Methods for Physiologic Tremor Characterization, Mitigation, and Modeling

by

Uriel Magaña-Salgado

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2023

Author .....
Department of Mechanical Engineering
January 20, 2023

Certified by .....
Brian Anthony
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Nicolas Hadjiconstantinou
Graduate Officer

Methods for Physiologic Tremor Characterization, Mitigation, and Modeling

by

Uriel Magaña-Salgado

Submitted to the Department of Mechanical Engineering
on January 20, 2023, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

Physiologic tremors exist in all healthy individuals, and occur in moments of excitement, anxiety, or muscle activation. They have been commonly observed in surgeons and other occupations requiring precise movements, but unlike tremors from Parkinson's disease or Essential Tremor, they are often disregarded in clinical and research settings as they are not linked to any neurological disease, nor disturb one's daily life. The magnitude of physiologic tremors has been observed to increase through stressful or fatiguing experiences, and decrease through relaxation exercises or medication; however, the combined mechanical, electrical, and physiological changes in the body render it a challenging phenomenon to study.

Advancements in physiological monitoring have allowed researchers to characterize tremors. Accelerometry and surface electromyography are often used to measure tremor patterns, both practical methods for measuring surface-level changes. Imaging modalities like ultrasound, on the other hand, are less prevalent techniques for these scenarios. Combining these methods can present a more holistic observation of physiological changes involved in an emerging tremor, potentially guiding research towards a more complete understanding of their cause and impact on the body.

This thesis highlights approaches to detect and characterize physiologic tremors in the upper limbs. It describes the methods used to process the signals and images acquired via the various modalities, and relates the changes among modalities to each other. Finally, the results of this analysis are highlighted, presenting strategies to mitigate tremors and a new understanding of the biomechanics behind them.

Thesis Supervisor: Brian Anthony
Title: Principal Research Scientist

Acknowledgments

I'd like to first thank my mom Lisbeth, my dad Jaime, and my sisters Aimee and Diana for their ongoing support throughout my academic endeavors and for always believing in me. Their motivation has pushed me to become a better engineer, and their confidence has made me work as hard as I have these past 6 years.

I'd also like to thank those that guided me through my research. My advisor Brian Anthony for his guidance in asking tougher questions and flexibility in choosing a research path. My mentor, lab-mate, and co-advisor Praneeth Numburi for answering all my questions and constantly helping with my academic, professional, and personal concerns.

Thank you to Micha, Shawn, and Talis for constantly sharing their expertise as I wrapped my mind around challenging technical concepts or tried new skills. Thanks to Katie for validating my concerns as a first-year student. And of course, thank you to Jess, Enya, Mariia, Lana, and all the other UROPs and peers that offered to help in the Immersion Lab when I needed extra hands.

Last but not least, thank you to the MIT MechE department for the enriching opportunities and experiences that helped make these past 6 years the most challenging and fulfilling of my life so far.

Enjoy the read!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis motivates and describes methods used to detect, characterize, image, and minimize physiologic tremors that exist in all healthy people. It first introduces the particular tremors of interest, and what distinguishes them from more commonly known tremors in skeletal muscle. It highlights the idea of task expertise in relation to these tremors as the central motivation behind the studies conducted for this thesis. It details the technologies used to capture physiological signals, and the methods used to extract, characterize, and analyze the tremors. Finally, it provides insights into the potential causes of these tremors and ways to mitigate them.

1.1 Physiologic Tremors

Tremors are defined as involuntary, rhythmic, and oscillatory movements of a body part [1, 2]. They can be categorized into resting and action tremors. Resting tremors are those that require low muscular effort and occur by relaxed body parts supported by gravity [1, 3, 4]. They can be enhanced under mental distress or by movement of a separate body part, and can be diminished by voluntary movement of the affected body part. On the other hand, active tremors occur with muscular voluntary contraction and can be sub-categorized into kinetic, isometric, and postural tremors. Kinetic tremors occur during any form of voluntary movement [5], and constitute cerebellar, dystonic, task-specific, or intention tremors. Isometric tremors occur specifically dur-

ing muscle contractions against a rigid or stationary object [1]. Postural tremors occur when body parts are maintained against gravity and include essential tremor (ET), the postural tremor of Parkinson's, drug-induced tremors, and physiologic tremors [6].

Physiologic tremors (PTs) exist in every healthy individual [6], and occur mostly in moments of excitement, anxiety, medication use, or other normal muscle activation [1]. This type of tremor has a typical frequency range of 8-12Hz that is often nearly invisible to the naked eye [7]. It can be exacerbated by stress, medication, withdrawal, etc, and its amplitude has been decreased by performing relaxation exercises, taking β-blockers, or ingesting small doses of propranolol when fine motor movement is required [8]. To quantify the severity of many tremors in a clinical setting, coarse and subjective, but digestible and practical, scaling tests are often used [9, 10]. However, physiologic tremors specifically are often disregarded as they do rarely disturb one's daily life and are not directly linked to any neurological disease. In fact, the root of PTs is still debated as there are many potential physiological parameters with limited measurability [11].

Three types of oscillations are thought to be the main contributors to the occurrence of these tremors: mechanical, simplifying the joint-tendon-muscle system to a mass-spring-damper system; reflex, extending brain signals down through the spinal cord and into the motor neurons; and central from the rhythmic patterns of brain electroencephalography (EEG) [12].

Most commonly, PTs are quantitatively assessed with various modalities such as electromyography (EMG) [13] and accelerometry [14, 15]. EMG sensors measure electrical activation of the muscle under the skin, and accelerometers measure 3-axis to 6-axis acceleration of a particular body part on which the sensor is placed [16], both having the advantage of being non-intrusive. These two types of signals will be used later in this thesis. The non-invasive nature of accelerometry and EMG are precise and accepted methods of understanding PTs on a surface level. To more deeply understand and potentially find the cause of these tremors, imaging the anatomy where the PTs arise is a natural first step. To do this, ultrasound imaging is used.

1.2 Ultrasound Imaging

1.2.1 Physics of Ultrasound

Ultrasound (US) is one of the most used, least expensive non-invasive forms of medical imaging available. US uses high-frequency sound to function. US transducers generate frequencies ranging from 2-55 MHz, 2 to 6 orders of magnitude higher than frequencies of audible sound [17]. These waves are produced by an array of crystals made from piezoelectric materials. When an electrical field encounters these material, they stretch and compress mechanically, creating a high-frequency vibration and producing the array of waves that form an US beam. This production of a high-frequency sound waves via electrical stimulation is known as the piezoelectric effect [18].

Pulses of sound beams are emitted from the US transducer at a set interval in a specific direction [17]. As the beams travel, their energy is either absorbed into tissue and dissipated as heat, scattered at interfaces, reflected back to the transducer, or refracted through changes in tissue density. These properties depend on the acoustic impedances of the tissues, speed of sound of tissue material, physical properties of the tissue surfaces, etc. Once the beams return to the transducer, their signals can be processed in various forms.

1.2.2 B-Mode Imaging

The beams that return to the transducer can be processed in as the pure scattered radio frequency (RF) signal tat contains all the information of acoustic waves and interactions with its medium. However, by far the most used form of US is brightness (B) mode, where transducer signals are converted into pixel brightness values [19].

B-mode imaging can give insight into the morphology of various tissues and features of the body; however, interpreting these images can heavily rely on the context of surrounding tissues [19, 20]. Tissues that vary in physical or mechanical properties will correspondingly reflect sound waves with various intensities, resulting in varying brightness in the B-mode image [21]. Hyperechoic structures will reflect nearly all

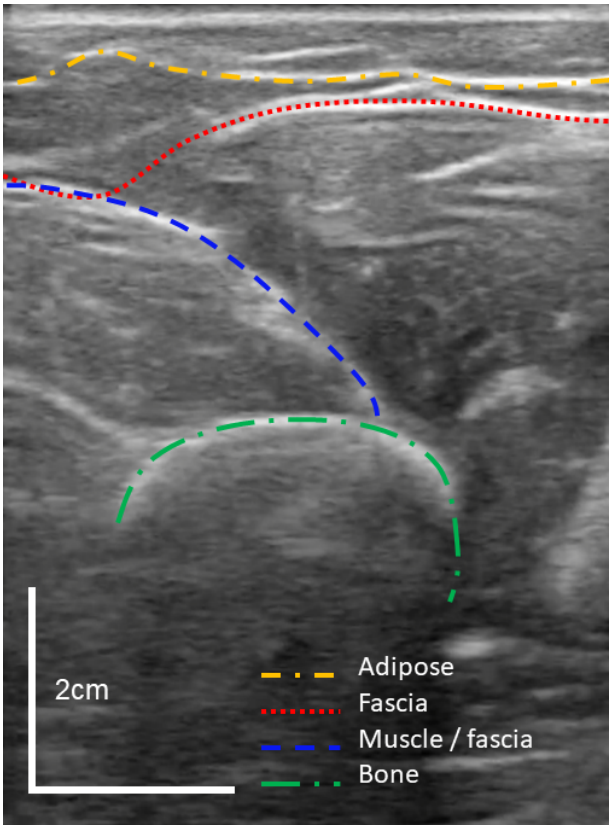


Figure 1-1: Transverse US image of upper arm. Each colored line represents interfaces between distinct features of the arm's morphology.

sound, resulting in bright regions in the image. Hypoechoic structures will partly reflect sound, resulting in gray-like regions. And anechoic structures will reflect nearly no sound, resulting in very dim regions. Scanning bone, for example, will result in a hyperechoic outline with an anechoic inside [21]. Muscles are hypoechoic with visible structure along the fiber lines. Fascia and other connective tissue appear as hyperechoic lines. An example transverse view of an upper arm and how various morphological structures reflect sound is shown in Figure 1-1.

Although one of the most common uses of US is imaging fetuses during pregnancy, medical sonographers can also be trained to perform abdominal, cardiac, maternity, cerebrovascular, breast, or tissue examinations, visualize blood flow [22], investigate muscle injuries [23], and identify cancer tumors [24]. US can also expand into a therapeutic setting by using its high-frequency sound waves to interact, modify, and even destroy tissues. This provides a non-invasive method of moving tissue, heating

tissue, dissolving blood clots, and even delivering drugs to specific locations in the body. In short, US can be a useful tool in a wide range of medical applications.

Most applications of US are meant to search for abnormalities while the patient is stationary, and often do not provide quantitative information about relative tissue deformations as a patient is moving. When using US as a method of studying human motion, B-mode scanning is typically done before, during, or after a particular movement. In prior work, US images of the medial gastrocnemius muscle were collected every 24 hours after delayed onset muscle soreness [25]. Changes in muscle area observed in 3 US images have been correlated to power output and glycogen levels in cyclists [26]. And US images of the biceps brachii muscle have been acquired every 5 seconds throughout a series of contraction exercises [27]. These studies show that US images can be captured with varying time intervals, time instances, and attachment styles to the region of interest. They also describe how imaging tissue with US B-mode can be used to observe fatigue states, detect anatomical changes, assess injuries, and analyze other features of interest. The diagnostic capabilities of US imaging can potentially be expanded by tracking features of tissues in motion.

1.2.3 Feature Tracking

When observing skeletal tissue using US during movement, there are various common features of interest known as muscle architecture parameters (MAPs) that are correlated with muscle contraction and motion [28, 29, 30, 31]. Features like the pennation angle and fascicle length can only be observed when imaging the skeletal muscle longitudinally, and features like bone rotation angle can only be observed when imaging the skeletal muscle in a transverse view [32]. Other MAPs include muscle thickness and cross-sectional area, and Figure 1-2 shows a transverse view of the upper arm that highlights a coarse way to estimate some of these features.

