E. Hamilton, Elements of Quaternions. London: Longmans, Green, & Company, 1866.

[21] J. B. Kuipers, Quaternions and Rotation Sequences. Princeton university press Princeton, 1999.

[22] A. J. Hanson, "Visualizing quaternions," in International Conference on Computer Graphics and Interactive Techniques: ACM SIGGRAPH, Los Angeles, CA, USA, 2005, .

[23] S. L. Altmann, "Hamilton, Rodrigues, and the quaternion scandal," Mathematics Magazine, pp. 291-308, 1989.

[24] S. L. Altmann, Rotations, Quaternions, and Double Groups. Courier Corporation, 2005.

[25] J. S. Perkell, M. Zandipour, M. L. Matthies and H. Lane, "Economy of effort in different speaking conditions. I. A preliminary study of intersubject differences and modeling issues," J. Acoust. Soc. Am., vol. 112, pp. 1627-1641, 2002.

[26] J. R. Westbury, "X-ray microbeam speech production database user’s handbook [Software manual] ," 1994.

[27] S. Narayanan, A. Toutios, V. Ramanarayanan, A. Lammert, J. Kim, S. Lee, K. Nayak, Y. Kim, Y. Zhu and L. Goldstein, "Real-time magnetic resonance imaging and electromagnetic articulography database for speech production research (TC)," J. Acoust. Soc. Am., vol. 136, pp. 1307-1311, 2014.

[28] R. N. Henriques and P. van Lieshout, "A comparison of methods for decoupling tongue and lower lip from jaw movements in 3D articulography," Journal of Speech, Language, and Hearing Research, vol. 56, pp. 1503-1516, 2013.

[29] D. J. Ostry, E. Vatikiotis-Bateson and P. L. Gribble, "An examination of the degrees of freedom of human jaw motion in speech and mastication," Journal of Speech, Language, and Hearing Research, vol. 40, pp. 1341-1351, 1997.

[30] S. Maeda, "Compensatory articulation during speech: Evidence from the analysis and synthesis of vocal-tract shapes using an articulatory model," in Speech Production and Speech ModellingAnonymous Springer, 1990, pp. 131-149.

[31] C. Kroos, "Using sensor orientation information for computational head stabilisation in 3D electromagnetic articulography (EMA)," in Tenth Annual Conference of the International Speech Communication Association, 2009, .

[32] P. Hoole and A. Zierdt, "Five-dimensional articulography," Speech Motor Control: New Developments in Basic and Applied Research, pp. 331-349, 2010.

[33] I. Steiner and S. Ouni, "Progress in animation of an EMA-controlled tongue model for acoustic-visual speech synthesis," 1201.4080, 2012.

[34] Y. C. Chiang, F. P. Lee, C. L. Peng and C. T. Lin, "Measurement of tongue movement during vowels production with computer-assisted B-mode and M-mode ultrasonography," Otolaryngol. Head. Neck. Surg., vol. 128, pp. 805-814, Jun, 2003.

[35] M. Stone and A. Lundberg, "Three-dimensional tongue surface shapes of English consonants and vowels," J. Acoust. Soc. Am., vol. 99, pp. 3728-3737, 1996.

[36] M. Hasegawa-Johnson, S. Pizza, A. Alwan, J. S. Alwan and K. Haker, "Vowel category dependence of the relationship between palate height, tongue height, and oral area," Journal of Speech, Language, and Hearing Research, vol. 46, pp. 738-753, 2003.

[37] S. Narayanan, K. Nayak, S. Lee, A. Sethy and D. Byrd, "An approach ot real-time magnetic resonance imaging for speech production," Journal of the Acoustical Society of America, vol. 115, pp. 1771-1776, 2004.

[38] A. Niebergall, S. Zhang, E. Kunay, G. Keydana, M. Job, M. Uecker and J. Frahm, "Real-time MRI of speaking at a resolution of 33 ms: Undersampled radial FLASH with nonlinear inverse reconstruction," Magnetic Resonance in Medicine, vol. 69, pp. 477-485, 2013.

[39] O. Engwall, "Combining MRI, EMA and EPG measurements in a three-dimensional tongue model," Speech Commun., vol. 41, pp. 303-329, 2003.

[40] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in Proceedings of the 1968 23rd ACM National Conference, 1968, pp. 517-524.

[41] G. Farin, "Triangular bernstein-bézier patches," Comput. Aided Geom. Des., vol. 3, pp. 83-127, 1986.

[42] J. C. Hart, G. K. Francis and L. H. Kauffman, "Visualizing Quaternion Rotation," ACM Transactions on Graphics, vol. 13, pp. 256-276, 1994.

[43] F. L. Markley, Y. Cheng, J. L. Crassidis and Y. Oshman, "Averaging quaternions," Journal of Guidance, Control, and Dynamics, vol. 30, pp. 1193-1197, 2007.

[44] R. W. Clough and J. L. Tocher, "Finite element stiffness matrices for analysis of plates in bending," in Proceedings of Conference on Matrix Methods in Structural Analysis, 1965, pp. 515-545.

[45] R. Patel, K. Connaghan, D. Franco, E. Edsall, D. Forgit, L. Olsen, L. Ramage, E. Tyler and S. Russell, "“The Caterpillar”: A Novel Reading Passage for Assessment of Motor Speech Disorders," American Journal of Speech-Language Pathology, vol. 22, pp. 1-9, 2013.

[46] J. J. Berry, "Accuracy of the NDI Wave speech research system," Journal of Speech, Language, and Hearing Research, vol. 54, pp. 1295-1301, 2011.

[47] N. Nguyen, "A MATLAB toolbox for the analysis of articulatory data in the production of speech," Behavior Research Methods, Instruments, & Computers, vol. 32, pp. 464-467, 2000.

[48] M. Tiede, "MVIEW: software for visualization and analysis of concurrently recorded movement data," Haskins Laboratory, 2005.

[49] S. Ouni, L. Mangeonjean and I. Steiner, "VisArtico: A visualization tool for articulatory data," in 13th Annual Conference of the International Speech Communication Association-InterSpeech 2012, 2012, .

[50] M. Summerfield, Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming. Pearson Education, 2007.

[51] M. Huckvale, "VTDemo- Vocal Tract Acoustics Demonstrator," 2009.

[52] S. Ghosh, "VTCalcs and VTSynth," vol. 1.0, 1999.

[53] A. W. Rix, J. G. Beerends, M. P. Hollier and A. P. Hekstra, "Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and

codecs," in Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference On, 2001, pp. 749-752.

[54] K. Mustafa and I. C. Bruce, "Robust formant tracking for continuous speech with speaker variability," Audio, Speech, and Language Processing, IEEE Transactions On, vol. 14, pp. 435-444, 2006.

7. APPENDIX A

This appendix contains high-level descriptions of the code used to create the software tools presented in the thesis. For each project, a link is provided to the source code that can be viewed online at GitHub. Some files are included across projects, for instance loadtsv, which is used in all tools to load kinematic data into the given program. These files contain their own descriptions in the comments of the code.

7.1 Biteplate Correction Code

The biteplate correction code is a set of MATLAB functions and scripts that are specialized for biteplate correcting both the position and orientation data as taken using the Marquette University Speechlab Protocol. The main files required for the biteplate correction are BPRotation and BiteplateCorrect. BPRotation is run first, and returns the average positions of the MS and OS sensors, as well as the quaternion rotation required to transform between the axes-flipped coordinate space and the biteplate-corrected coordinate space (See Chapter 2, Table 2.1). These values are saved in the Matlab workspace as BP_Info.mat. This .mat file is then used in BiteplateCorrect to correct a file associated with the BP_Info file.