Successes in US feature tracking have been observed in prior work, although registration and feature tracking in US is still a challenging and unsolved problem [33]. Feature segmentation algorithms such as the Rayleigh distribution [34], Curvelet transforms [35, 36], and curve fitting [37] have been used to distinguish regions of

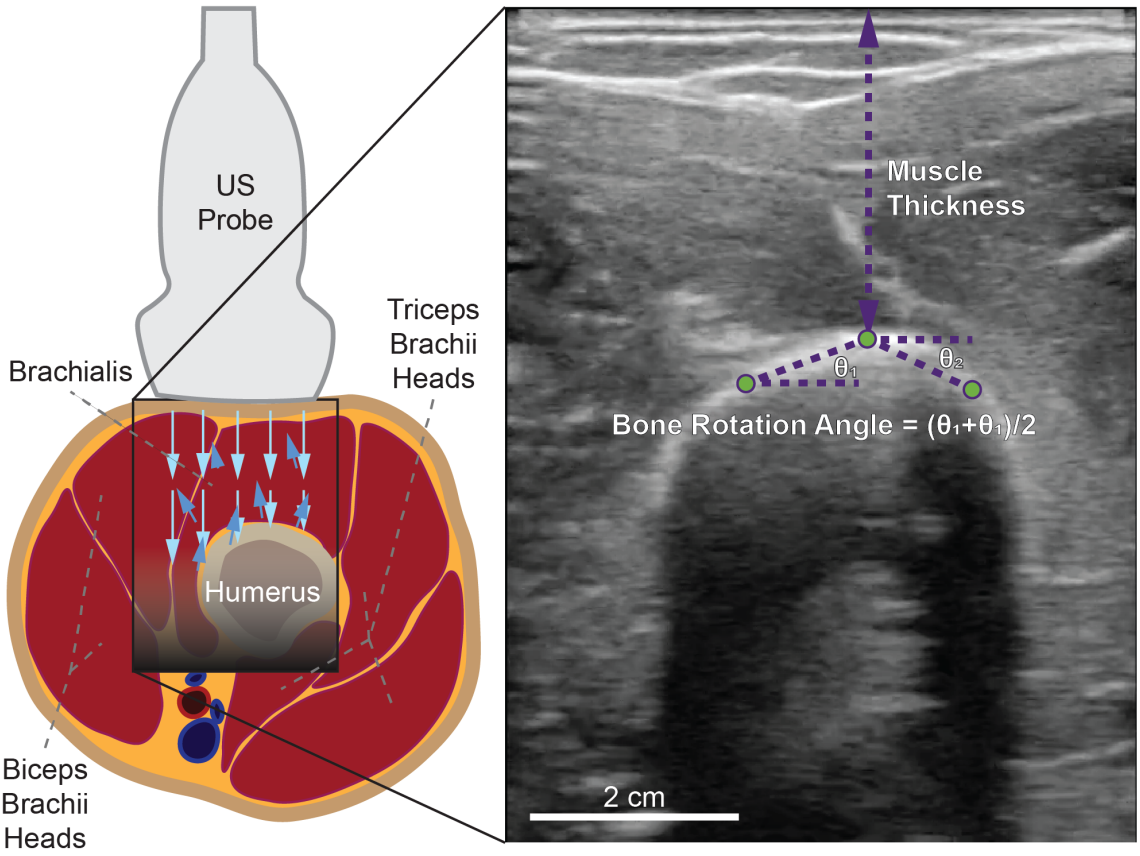


Figure 1-2: Ultrasound probe position on the upper arm for imaging the brachialis and the long head of the biceps brachii (left). A representative US image (right) showing how muscle architecture parameters such as muscle thickness, and bone rotation can be estimated from tracking specific points of interest in ultrasound videos.

interest within the probe's field of view. Speckle tracing has been found to be as accurate as MRI observations when measuring ventricular torsion [38]. Optical flow and block matching have been combined to find sub-millimeter accuracy when measuring finger tendon motion [39]. Features like pennation angle, thickness, and fascicle length of tissues have been extracted from lower-limb US video and used to estimate information about lower-limb motion [32, 40]. These methods of tracking features have all shown success, but have approached tracking on a case-by-case basis to develop an appropriate algorithm, a process that can be time-consuming, computationally expensive, and challenging to generalize.

The ability to track increasingly smaller regions can be helpful in generalizing

tracking for all kinds of features. Region and point-tracking algorithms like optical flow [41, 42], block matching [43, 44], and template tracking [45] have been used for the identification and tracking of various tissues, tendons, and bones. Optical flow uses pixel intensity (brightness) between two consequent frames to determine pixel velocity and displacements [39] which can be helpful to track slow-moving features in any video. Block matching and template tracking can estimate the movement of a region of interest from one frame to another by maximizing the similarity between all regions in the next frame [39].

These methods often encounter various challenges due to the quality of US imaging. Optical flow, for example, can fail with sudden movements in the video, especially due to high speckle noise and fluctuating image brightness. The issue of despeckling in particular has been addressed using the Rayleigh Mixture Model [46], Daubechies complex wavelet transform [47], ripplet domain nonlinear filtering [48], and Laplacian pyramid-based nonlinear diffusion [49], which have been shown to improve quality of US videos and tracking. Template tracking can encounter similar issues due to a short depth of view or poor image quality due to the varying speed of sound characteristics of the imaged tissues. Additionally, depending on the orientation of the probe and the body movement being observed, any point in the region of interest can inconsistently appear and disappear frame-to-frame.

Many of these challenges were encountered during data collection as skeletal tissue features were tracked. This thesis will outline the various methods used to tackle some of these issues in order to track features reliably in US footage. The goal of these approaches is to provide an approach for consistent tracking to be used both in the analysis of physiologic tremors and hopefully generalized to other applications.

1.3 Study Motivation

Months ago, the physiology of factory sewing machine operators was studied, hoping to understand how the repeated movements played a role in work conditions and their overall health. When wrist accelerometer data was collected, PTs with characteristic

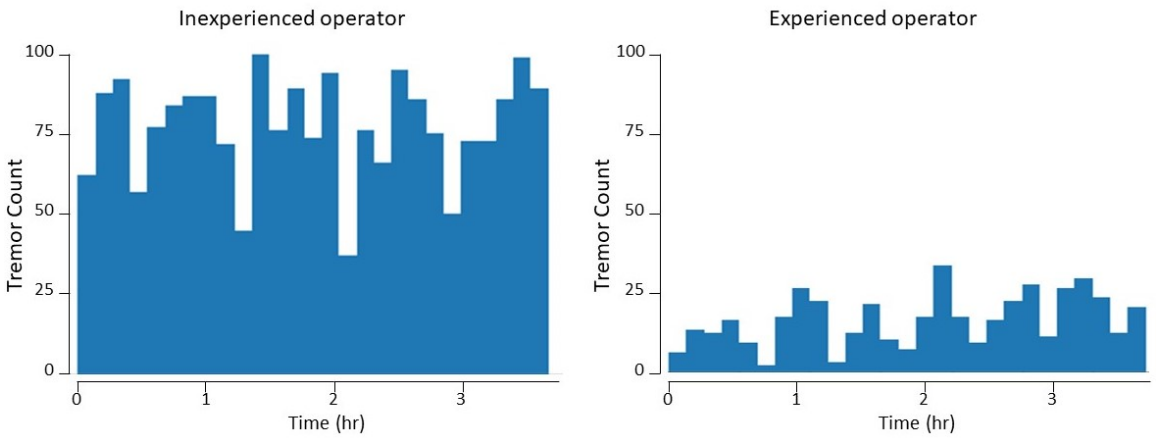


Figure 1-3: Significant difference between tremors seen in experienced vs inexperienced operators during identical tasks.

frequencies between 10-20 Hz were measured and counted. One particular machine operator with decades of experience underwent less than half of the number of tremors when compared to her less experienced counterparts, as shown in Figure 1-3, bringing a few points into question.

First: what, if any, confounding variables played a role in these initial results, and how can a new experiment be designed to mitigate them? Initial inspection of the videos hinted that signals considered tremors were sometimes only moments of impact on the wrist, walking around the factory floor, or other miscellaneous tasks not related to sewing. Second: once a potentially more controlled set of data is collected, can higher task expertise be quantitatively linked to the occurrence of these PTs? The correlation between slow-movement task expertise and tremors in this frequency range has yet to be analyzed, so this is a major question of interest for this thesis. And third: if no correlation is found, do alternative movement methods exist that can still mitigate tremors? Answering these questions can present strategies to the many professions and fields of expertise that rely on smooth movement for success. Next, a brief review of how task expertise has been studied will be presented.

1.3.1 Task Expertise

Expertise has been studied for thousands of years in art, science, sports, etc. [50]. Various factors are commonly correlated with improved skill levels. The role of deliberate practice on expertise explained first in 1993 by Ericsson et al. has been mentioned over 10,000 times since its publication [51]. An alternative model of expertise was proposed taking into account the effect of one's genetics (including innate abilities and characteristics, personality, interests) and the model's interaction with one's environment as it morph one's neural mechanisms, physical characteristics, and practice habits [52]. Improved accuracy and speed in motor tasks has also been observed, explaining factors like motivation and effort when performing said task, pre-existing knowledge, immediate informative feedback, and repeated activity can all affect performance [50]. In all of these cases, indicators of expertise are quantifiable measures of success within the task itself (number of wins in sports, number of awards in film), or qualitative measures of progress (quality of feedback, inspiration).

Aside from the mechanisms underlying expertise, novices and experts have also been analyzed when performing their occupation. In studying authors' and scientists' accomplishments, it was found that more than 10 years passed between their first work and their best work in their field [50], suggesting tenure in a specific task is correlated with skill. On the side of sports, skilled dancers used more vertical posture and an anterior core strategy to perform one-legged balancing tasks [53], exemplifying that subtle changes in physiological states can help differentiate expertise level. On the side of medicine, expert surgeons were found to have smoother movement continuity and more stable dexterity when performing instrument manipulation movements than their beginner counterparts [54]. In short, there are inherent and often subtle differences in how movements are performed by experts and novices.

When studying slow and controlled movements specifically, even more subtle differences between experts and novices emerge. Surgeons were found to experience more tremors in their hands over time as they performed a micromanipulation task holding tweezers [55], potentially due to fatigue. Adults were separated into experts

and novices in their respective occupation, and the expert group was found to have more precise, less variable, and more complex hand movement performance [56]. Pre-elite and elite air pistol shooters were compared, and it was found that more skilled shooters experienced significantly fewer tremors [57] when holding the gun statically.

There appears to be an inherent correlation between the level of an individual's expertise level in a specific task and the smoothness of their movement when performing it. Specific types of tremors were observed in the case of the pistol shooters, surgeons, and factory operators, but the frequency ranges and amplitudes explored varied in each scenario.

For this thesis, an experiment is run during which slow controlled movements are performed by subjects with various skill levels in different movement fields. The frequency ranges and amplitudes will be detailed, and the methods of signal processing will be described. As mentioned in Section 1.3, the slow controlled movements are meant to understand PTs in a more controlled setting than that of the operators, while analyzing external and internal physiological changes.

Chapter 2

Data Collection

28 healthy subjects (14 females, 14 males, mean age 27.2 ± 7.7) participated in a motion study held at the Immersion Lab in MIT.nano. All subjects provided written informed consent according to MIT's Committee on the Use of Humans as Experimental Subjects (COUHES). During the experiment, each subject performed a sequence of movements meant to invoke various physiological and morphological changes. Table 2.1 below shows the movement and description of each trial:

The experiment was run to better understand the relationships between muscle activation, muscle length, muscle thickness, joint angle, joint angular velocity, and heart rate on the occurrence and magnitude of PTs. To observe these factors, a set of three main sensing modalities were used.

2.1 Sensing Modalities

2.1.1 Optitrack Motion Capture Markers

To prepare the subjects, 30 to 33 9.5mm motion-tracking markers from Optitrack (Optitrack, Corvallis, Oregon, USA) were attached to adhesive velcro around the body (spine, shoulders, arms, chest, and feet), the exact quantity depending on the subject's height and shoulder width. 20 Optitrack Prime 13 cameras captured 3D location data with a sampling frequency of 240 Hz and an accuracy of ±1mm.

Trial # :	**Description:**
1	Walking: 3 trials walking on a treadmill for 1 minute each
2	MVC: 1 trial performing brief, full-arm maximum voluntary contractions (MVCs)
3	Rest: 1 trial resting with eyes closed
4	Exertion: 1 trial planking for 1.25 minutes followed by a resting period
5	Reaching (Group 1 and 2): 3 trials extending and retracting the US arm. 1 of the 3 was performed with a pronated palm and extending forward, 1 of the 3 was performed with a supinated palm and extending forward, and 1 was performed alternating palm orientations and extending to randomized directions around the body
6	Reaching (Group 2 only): 10+ trials extending and retracting US arm. Each trial introduced an intervention hypothesized to vary PTs in some way.
7	Arm Rotation Movements (Group 1 only): 16 trials performing 2 consecutive types of arm movements. 1 type of arm movement consisted of elbow flexions and elbow extensions starting from a specific configuration of shoulder, elbow, and wrist rotations, and the other consisted of extending and retracting one's arm to randomized directions, sustaining the rotations.

Table 2.1: Experiment trials and descriptions

2.1.2 Delsys Physiological Sensors

Next, Delsys (Delsys, Natick, Massachusetts, USA) sensors were adhesively attached to the skin. On each arm, one Trigno Avanti Sensor (TAS) was placed on the long head of the biceps brachii muscle, one Trigno Snap-Lead Sensor (TSS) was placed on the medial head of the triceps brachii muscle, and one TAS was placed on the flexor pollicis brevis muscle on the palm. An additional TAS was placed on the lower part of the pectoralis major muscle on the chest. All 7 of these sensors captured acceleration data at 240 Hz with a capture range of $\pm$4G, and EMG data at 1920 Hz. The last sensor on the lower chest also obtained electrocardiographic (EKG) data at 1920 Hz to monitor heart-rate.

2.1.3 Cephasonics Ultrasound Probe

A 64-channel Cephasonics Linear Ultrasound probe (40 MHz) (Cephasonics Ultrasound Solutions, San Jose, California, USA) was attached to the brachialis of a randomly selected arm in order to capture a transverse view of its internal morphology - this arm will be referred to as the "US arm." A linear probe with sufficient gel was used to ensure clear and continuous contact. The probe captured 656x496 pixel B-mode images at 60 fps with a depth of 80.7mm, so 1 pixel was equivalent to 0.123mm. It Figure 2-1 illustrates the setup and experiment.

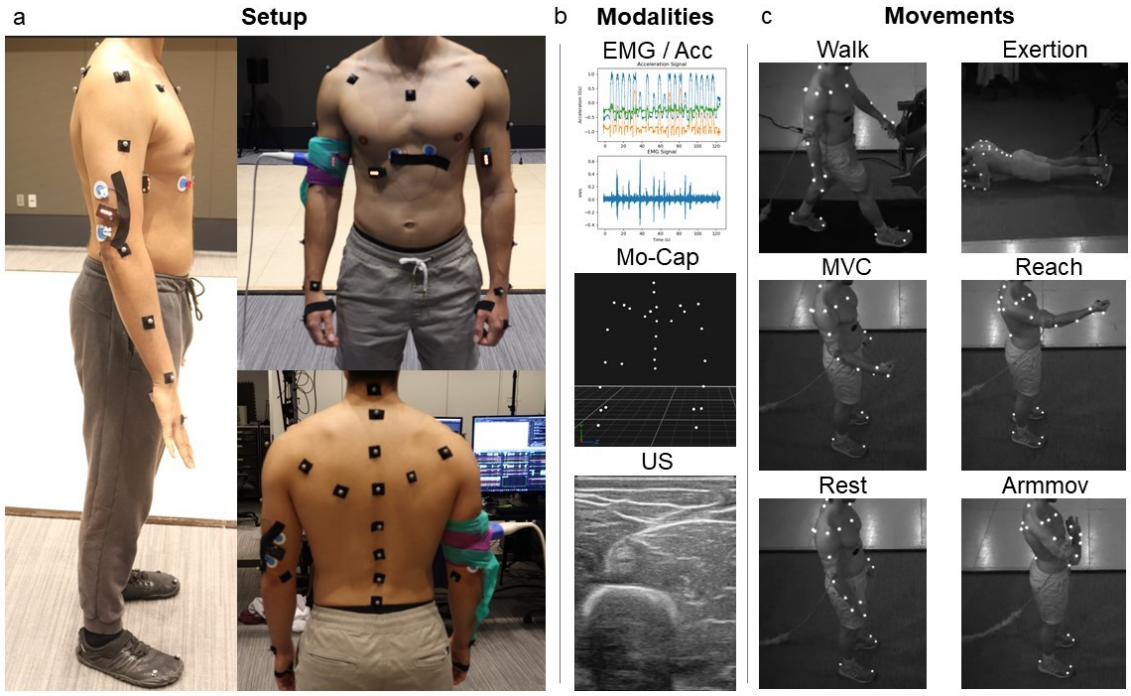


Figure 2-1: (a) Experimental setup consisting of Optitrack motion-tracking markers, Delsys physiological sensors, and a Cephasonics ultrasound probe. (b) Examples of signals from all 3 sensing modalities. (c) Clips of subject performing the movements from each type of trial.

Chapter 3

Tremor Characterization

3.1 Tremor Extraction

As mentioned in Section 1.1, PTs are found to range between 8-12Hz. During data collection, these tremors became visibly evident within the x-y-z components of the accelerometer signals, so various methods were used to accurately extract them.

3.1.1 Fast-Fourier Transform

To ensure the signals observed were described by tremors in PT literature, a full-windowed discrete Fast-Fourier-Transform (FFT) was applied to each trial. This was done to visualize the frequency spectrum of each trial. A discrete FFT is an efficient version of the Fourier Transform algorithm that transforms any discrete signal from the time domain to the frequency domain. In this case, the SciPy library's fast fourier transform method (scipy.fft.fft()) was used, a function that applies Eq (3.1) to transform the magnitude signal from the sampled time domain $x[n]$ to the frequency domain $y[n]$.

$$y[k] = \Sigma_{n=0}^{N-1} e^{-2\pi j \frac{kn}{N}} x[n] \tag{3.1}$$

This function effectively splits the magnitude signal x into N sinusoids with increasing frequencies k, and results in

$$y[k] = A_k^2/4 \tag{3.2}$$

where A_k is the magnitude of the sinusoid of frequency k. Taking the Fourier Transform of a raw signal results in a two-sided power spectrum symmetrical about the DC frequency ($k = 0$), since the complex exponential from Eq (3.1) can be computed with positive and negative frequency values. The negative frequency values were therefore discarded, and the positive value magnitudes were doubled. This condensed signal was then converted into logarithmic units because noisy signals, like those captured during this experiment, can have an unproportionally high amplitude at low frequencies. Finally, because the power spectrum from the FFT was still noisy, a moving average filter of length 240 (equivalent to 1 second at the accelerometer sampling rate) was applied for signal smoothing. This process is fully highlighted in Figure 3-1.

The main takeaway from these steps is the clear magnitude peak around 11Hz, implying at a very general level that there is repeated motion at this frequency in most of the trials. To further understand the temporal dynamics of each trial, more comprehensive frequency analysis needs to be done.

3.1.2 Welch's Method

The smoothed FFT signals seen in Figure 3-1 highlight the frequency characteristics of entire trials, often lasting between 1 to 3 minutes. From viewing a trial's accelerometer data, it was clear that the tremors of interest often occured in short bursts of around 1-2 seconds, especially in trials where reaching is performed. To better understand and visualize the frequency characteristics temporally, Welch's Method was applied.

The idea behind Welch's method can be summarized as follows. Before taking the FFT of an entire time signal x, the signal is first broken up into segments of a window size w. These segments can optionally overlap by some number of samples p. Each segment is tapered using a Hanning taper because the discontinuous jumps at the segment's ends require high frequency sinusoids to estimate it, creating edge effects

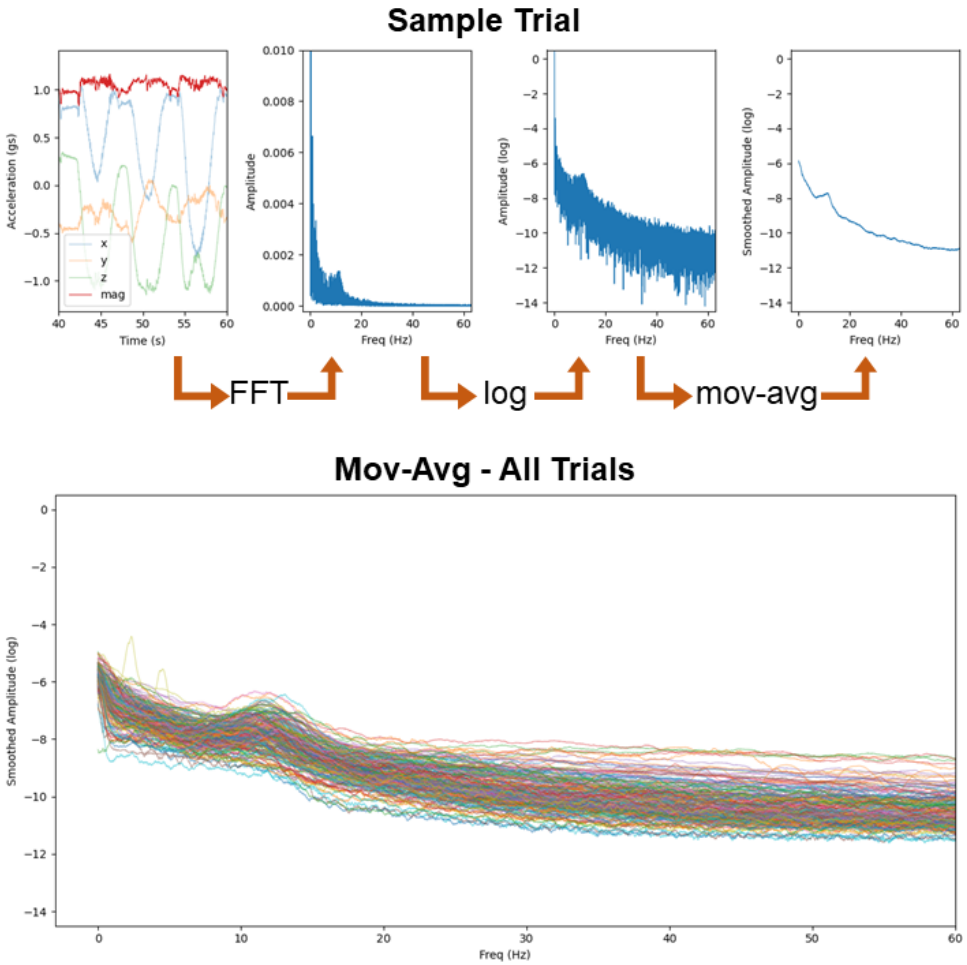


Figure 3-1: (a) Process of extracting the frequency information from a signal by first applying the SciPy library's fft method, eliminating the negative-frequency components, changing its power spectrum to a logarithmic scale, and applying a moving average filter. (b) shows a coarse view of the smoothed power spectrum for all trials in this experiment.

that distort the desired power spectrum. After tapering, their FFTs are calculated and transformed into logarithmic units, resulting in a power spectrum for each time segment. Figure 3-2 illustrates this process.