This work's contribution to the biteplate correction process, the quaternion correction, can be found in the file correctQuat. CorrectQuat uses the rotation provided in BP_Info to place the quaternions in the biteplate-corrected coordinate space, as described in Section 2.4. It utilizes a function called getQuat, which is an implementation of Equation(2.4.4). The remainder of the functions in the biteplate correction code are taken from the Matlab Quaternion Toolbox, provided on the Matlab File Exchange.

Future work with the biteplate correction code will likely include creation of a standalone, GUI-based tool that does not use MATLAB, so it can be used by non-technical users who do not have a MATLAB license.

All of the biteplate correction code can be found at:
https://github.com/andrewkolb08/BiteplateCorrection

7.2 Jaw Angle Code

The jaw angle code is also a set of MATLAB scripts and functions. The two files of interest are finalJawAngle and getJawAngles. FinalJawAngle is a script that demonstrates the two different jaw angle measurement methods described in Section 3.1. The position and quaternion method are both implemented and plotted, along with the acoustic data taken from the subject at the time. GetJawAngles returns the quaternion-derived angles and position-derived angles from an entire file that is passed in. It also requires the columns in the file where the midsagittal incisor and molar data are stored.

In the future, the jaw angle tool should be integrated into the visualization tool to provide a real-time display of the jaw angle over time, with the ability to export orientation-derived jaw angles to file.

All of the jaw angle code can be found at:
https://github.com/andrewkolb08/JawAngle

7.3 Tongue Mesh Code

The interpolated tongue mesh code is another group of MATLAB scripts. It is somewhat complicated with several different functions used to create the final mesh output. The script newestTongueMesh runs a demonstration of overall mesh creation for a single word uttered by a single subject (05_ENGL_F). The demonstration plots a spectrogram of the subject's speech alongside a tongue mesh. A scrollbar under the tongue mesh can be used to show the tongue at different time points, which are identified by a black cursor on the spectrogram. The time interval plotted can be changed by adjusting the startTime and endTime variables in the script.

The beginning of the script implements quaternion baseline adjustment, using the function newAdjustTongueQuats. This function determines the quaternion baseline adjustment using the averaging discussed in Section 3.2. AdjustTongueQuats, a different function, implements the position-based quaternion adjustment scheme, also detailed in Section 3.2. The script then calls getTonguePoints, which is a function to calculate the centroid heights at each of the triangles formed by the Clough-Tocher split. The function findTriVal implements Equations (3.4.3) through (3.4.6) to get the new interpolated tongue values. When these values are returned to getTonguePoints, the entire tongue mesh for one time point is created and returned to the script to be saved for display.

A tool for looking at an arbitrary time segment for a subject is the function viewTongueAndSpec. This function accepts as an input a given sound file, kinematic file, start time and end time to be plotted. It will display a spectrogram and tongue mesh, just as was done in the example file newestTongueMesh.

All of the tongue mesh code can be found at:
https://github.com/andrewkolb08/TongueMesh

7.4 Visualization Tool Code

The visualization tool is by far the most complicated of any of the software tools. It is implemented in Python using the combination of software libraries discussed in Sections 4.3 and 4.4. There are five major components to the visualization tool. The first is the main window, called visualizationTool. The visualizationTool serves as the container for all other widgets used in the program. It contains several shortcuts, a toolbar, a file and menu. The dataController is the main widget inside the main window. The dataController class contains the sensor plot, the spectrogram, the timer, control buttons, and sensor data viewer. Each of these are implemented as their own class.

The sensor plot is a 3D visvis plot that contains cone objects representing the sensors. While playing, the cones' positions and orientations are constantly updated, as are their associated labels. The sensors are instances of their own Sensor class, which holds all data pertinent to displaying information about a given sensor. The motion of the sensor is updated using an index variable and a timer which governs how often an update occurs, and the new value to be displayed. The view of the plot can be changed using the camera object associated with the plot. The axes labels are controlled by an Axis object that is associated with the plot.