Once all power spectrums are obtained, Welch's method typically averages them together to obtain a smoother power spectrum than when taking a full-windowed FFT, but for this application, the individual power spectrums were plotted as a function of time, resulting in a 3D representation of frequency distributions throughout the course of a trial. An example trial and frequency distribution is shown in Figure

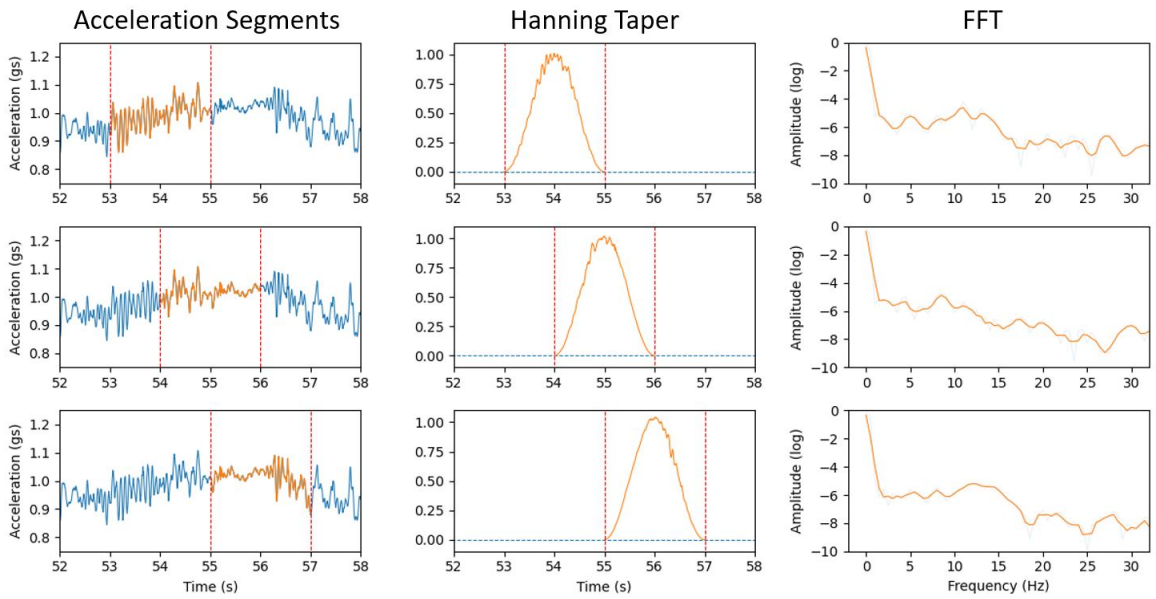


Figure 3-2: (a) Welch's Method visualized. A 2-second acceleration signal ($w = 2f_s$) is multiplied by a Hanning taper of the same length, and a power spectrum of that tapered signal is obtained via FFT.

3-3.

As shown, the time intervals when an oscillation is visually present in the acceleration signal corresponds to darker shades around the frequency of interest, distinguishing that the motion occurring at these frequencies matches that of the tremor of interest - 8 to 12Hz. To achieve a less discretized time distribution, a smaller window size w needs to be chosen smaller; however, using a smaller w also means including fewer samples for the FFT to use, resulting in a more noisy power spectrum in each time segment. A bandpass filtering method is therefore used to obtain less discredtized intervals when the tremors occur.

3.1.3 Bandpass Filter

The goal of a bandpass filter is to eliminate the contributions to a signal from outside a frequency band. In this application, it is to eliminate the contributions from outside the stated 8-12Hz range and visualize the remaining signal. To do so, the SciPy library's firwin method is used (scipy.signal.firwin()) to compute the coefficients of the desired Finite Impulse Response (FIR) filter. Because the signal's sampling rate

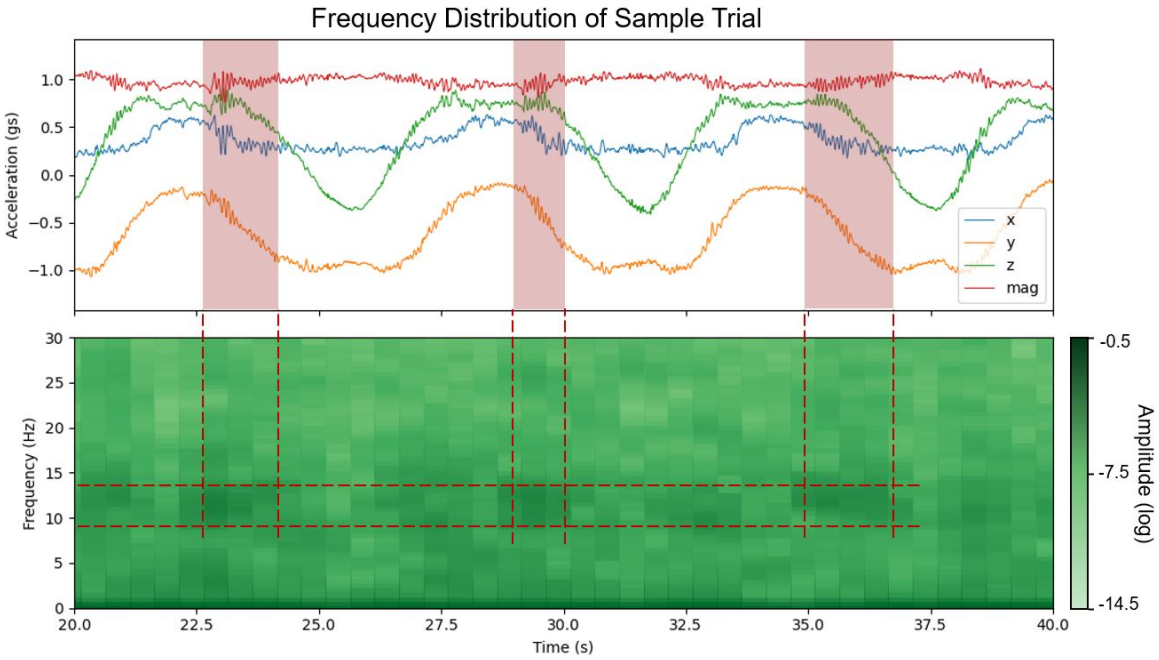


Figure 3-3: The top shows an acceleration signal and its x-y-z components with potential tremors highlighted in red. The bottom plot shows the corresponding power spectrums at every time segment, where darker shades correspond to higher amplitudes in the power spectrum.

is 240Hz, the length of the filter was 121, or 1 more than the Nyquist frequency.

Once the fiter's coefficients are calculated, the SciPy library's filtfilt method is used (scipy.signal.filtfilt()) to apply the filter to the signal forwards and backwards, resulting in the filtered signal of interest.

The next step is to determine a threshold value, or how high the filtered signal must reach to be considered a tremor that is visually apparent in the original signal.

3.1.4 Thresholding

When various researchers observed and pointed out signals they believed to be tremors, it was clear that the 8-12Hz bandpass filter range was insufficient to encompass all tremors of interest. To determine the filter parameters more objectively, various researchers were shown a set of snippets from various trials and were asked to specify the signal ranges they considered to be tremors, if at all. Then, the signals from various trials were filtered using a set of 5 parameters. Aside from low and high

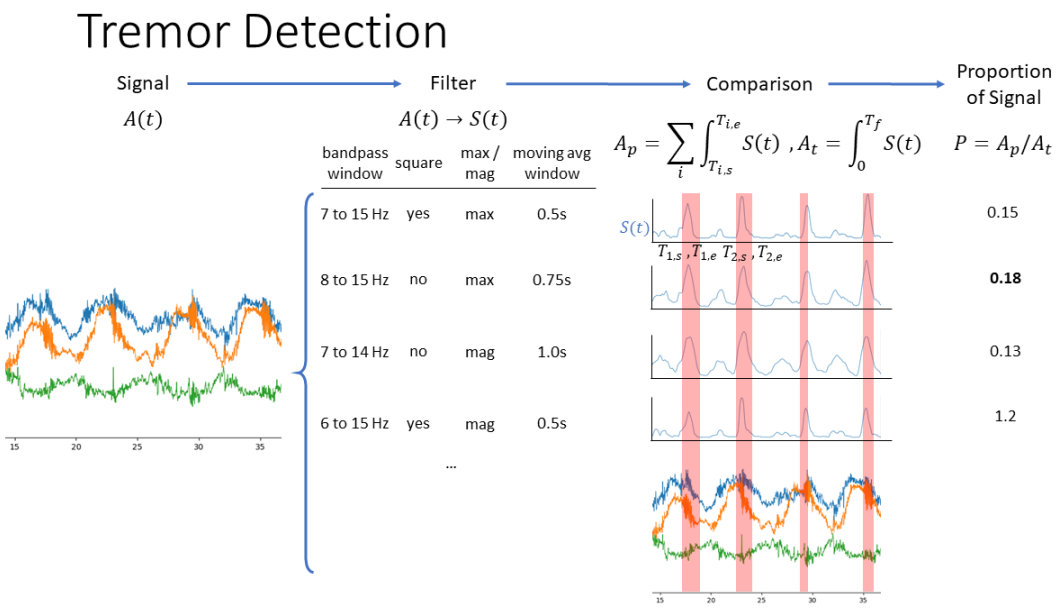


Figure 3-4: Full parameter-optimization algorithm highlighting signal filtering, correlating labeled time snippets to filtered signal peaks, and finding parameters with the highest peak proportion.

bandpass frequencies l and h, a moving average window of size w could be applied to smoothen the noisy bandpassed signal. Second, the signal could be magnified (raised to an exponent e) to amplify higher peaks. And third, instead of filtering the magnitude signal from all 3 acceleration axes, the filter could instead be applied post-bandpass filter to the axis with the most apparent tremor, since the tremor from the arm muscles were more prominent in the component with the maximum value after bandpassing. The algorithm illustrated in Figure 3-4 summarizes these steps and highlights how the optimal filter parameters were determined.

After researchers selected the signal snippets they considered to be tremors, those signals $A(t)$ were filtered into $S(t)$. To determine how efficient each filter was at highlighting the same features labeled by the researchers, the ratio of the area under the labeled regions (A_p) to the total area under the signal (A_t) was found. The filter with the highest ratio was chosen as the optimal filter for that signal, the process was repeated for all signals, and the most common parameters were found, resulting in the following optimal filter:

$$S(t) = smooth(m(A(t)).bandpass(l,h)^e, w) \tag{3.3}$$

where $l = 7$Hz, $h = 14$Hz, $e = 2$, $m = \text{np.max}(A(t)\text{,axis=1})$, $w = 0.5$s

The last parameter needed for analysis was the threshold value $S(t)$ should surpass to be considered a tremor. The threshold that best highlighted peaks in $S(t)$ during the time snippets previously labeled was manually found to be $2.25\text{x}10^{-4}(\frac{m}{s^2})^2$. This completes the tremor extraction methodology. Next, analysis of all signals was conducted with the goal of extracting potential causes of PTs or relationships between them and other physiological signals.

3.2 Tremor Metric

All "Reach" trials were filtered using the tremor extraction method from above, outputting a plot like that shown in Figure 3-5 highlighting tremors in red. Because the interest in this study was to understand PTs during movement, sections of the signal when movement was happening (acceleration derivative above a chosen threshold) were highlighted green. The study motivating this thesis in Section 1.3 used the quantity of tremors during a sewing session as the metric of interest; however, the filtered signals illustrated that this metric did not accurately describe the magnitude or duration of each tremor. To better highlight these details, the metric chosen for analysis instead was the percentage of moving signal (highlighted in green) that is also above the tremor threshold (highlighted in red), abbreviated as TPM (Tremor Proportion in Movement) for the rest of this thesis.

TPM provided a more granular understanding of tremor characteristics, and allowed for more exact comparisons between accelerometry, EMG, and ultrasound.

3.3 TPM in Motion Capture

Reaching tasks can be broken down into two categories: extension and retraction. The most immediate observation during data collection was that the tremors tended

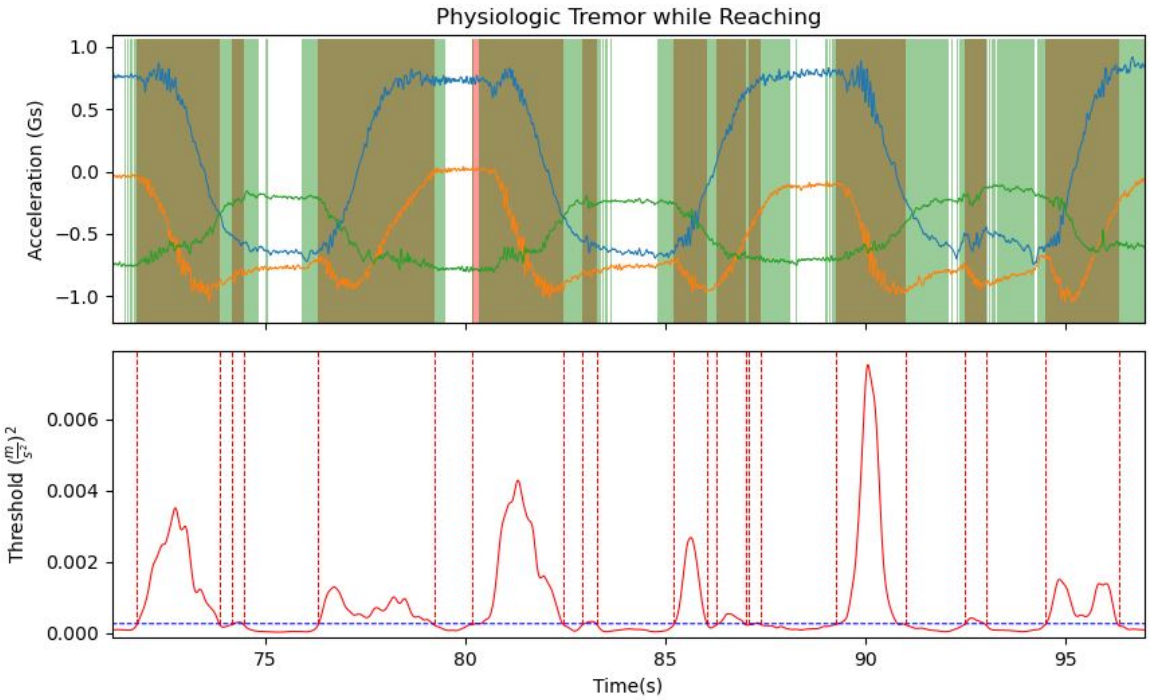
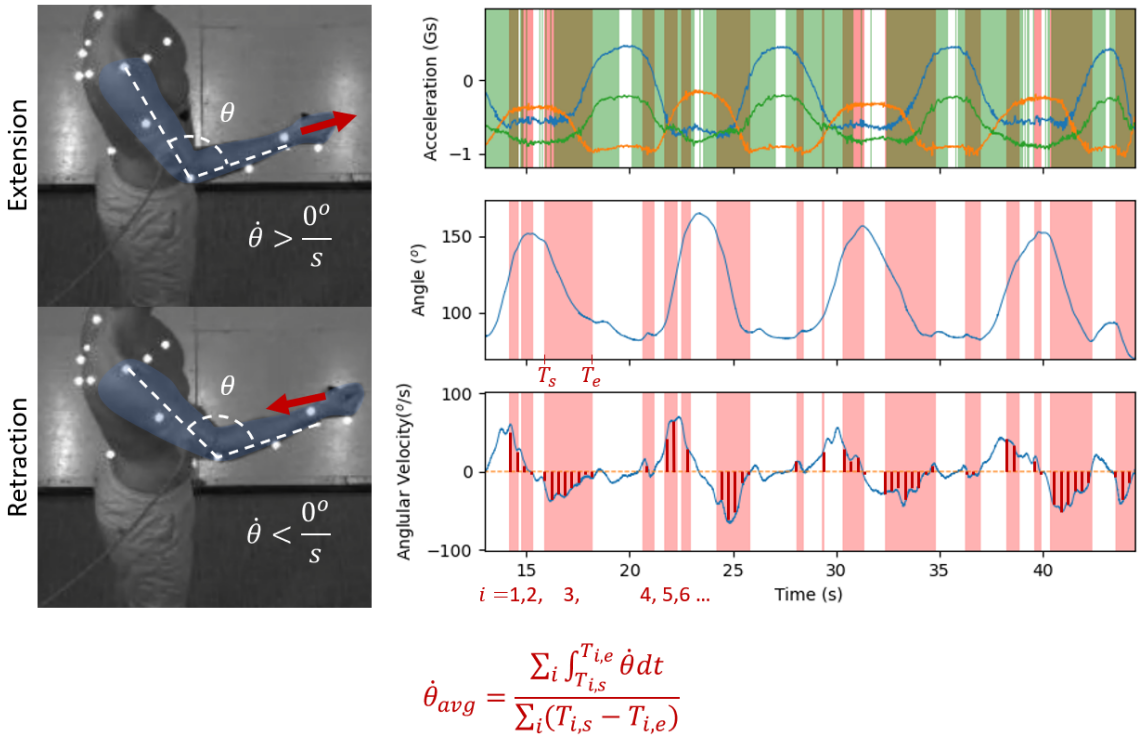


Figure 3-5: Example signal filtered. The top plot highlights the signal sections considered movement in green, and the signal sections considered tremors in red, determined by when the signal in the bottom plot rises above the threshold value.

to occur most often during retraction. To quantify this, motion capture data from the shoulder, lateral elbow, and lateral wrist was taken, and the relative elbow angle was estimated. Extension is characterized as the hand stretching forward and the elbow angle increasing, while retraction is characterized as the hand pulling towards the body and the elbow angle decreasing. Therefore, the angular velocity was calculated. For each subject, the average angular velocity while above the tremor-threshold was found, a process shown in Figure 3-6.

This method was used to determine whether tremors occurred more often during extension or retraction, as well as determine whether this pattern was true for tremors found in all 3 muscles.



$$\dot{\theta}_{avg} = \frac{\sum_i \int_{T_{i,s}}^{T_{i,e}} \dot{\theta} dt}{\sum_i (T_{i,s} - T_{i,e})}$$

Figure 3-6: Method for extracting average angular velocity of the elbow angle in a trial.

3.4 TPM in Ultrasound

The last piece of accelerometer analysis done for this thesis involved exploring potential relationships between the filtered acceleration signal for tremors and the underlying tissue seen in ultrasound (US). The details of will be described in the next chapter in more detail, but in summary, the goal of US analysis was to establish a reliable method of tracking individual points in an US video. Once this method was established, the points' estimated coordinates over time could be filtered just as the acceleration signal was in Section 3.1.4. Although the sampling frequency for US image capture (60 Hz) was much lower than that of the acceleration signal (240 Hz), was still met the Nyquist criteria when compared to the tremor frequency of interest (7-14 Hz).

To compare them, the filtered coordinate signals were normalized to have mean of 1, as was the filtered acceleration signal. Then, the root mean squared error (RMSE) between the two was calculated and stored based on both the coordinate (x vs y) and

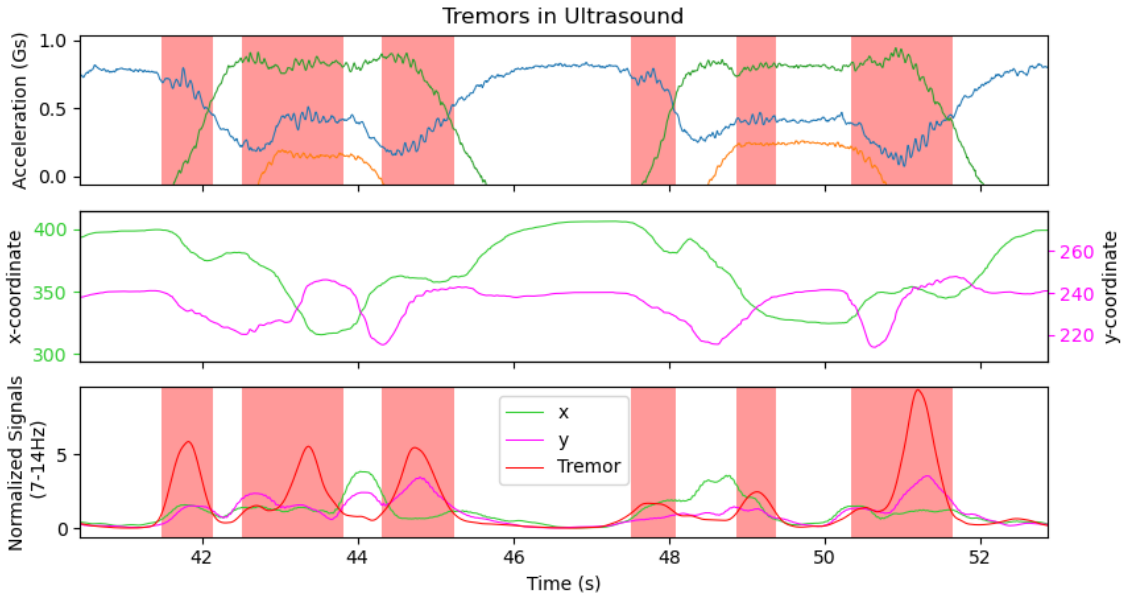


Figure 3-7: Method for comparing tremor observed in US and in accelerometry. The upper plot shows the 3-axis accelerometry signal. The middle plot shows the x and y coordinate pixel values of each the tracked point. The lower plot shows the 3 filtered signals (acceleration, and x and y of the pixel); the RMSE between these 3 signals is calcalated to determine the similarity between each of these coordinates.

the type of tissue tracked. Figure 3-7 illustrates this process.

This established a metric of comparison between PTs and movement seen in US. The methods for ultrasound video and image analysis will be described next.

Chapter 4

Ultrasound Analysis

In this section, the approaches to both track distinct points in ultrasound (US) videos and compare these tracking methods will be described.

4.1 Image Preprocessing

Each of the tracking algorithms chosen had varying performances depending on the characteristics of the feature being tracked. It was therefore necessary to choose various features to label to compare the performance of each tracker. Each upper arm anatomy varied, from the arm chosen to the thickness of the brachialis to the shape of the humerus, so different points were selected to be labeled for each subject, highlighted in a randomly selected frame. These labeled frames were used as a templates to highlight specific points in bone, tissue, fascia, etc., ensuring the points and features around it did not fade off-view in any video. Then, one randomly selected video from each type of movement was chosen (walking, MVC, rest, and reaching) to be labeled using the template for that subject, labeling eight frames equally spaced temporally. This process is illustrated in Figure 4-1.

With this process, each subject had a total of 48 labeled frames with 3 labeled points, or 144 total labeled points used to assess each algorithm's tracking accuracy.

The ultimate goal of this process was to find the most accurate tracking algorithm. Having each subject's template highlight different features prevented tracking to be

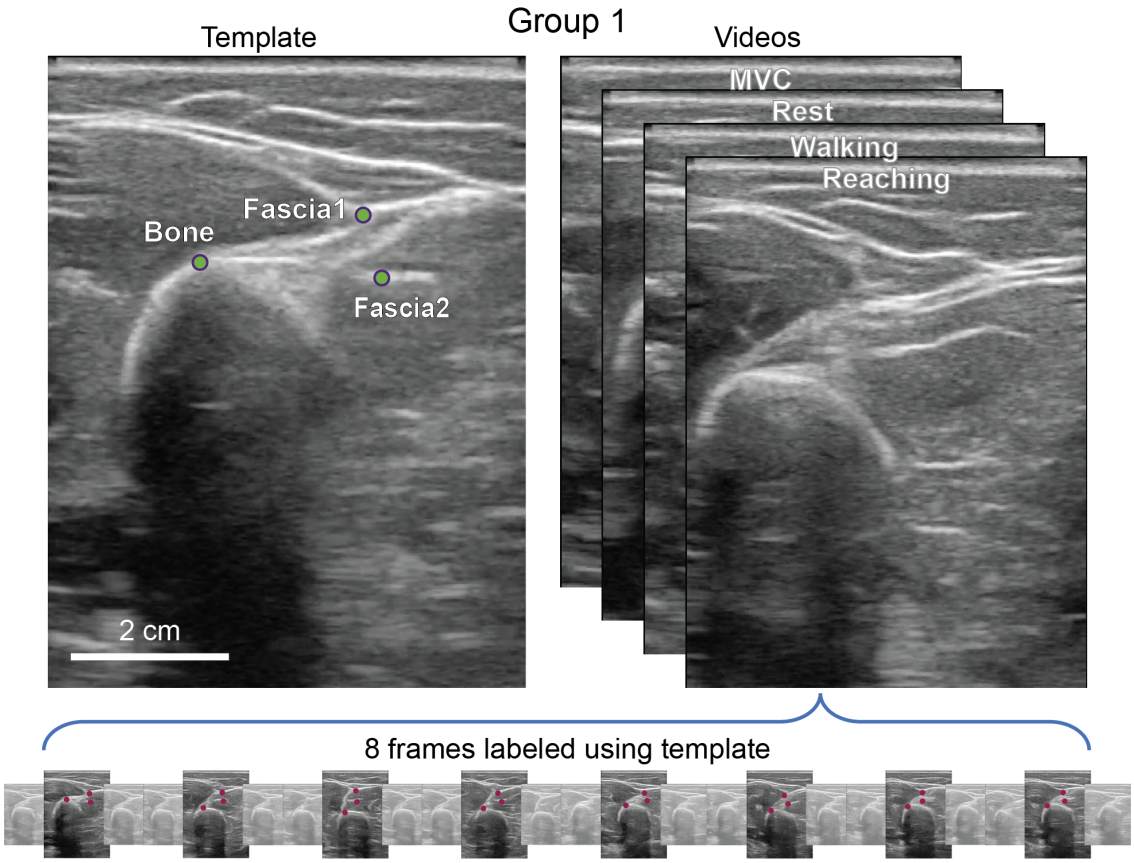


Figure 4-1: The first video labeling process. The tracking template for one subject is shown at the top with green labeled points. For each of the six randomly selected videos, eight frames are manually labeled with the same morphological features as the template, shown at the bottom frames with red labeled points.

generalized for this purpose. Therefore, a second group of videos was collected from a smaller batch of subjects, and three points along the edge of the bone were labeled as shown in Figure 4-2. For this group, each subject had between 13 and 20 US videos. 10 frames that were 0.33s (20 frames) apart were labeled with 3 points. Choosing a shorter gap between all labeled frames of this batch of subjects allowed for a more precise manual labeling process, especially because only small and clear movements would occur from one labeled frame to another.