The spectrogram is also its own separate class. The spectrogram contains a Matplotlib spectrogram plot, with several buttons and scroll boxes to control how the data is displayed. The plot has an event handler which waits for the user to click the plot, and then places a cursor at the point they clicked. The time intervals displayed are saved so the user can proceed back to their previously viewed interval when clicking the 'minus' button. The audio data is filtered using a high-pass pre-emphasis filter to balance the frequency response of the audio, which is biased toward low-frequency information.

The timer is the clock shown while the sensors are moving, keeping track of the time point being displayed in the record. The timer includes and LCD timer class which accepts as input a number of seconds elapsed. The LCD timer is updated to reflect the current index every 100 ms.

The timer is synchronized with the control buttons, which control how the time moves. The buttons are contained in the dataController class, and have associated functions called when any of them are pressed. Buttons are enabled and disabled as needed when different data views are enabled. For instance, the fast forward/rewind buttons will be disabled when the 3D view is activated, as the user cannot play through the data in this mode.

The final component is the sensor data viewer class, which holds displays for the range, speed, and acceleration of a sensor. When it is instantiated, it is automatically associated with a specific sensor whose data it will display for the current record. These sensor data viewer instances are updated as the data plays, and their contents can be exported to file using the export menu.

There are several other ancillary files that go along with the visualization tool. These include the build.bat script, which builds the executable from source code; the qrc_resources file, which holds the locations of pictures and icons used by the program; and the resource compilation batch file, which is used to compile qrc_resources and include these resources in the code. There is certainly much more to be said in-depth about the components of the visualization tool, and many more notes are available in the comments of the code. Each class features a class description at the top, with many comments throughout the functions to help a new developer understand what they do and how they accomplish it. Also, for a great tutorial on Python and PyQt4, refer to Mark Summerfield’s book, *Rapid GUI Programming with Python and Qt* [50]. The book provides all the materials to understand the visualization tool at a very deep level, and should be a reference for anyone who seeks to continue its development.

All of the visualization tool code can be found at:

https://github.com/andrewkolb08/TOAK

8. APPENDIX B

This appendix contains descriptions of other associated EMA tools that are not specifically mentioned in the body of the thesis. This includes two major components: RASS Tools and R Tools. These tools are currently only suitable for in-house data processing, so no links will be included to the source code.

8.1 RASS Tools

RASS is the abbreviation for the Marquette Speech and Swallowing Lab's Rehabilitory Articulatory Speech Synthesizer. RASS is a virtual vocal tract that is controllable by a subject who has EMA sensors attached to their articulators. By moving their lips, jaws, and tongue, the subject controls the synthesizer to create an acoustic output. RASS is a modified version of Mark Huckvale's Vocal Tract Demonstrator (VTDemo) Software [51], which is in turn based on a vocal tract model created by Shinji Maeda [30]. The tools created for working with RASS include a new biteplate implementation, mapping scheme adjustment, playback of previous subject data, and a MATLAB version of RASS for testing new mapping schemes. The tools created for working with RASS include a new biteplate implementation, mapping scheme adjustment, playback of previous subject data, and a MATLAB version of RASS for testing new mapping schemes.

Before describing the tools, a brief description of the VTDemo workflow needs to be provided. After a subject is connected to the NDI Wave system, the typical process for setting up a subject needs to be done, including biteplate correction. In addition to these steps, the synthesizer must be tuned to work with the subject's unique anatomy. As an example, one subject's jaw might be all the way open and provide a jaw sensor y-value of -25 millimeters. Another subject might have their jaw all the way open at -30 millimeters. Regardless of these anatomical differences, the synthesizer must recognize these positions as maxima for open jaw position. Similar tuning must be done for the tongue and lip sensors.

Many different schemes are possible for mapping the articulator positions to synthesizer positions. Future research will include determining better schemes for doing this mapping, as it has an outsize effect on the quality of speech the subject can produce through the synthesizer. For this reason, the ability to have a mapping scheme that is able to be fine-tuned is very desirable.