The more points that were labeled along the edge of the bone or other tissues, the more accurate these feature approximations could be.

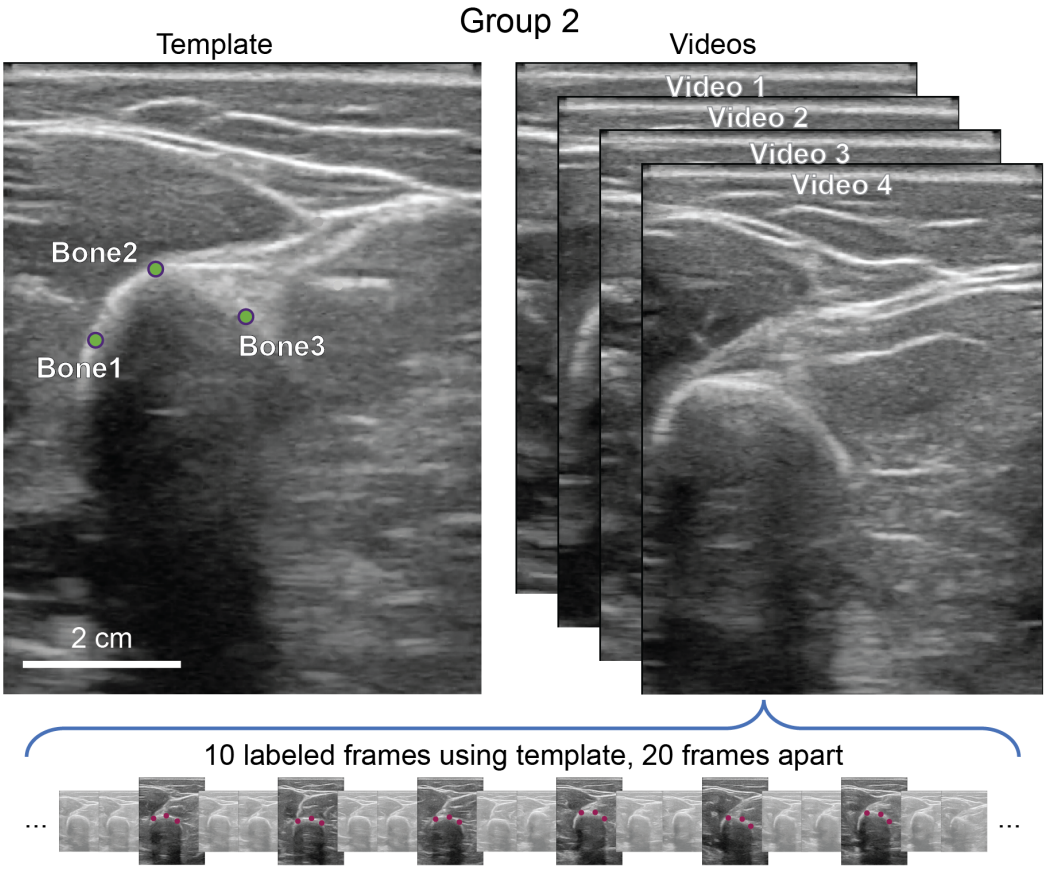


Figure 4-2: The second video labeling process. The left frame shows an example template for the second video labeling process. The right two frames show approximate methods for obtaining relative bone rotations and muscle thickness.

4.2 Tracking Methods - Optical Flow & OpenCV

The first set of algorithms used is the newest set of object-based, frame-to-frame trackers within the OpenCV library. These algorithms have been analyzed extensively on various types of videos [58, 59, 60], most of which analyze the tracking of pedestrians in a crowd, vehicles, or faces. Although many image-processing algorithms have been developed to tackle specific tracking challenges in US video (liver tracking [61], surgical instruments [62]), little work has been conducted comparing the efficacy of OpenCV algorithms. Imaging the upper limbs transversally can specifically display detailed cross-sections muscle thicknesses, bone rotation, and fascia distributions, but it has yet to be thoroughly analyzed using OpenCV.

At this moment, OpenCV has incorporated 8 major frame-to-frame trackers into

their libraries including BOOSTING [63], MIL (Multiple Instance Learning) [64], MedianFlow [65], MOSSE (Minimum Output Sum of Squared Error) [66], TLD (Tracking Learning Detection) [67], KCF (Kernelized Correlations Filter) [68], GOTURN (Generic Object Tracking Using Regression Network) [69], and CSRT (Channel and Spatial Reliability Tracker) [70]. Analysis of only the CSRT, KCF, and MOSSE trackers was done due to promising prior results from non-US video tracking [58] and initial qualitative analysis of these algorithms.

Most tracking issues encountered were due to feature obstruction, fast feature movement between frames, feature movement away from the field of view, and tracker drifting. Viewing the arm transversally made it possible for some similar issues to be encountered since tissue features could travel out of the plane; however, various aspects of tracking in this US application actually lessened these burdens. First of all, tissues always remained close and in the same general orientation relative to each other. Second, features that exit the plane of view during one movement will return to the plane during the reversal of that movement. And third, once the US probe was attached to a specific location, the average size of morphological features did not vary greatly. These benefits when tracking US videos are not often found in the other common pedestrian or vehicle applications.

In order to begin tracking, all OpenCV algorithms required a bounding box that acts like a template for some of these trackers to capture frame-to-frame movement of image features. A 100x100 pixel square was used as the initial bounding box size, but smaller square sizes were also used for comparing tracker accuracy and speed. After each update to the bounding box, the center point of the new box was found, and the new box's size was reset to its initial dimensions to avoid the potential distorting of box dimensions over time.

4.3 Tracking methods - Lucas-Kanade

The Lucas-Kanade (LK) method [71] has been one of the most widely-used computer vision techniques for tracking, motion estimation, face decoding, etc. [72] for the last

30 years. It uses some template image $T(x)$, usually a small subsection of a video frame, and aligns it to an image of interest $I(x)$, usually the video's next frame. A parameterized warp matrix $W(x;p)$ takes some pixel of the template $T(x)$ and maps it to the coordinate frame in the image I. This matrix allows the algorithm to check how various features are not only moving, but also distorting throughout the video. The algorithm is described in [72], but a brief summary is presented here.

The goal of the algorithm was originally to find where the template image likely moved to in the next frame, regardless of how the template might have distorted between frames. Mathematically, this refers to minimizing the sum of squared errors between the template $T(x)$ and the image I warped back onto the template's coordinate frame:

$$min\Sigma_x[I(W(x;p)) - T(x)]^2 \tag{4.1}$$

Some Δp is computed using the partial derivatives of the gradient of I evaluated at $W(x;p)$. This value is updated until the norm of the vector is below some threshold.

$$\text{while } ||\Delta p|| < \epsilon$$
$$\text{compute } \Delta p$$
$$p \stackrel{+}{=} \Delta p$$

The OpenCV adaptation of the algorithm simplifies this method by simply minimizing the sum of squared errors between a template $T(x)$ and the next image in a video I, to avoid considering warping effects. Under the assumption that the pixels in the template continue to have the same size, shape, and intensity, one of these pixel's change as follows:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{4.2}$$

which can be simplified via Taylor-series expansion to:

$$\frac{\partial f}{\partial x}\frac{\partial f}{\partial x} + \frac{\partial f}{\partial x}\frac{\partial f}{\partial x} + \frac{\partial f}{\partial t} = 0 \tag{4.3}$$

$$f_x u + f_y v + f_t = 0 \tag{4.4}$$

Expanding this to all nxn pixels in the template, the problem expands to solving n^2 equations with 2 unknowns, which can be estimated with a least-squares to get the solution:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \Sigma_i f_{x_i}^2 & \Sigma_i f_{x_i} f_{y_i} \\ \Sigma_i f_{x_i} f_{y_i} & \Sigma_i f_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\Sigma_i f_{x_i} f_{t_i} \\ -\Sigma_i f_{y_i} f_{t_i} \end{bmatrix} \tag{4.5}$$

This solution provides the corresponding velocity vector of a particular pixel at that time.

The LK algorithm has been used in many US tracking applications. From studying the gastrocnemius medialis tendon in [45] to tracking the internal jugular vein in [73] to tracking the carotid artery in [74], Optical Flow algorithms like LK are observed to be generally accurate when compared visually or quantitatively to ground truth. However, LK has only been used when tracking a single morphological group - a single ventricle, artery, tendon, etc. Therefore, tracking various features in an US video like points in soft tissue, bone, and adipose were be a new feat for this tracker.

4.4 Anti-Speckling

Initial visual inspection of these algorithms revealed that drifting occurred consistently, especially in longer videos with more tissue and bone movement. To partly tackle this issue, the wavelet denoising algorithm from Python's Scikit library was added, as it was shown to increase the Peak Signal to Noise Ratio (PSNR) of each frame. Tracking visually improved after this process.

4.5 Drifting Correction Methods

4.5.1 Sigmoid Tracking Correction

In general, the issue of cumulative drifting in the frame-to-frame algorithms described above was decreased significantly with anti-speckling techniques when analyzing videos with a low-movement activity, such as when resting. This unfortunately did not fully eliminate the problem for videos with a high-movement activity. In an attempt to minimize the effect of drifting using labeled data, an algorithm that corrects drifting motion over time is proposed.

The movements performed by the subjects were both slow and repetitive which made the drifting patterns of the bounding boxes somewhat predictable. If drifting did occur during tracking, it often had a slow and steady progression away from the feature of interest. For this reason, a method called Sigmoid Tracking Correction (STC) is proposed.

The idea behind STC is relatively simple - if a point being tracked drifts away from the desired feature, and points were labeled at some initial and end frames, a smooth function can be added to connect the labeled points while maintaining most of the higher-frequency tracking movements. More explicitly, say a point's coordinate at frame i is

$$\boldsymbol{x}_i$$

and the point's coordinate at the next labeled frame $i + T$ is

$$\boldsymbol{x}_{i+T}$$

When the point is tracked from frame i to frame $i + T$, it drifts to some:

$$\boldsymbol{x}_{i+T} + \boldsymbol{d}$$

An example of this drifting is shown in Figure 4-3 below. As tracking occurs from some initial Labeled frame 1 to the next Labeled frame 2, the original yellow point in Figure 4-3(a) drifts away from the desired (purple) point on Figure 4-3(b). The

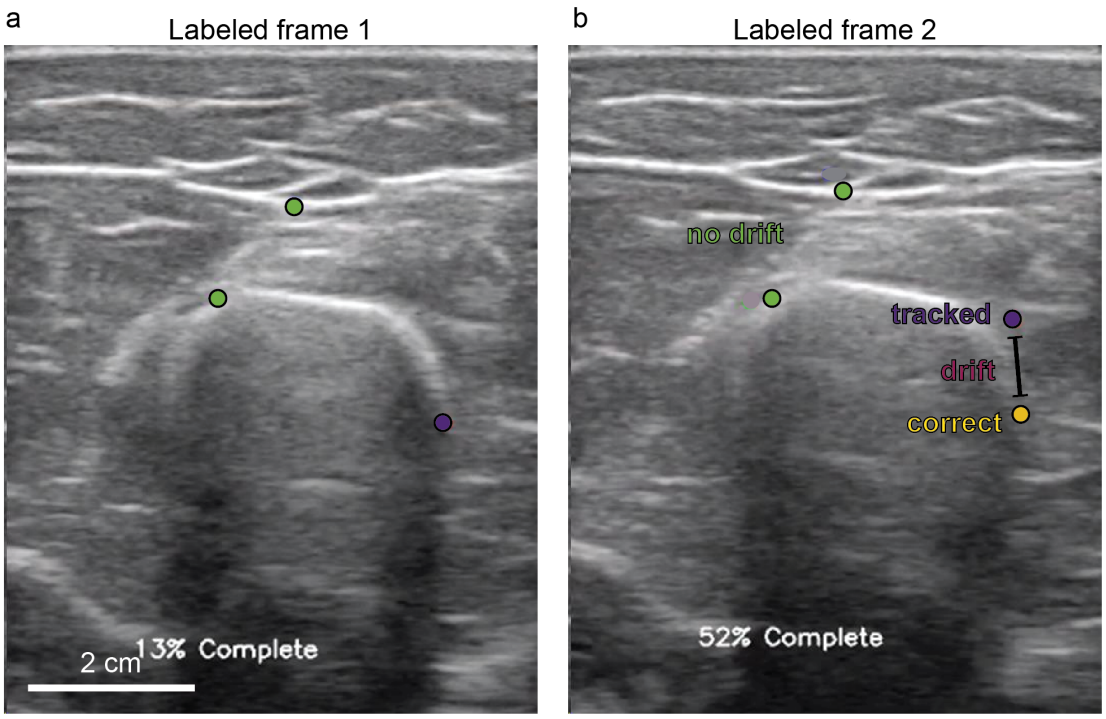


Figure 4-3: (a) The video at some initial labeled frame. (b) The video at the next labeled frame, with the yellow point representing where the purple point on (a) should have been tracked to.

drift value $\boldsymbol{d}$ is therefore just the distance between the yellow and purple points on Labeled frame 2.

To correct the point's drift, a sigmoid function is added to the frame's tracked coordinates between some initial and following labeled frame. Sigmoids have the form:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{4.6}$$

which are smooth functions that approach 0 and +1 as x approaches $-\infty$ and $+\infty$ respectively, with a maximum slope at $x = 0$. The function can be parameterized to the form:

$$S(x) = \frac{1}{1 + e^{-B(x-C)}} \tag{4.7}$$

and tuned to shift this function accordingly. The value of A scales the height of

the function as x approaches $+\infty$, the value of B determines the maximum steepness of the function (slope= $\frac{AB}{4}$) at its midpoint, the value of C determines the horizontal shift of the function, and D determines the vertical shift of the function. This function only approaches its asymptotes, so the drifting path error can also only be corrected up to a small tolerance ϵ times the error d. An ϵ of 0.01 is chosen in this case to reach a 1% error. Therefore, the new coordinate after summing the appropriate sigmoid function at the initial labeled frame $j = i$ should be

$$\boldsymbol{x}_{new,j} = \boldsymbol{x}_j + \epsilon \boldsymbol{d} \tag{4.8}$$

and the new coordinate right before the next labeled frame $j = i + T - 1$ should be

$$\boldsymbol{x}_{new,j} = \boldsymbol{x}_j + (1 - \epsilon)\boldsymbol{d} \tag{4.9}$$

Knowing the coordinate should approach $\boldsymbol{x}_j + \epsilon \boldsymbol{d}$ instead of 0 at $-\infty$, and $\boldsymbol{x}_j + (1-\epsilon)\boldsymbol{d}$ instead of 1 at $+\infty$, it can be concluded that the vertical shift to the sigmoid function should be $D = \boldsymbol{x}_i$. The horizontal shift to the function is determined by the location of the midpoint which is $C = 0$ for the standard sigmoid. For this application, the midpoint between the initial and end frames (and therefore the horizontal shift) should be $C = i + T/2$. The scaling factor should be the drift $A = \boldsymbol{d}$, as this is the coordinates' difference in the initial and end frames. Substitution of these three terms into (4.7) for $i = 0$ (assuming tracking starts at the first frame) is required to find the last parameter B:

$$\boldsymbol{x}_0 + \epsilon \boldsymbol{d} = \frac{A}{1+e^{-B(0-C)}} + D$$

$$\boldsymbol{x}_0 + \epsilon \boldsymbol{d} = \frac{\boldsymbol{d}}{1+e^{-B(-T/2)}} + \boldsymbol{x}_0$$

$$\epsilon = \frac{1}{1+e^{BT/2}}$$

$$\epsilon + \epsilon e^{BT/2} = 1$$

$$e^{BT/2} = \frac{1}{\epsilon} - 1$$

$$B = 2ln(\tfrac{1}{\epsilon} - 1)/T$$

Therefore, the full algorithm becomes:

$$\text{from } j = 0 \text{ to } j = T - 1\text{:}$$

$$\boldsymbol{x}_{new,j} = \boldsymbol{x}_j + \frac{\boldsymbol{d}}{1 + e^{-2ln(1/\epsilon - 1)(j - T/2)/T}} \tag{4.10}$$

In the context of US tracking, this algorithm would be applied to tracking points for both x and y pixel coordinates.

In Figure 4-3, the tracking of the purple point drifts away from the desired feature to some higher y-coordinate. Figure 4-4 graphically shows an example of this algorithm in action. Once the labeled frame (around frame 2600) is reached, the tracker is reset to the desired coordinate, and the appropriate sigmoid function is added to compensate for the drift.

STC was used to improve the issue of drifting for all frame-to-frame trackers. Because STC requires labeled data to exist, it was important to understand how the quantity of labeled data improves tracking accuracy. 0-4 labeled frames per video were used for correction, called "correction frames," which dictated at what frames the algorithm would reset the tracked points to their labeled location. As mentioned in the Labeled section, each subject had 8 equally distributed labeled frames per video, so the chosen correction frames were also distributed evenly throughout. For 1 correction frame, labeled frame number 4 was used for correction. For 2 correction frames, labeled frame 3 and 6 were used for correction. For 3 correction frames, labeled frames 2, 4, and 6 were used for correction. And for 4 correction frames, labeled frames 1, 3, 5, 7 were used for correction.

4.5.2 Reversed Tracking Correction

STC is a heuristics-based technique that requires labeled frames throughout the video of interest and is used after tracking is complete. In the scenario where no labeled

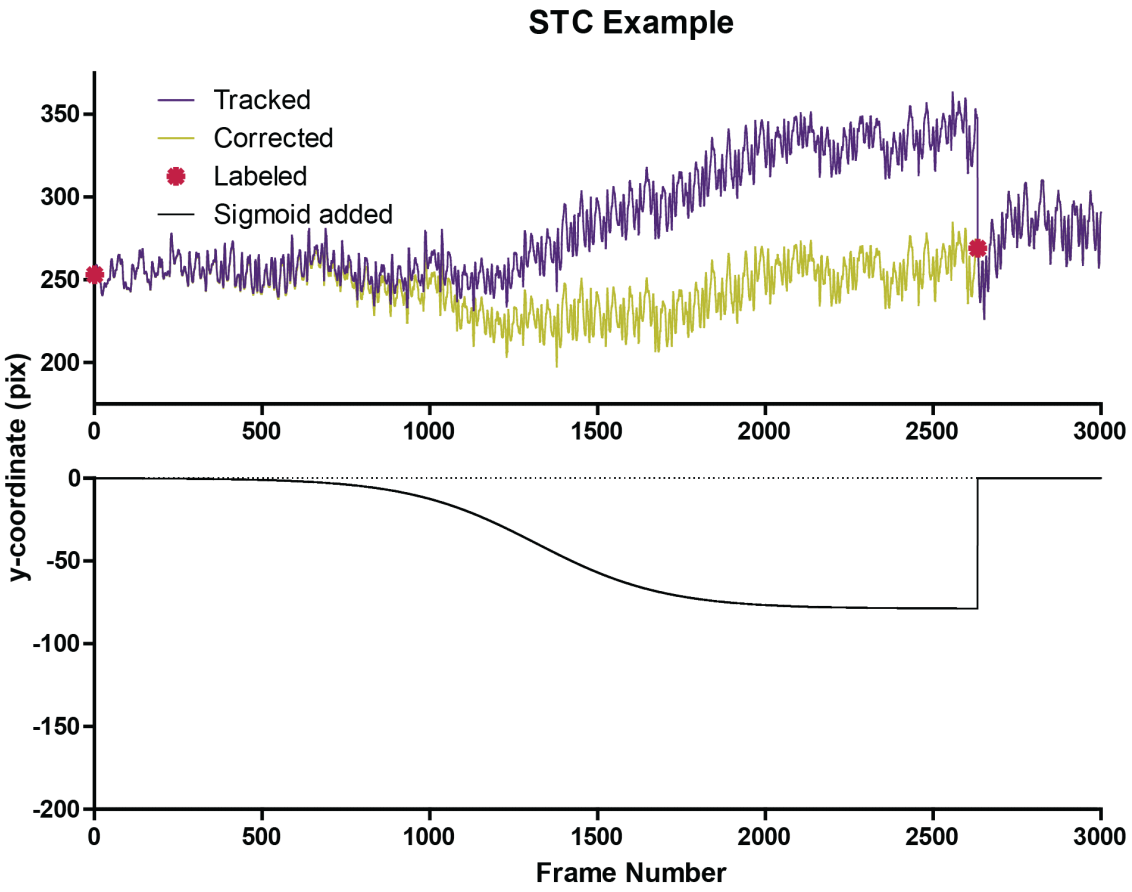


Figure 4-4: This shows how STC can be applied to the example in Figure 10. The purple line shows the y-coordinate of the purple point as it's tracked. The green line shows the path this point takes after being corrected by the sigmoid in the lower plot.

point data is available, a method called Reversed Tracking Correction (RTC) was proposed.

Given a set of initial points, one can assume in the worst scenario that drifting will increase error throughout the course of a video. Under this assumption, if the video was reversed with a set of initial points, running the same frame-to-frame tracker will also lead to drifting with increasing error. Using this information, a given tracker was applied normally to the original video starting at the start frame's labeled points, and the tracked points were stored as the forward path P_f. Then, the tracker is applied to the reversed video starting at the end frame's labeled points, and tracked points were stored as the backward path P_b.

P_f ,P_b (Nxpx2) $\rightarrow$ path forward, path backward (x, y coordinates)

$$t_i, t_e \rightarrow \text{initial frame, end frame}$$
$$N = \text{number of frames, } p = \text{number of points}$$

RTC averages the two paths $\frac{P_f + P_b}{2}$, returning a new tracked path that includes an equal contribution from both.

4.5.3 Reversed Sigmoid Tracking Correction

STC is useful when many frames throughout a video are labeled. RTC can be useful when only the start and end frames of a video are labeled. Reversed Sigmoid Tracking Correction (RSTC) combines ideas from both methods to potentially improve accuracy given the same minimal information as in RTC. However, instead of an equal contribution from both paths, RSTC creates a new path that is more influenced by the forward path near the start of the video, and more influenced by the backward path near the end of the video. To do so, it scales the forward path by a sigmoid function with a value near 1 at the start frame, and a value near 0 at the end frame, and repeats this in reverse for the backward path. The sigmoid function parameters are found using a similar process as in Eq (4.10).

$$s_f = \frac{1}{1+e^{-B(x-C)}},\ s_b = \frac{1}{1+e^{B(x-C)}}$$
$$\text{where } B = 2ln(\tfrac{1}{\epsilon} - 1)/(t_e - t_i) \text{ and } C = (t_i + t_e)/2$$
$$P_{fs} = s_f P_f,\ P_{bs} = s_b P_b$$
$$P_T = P_{fs} + P_{bs}$$

To visualize the difference between RTC and RSTC, the path of an example point's y-coordinate while tracking is shown below in Figure 4-5 below.