VTDemo Verification Scheme

The first piece of software to aid in this setup process was a series of dialogs that walks a user through the workflow of preparing a subject. This was completed as part of a senior design team project. The dialogs include an initial setup dialog, a vowel-space setup dialog, and a verification/playback dialog. The initial setup dialog prompts the user to collect a biteplate record and a sensor boundary record. As previously described, this sensor boundary record will determine the minimum and maximum positions of the articulators to be mapped onto the minimum and maximum values of the synthesizer.

Following collection of a sensor boundary file, the vowel-space setup dialog aids in mapping vowels from the subject onto the synthesizer. Several (up to 8) vowel records can be collected and incorporated as part of the mapping scheme for the synthesizer. These values contribute vowel minimum or vowel maximum sensor positions to the mapping scheme.

After the vowel records have been collected, the mapping scheme needs to be tested to see if it adequately maps the movement of a subject's articulators onto the movements of the synthesizer. The verification dialog aids in this pursuit. In the verification dialog, any previously recorded record for a subject can be played back via the synthesizer using the current mapping scheme. If the record was of a subject producing the vowel sound "eeee," the synthesizer should produce the vowel sound “eeee,” and match the subject as closely as possible. By comparing the actual acoustics of the subject and the acoustics formed by the virtual vocal tract, mapping quality can be assessed.

Mapping Scheme Adjustment

If the mapping is determined to be satisfactory, experiments utilizing the synthesizer can proceed. However, if adjustments need to be made, the verification dialog provide a means to make changes to the mapping. Using two sliders, the min, max, vowel min and vowel max values can be adjusted upward or downward by a certain percentage. This can be done for any of the jaw, lip, or tongue parameters used by the synthesizer. Following the adjustment, the synthesized speech can be played back again using the adjusted mapping scheme, and assessed again for quality. This process can be done repeatedly until the best mapping scheme is obtained.

Offline Synthesis

While the rapid mapping adjustment is a good feature for changing the mapping when the subject is hooked up to the system, often, it is desirable to collect data from a subject and determine the optimal mapping scheme offline. The subject might become uncomfortable or impatient when the mapping scheme is being iteratively adjusted, so simply collecting some data and allowing them to return later can increase efficiency and subject comfort level.

If a subject has a biteplate record, a sensor boundary record, and any number of vowel records, VTDemo has been adjusted so it will playback these records from the kinematic data alone, and does not need to be connected to a subject to synthesize speech from kinematic data. The workflow is exactly the same as if a subject is connected to the system, except that existing files are loaded and played back, rather than recorded and played back. Pre-recorded files allow for the articulator to synthesizer mapping to be adjusted while a subject is gone, and a reasonable mapping to be used when they return. It also allows for "virtual" subjects, whose data was not even collected using RASS at all, but has been put in the proper format for RASS to read in and synthesize speech based on the data. This opens up RASS to studies using existing datasets, and not exclusively for real-time speech synthesis.

New Mapping Scheme Assessment

Fine-tuning the mapping scheme is currently possible in RASS, but implementation of a new mapping scheme is not. As stated, future research must include improved mapping schemes for subjects to synthesize the most realistic speech possible. Toward this end, as part of a Digital Processing of Speech Signals project, a MATLAB version of the RASS was created for new mapping schemes to be implemented and tested rapidly.

When the kinematic data has been transformed into synthesis parameters by a test mapping scheme, a function loads the parameters and plays the synthesized speech. This is done using a modified version of the MATLAB Maeda synthesizer from Ghosh [52], combined with objective quality assessment metrics. The quality metrics included both PESQ [53]and formant distortion [54]. The tool can thus rapidly and objectively assess the mapping quality for a given mapping scheme, expediting the research related to new articulator to synthesizer mappings.

Quaternions in RASS

Previously, RASS ignored quaternion data and made no use of quaternions or the quaternion data provided by the EMA sensors. Modifications were made to RASS to load the quaternion data into the program for future use. These modifications also included a group of functions for quaternion mathematical operators. This includes rotating a vector using a quaternion, multiplying two quaternions together, adding/subtracting two quaternions, and getting the quaternion representing the rotation between two vectors. Other vector operations (cross product, dot product, L^2-norm, etc.) were also implemented. The boost library (http://www.boost.org/) was used for its quaternion class, and all operators utilizes this class as the data structure to hold quaternion data.

Improved Biteplate Correction

Because biteplate correction is used in RASS, the new biteplate correction scheme for both the position and orientation data was implemented to improve the speed and accuracy of biteplate correction. Previously, RASS used Euler angles from the biteplate record to determine the rotation needed to move from the axes-flipped space to the biteplate-corrected space. This algorithm contains singularities that make it unreliable, and consumed more time and memory to implement the rotation of the subject's data into the biteplate-corrected space. The new quaternion-based biteplate correction scheme is exactly analogous to the biteplate code presented in Appendix A.

8.2 R Tools

R is a programming language traditionally used for statistical processing and analysis. It is the open-source version of S/S+, a language developed by Bell Labs beginning in 1976. R provides excellent built in capabilities for manipulating matrices and tables of data, and features a large community of researchers who contribute to the functions and capabilities of the software. Large numbers of R packages are freely available online to be used as part of any R-based project.

Because of its ability to handle large datasets and many built in statistical operators, R has become the programming language of choice in the Marquette Speech and Swallowing Lab for working with EMA data. Several functions have been developed for working with the EMA data collected by the NDI Wave, described here.

Basic Data Manipulations

A series of functions were written for copying, renaming, and moving kinematic data files. This includes the ability to load an NDI Wave-style kinematic data file into R, as well as write it back out to file following some data manipulations. One major manipulation implemented was the ability to linearly interpolate NaN (not a number) instances in the data or set them to some nonsense value if the run of NAs is longer than some specified length of time.

Numerical Derivatives and Integrals

As part of many kinematic analyses, measures like speed, velocity, or acceleration are used to characterize and differentiate between subject movements. Built in functions exist in R to measure single dimension velocity and acceleration, but this arbitrarily reduces the movements produced by the articulators to three principle directions. Combining the X, Y, and Z velocities together to produce a single velocity and acceleration magnitude were implemented as their own functions. Integration was also implemented numerically to get values such as distance travelled.

A non-built-in function that was also created was jerk cost, presented in [55]. Jerk is the third derivative of position, and jerk cost is the accumulation of jerk over time (corresponding to an integral operator). A function was written to calculate jerk cost in R, using the numerical derivative and integration functions mentioned previously.

Indexing and Segmenting

Breaking a total record into pieces of interest is a common task in speech kinematic processing. Functions have been created to take in a list of time values of interest, and output the kinematic data corresponding to these time intervals. This includes both getting kinematic data and formant data, provided that an FBW file (from TF32) is also given to the function. Codes in the indexing file can be used to specify what phoneme is being extracted from the kinematic data, and the extracted data matrix will contain that information in its name.

Plotting and Data Display

A variety of plotting and display functions have been developed. Given some kinematic data for a vowel, the convex hull of the vowel space can be plotted. The area of this hull can also be calculated and displayed along with the data points themselves. There is also an example of this in [55] . Similarly, the ellipse giving the first two principle components and standard deviation of the data distribution can be plotted over the data. Box and whisker plots and scatter plots have been implemented for use with arbitrary kinematic data.

Convenience Functions

Several functions to make programming in the R environment more convenient were also created. This includes backing up all functions created in a workspace to an external folder (outside of R), setting the new working directory based on the last current working directory, saving all variables in the workspace with a timestamp provided, and resetting the workspace (removing variables while keeping functions).