STC, RTC, and RSTC attempted to decrease the drifting errors of frame-to-frame trackers, but these heuristics-based approaches did not necessarily fix full tracking failures or tracking subtle movements. In an attempt to incorporate training using the labeled frames, an existing method of tracking using Convolutional Neural Networks (CNNs) will be described next.

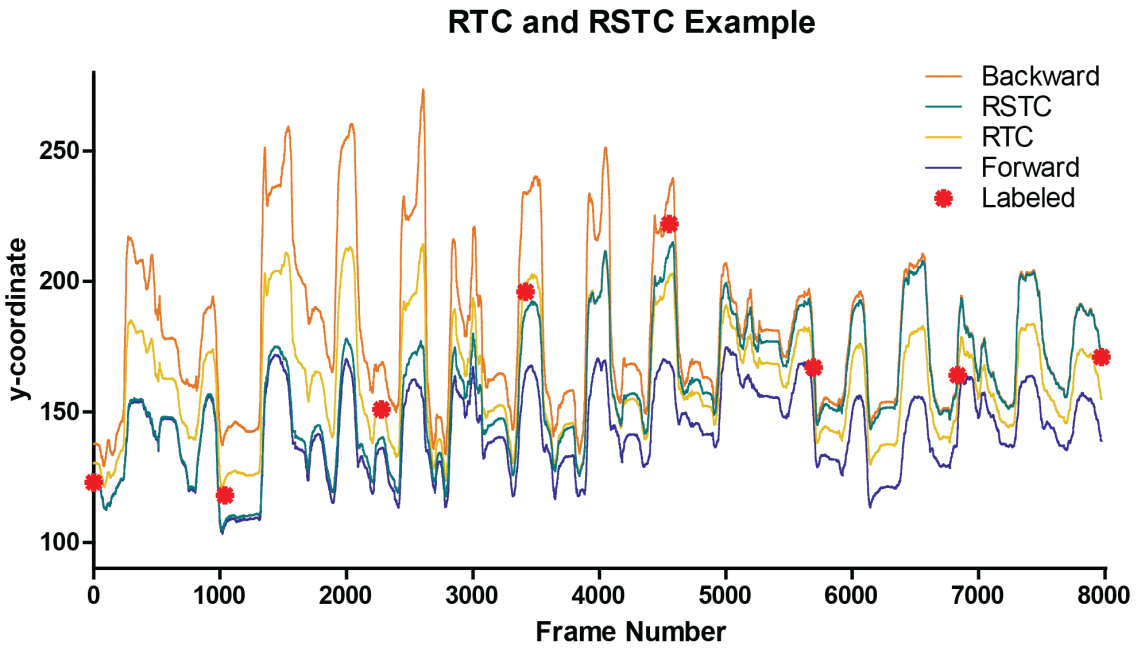


Figure 4-5: Visual of RTC and RSTC using the y-coordinate of an example point. The purple line represents the path of the point tracked forward in time. The orange line represents the path of the point tracked backward in time. The yellow line is the averaged path from RTC. The green line is the sigmoid-scaled path from RSTC. The red points are the manually labeled points.

4.6 Tracking Methods - DeepLabCut

DeepLabCut (DLC) is an open-source software used typically for markerless feature tracking of animals [75]. It uses the feature detector architecture from DeeperCut [76], a high-accuracy human pose-estimation algorithm that includes a network pretrained on more than 1.4 million images. This allows it to use a small number of training images (in order of a few hundred) to achieve human-level accuracy. DLC combines the pretrained ResNet encoder (typically used for object recognition with excellent performance) and deconvolutional layers (used to produce spatial probability densities) into a robust tracking CNN [75].

While DLC has been used extensively to track animal and human features during movements, its ability to track features in US videos has been minimally explored. DLC has been used to track the upper surface of the tongue, comparing it to other US contour estimators [77], concluding that DLC requires significantly less training

data to perform with the same level of accuracy. It has also been used to track the gastrocnemius muscle-tendon junction, observing the morphology of the lower leg longitudinally [78]. DLC has also been used as a general CNN to track hyroid kinematics during swallowing, comparing its accuracy to other manual and automatic hyroid-specific trackers [79]. All related applications of this package not only focused on tracking different anatomies than those studied in this paper, but they also focused on tracking a singular feature. In this paper, DLC is used to simultaneously track features in soft tissue, bone, and adipose while comparing its accuracy with non-CNN trackers.

To use it, DLC has a Python toolbox [80] used to facilitate the manual steps involved with selecting videos, labeling points, and training and evaluating the network. The toolbox can be accessed via any text editor or its own GUI. As described in Section 4.1, both subject groups were labeled differently to tackle how tracking various features (Group 1) varied from tracking specific features across subjects (Group 2). Subjects in Group 1 had 8 equally-spaced labeled frames per video, and half of those labeled frames across all videos were chosen as training data. Subjects in Group 2 had 10 equally-spaced labeled frames per video, and 80% of the videos were chosen as training data. The rest of the frames and videos for these groups was chosen as testing data on which the DLC models would be evaluated. The manually labeled data of both Group 1 and Group 2 was restructured into the appropriate file types for DLC to use for training. The models were trained using a 50-layer ResNet network and the imgaug augmentation method for 500k iterations, the default number chosen by DLC's GUI that resulted in plateauing errors in [77] for ResNet50.

Although this series of steps typically performed visually well, DLC had an additional feature that checked the results for outlier frames by observing large changes in feature trajectories. Depending on the accuracy when manually labeling and the robustness of that particular network, the 144 originally labeled points might not have sufficiently eliminated these outliers, so the option to label additional frames was provided. However, in order to establish a standard benchmark in which all trackers used the same labeled frames, this outlier-detection feature was not used.

4.7 Tracking & Machine Error

In order to compare the accuracy of all algorithms, the error of interest was measured as the Euclidean distances between the ground truth labeled points and the center of the tracker's estimated bounding boxes. This was calculated both before and after correction to compare both the accuracy between frame-to-frame algorithms (like OpenCV trackers) and trained deep networks (like DLC) given a specified number of labeled frames. These same distance metrics were used to compare tracking accuracy depending on the type of movement performed, or the number of correction frames used.

Next, to understand the impact of cumulative drifting, 8 copies of each labeled video were made and cropped to have an increasing number of frames (N) - in this case, $N = 200, 400, \ldots\ 1400, 1600$. Each copy was then reversed and concatenated to the end of the non-reversed copy to create videos with $2N - 1$ frames as if each copy was mirrored onto itself. Then, from the starting frame and its labeled points, each tracker was applied to these copies, and the distances between the points at the start and end frames were measured. Because these copies are mirrors of themselves, the start and end frames were the same, so an ideal non-drifting tracker would return an error of 0. The larger the error between the points, the worse the drifting for that tracker.

Finally, to analyze the speed of these various methods, the time to completion of each algorithm was determined. A computer with a 3GHz CPU and 64GBs of RAM was used (Intel i9-9980 XE) to run all frame-to-frame trackers, and only one tracking process ran on it at a time. Another computer's 1860MHz GPU (NVIDIA GeForce RTX-3090 Ti) was used to run all DLC processes, as the DeepLabCut GitHub (https://deeplabcut.github.io/DeepLabCut/) explains code will run an order of magnitude faster using a GPU.

Chapter 5

Results

5.1 Tremor Characterization

5.1.1 Accelerometry

As described in Table 2.1 (Trial #6), the subjects from Group 2 performed additional reaching trials with various interventions. The control movement to establish a baseline Tremor Proportion in Movement (TPM) was reaching while looking at a ball move on a **screen**. The interventions for the other trials included reaching only after a **verbal cue**, reaching while looking at the tip of one's index **finger**, reaching only after **imagining** the movement, and reaching after being told to minimize or maximize the tremor visible in the live accelerometer signal (**biofeedback**). The goal of these interventions was to determine whether external stimuli could sufficiently alter the movement to change one's TPM. First, Figure 5-1 highlights the results.

Most interventions had little to no effect on the TPM of a subject. Across all muscles, the trials imagining the reaching movement before doing it did not produce a significant change compared to the control trials ($t_i = 0.075$, $p_i = 0.941$), nor did the trials looking at one's finger ($t_f = 0.244$, $p_f = 0.808$) or the trials waiting for a verbal cue ($t_c = 0.790$, $p_c = 0.434$). The only intervention where a change in TPM was seen was during the biofeedback trials ($t_{max} = 3.964$, $p_{max} < 0.001$; $t_{min} = 3.150$, $p_{min} = 0.002$).

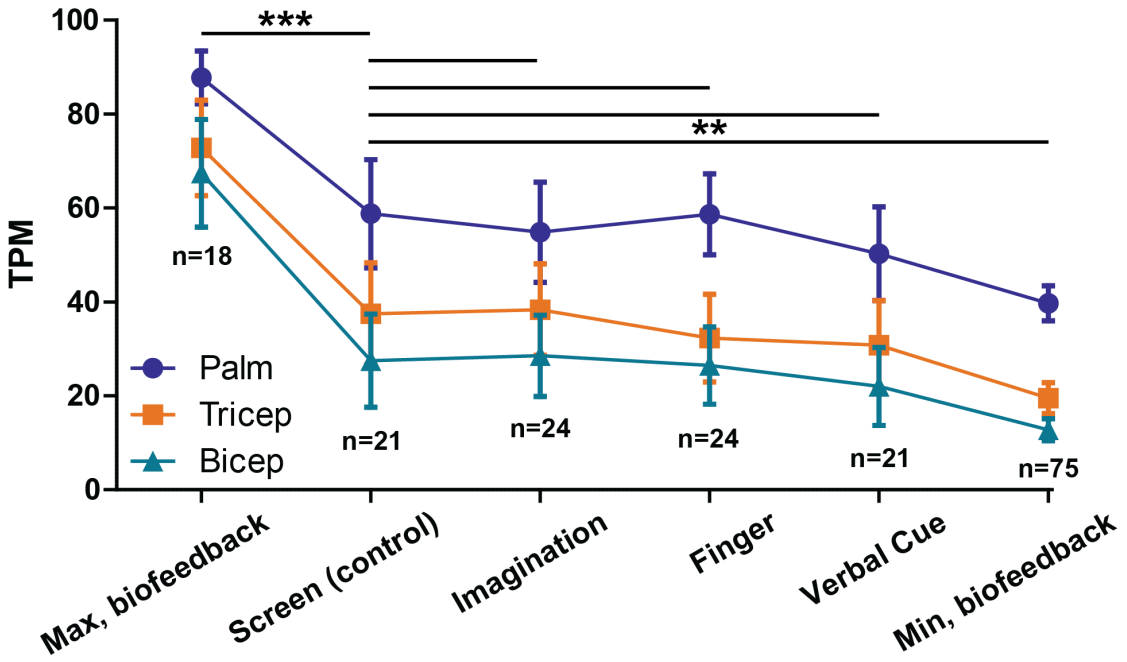


Figure 5-1: TPM when reaching under various interventions.

In order to ensure the movements were done as ordinarily as possible, subjects were not made aware that the experiment analyzed the occurrence of tremors until immediately before the biofeedback trials when they were shown the tremor-like signal and instructed to purposefully alter it. So although not many interventions altered TPM, the main takeaway from these results was that subjects in this group were *able* to modify their movement once a visual stimulus of their own physiologic tremors was presented.

This brought the effect of sight into consideration. Some of the trials without biofeedback were split into trials performed with eyes open and eyes closed. Figure 5-2(a) shows there was no significant difference in TPM for any movement type (screen: $t_s = 0.607$, $p_s = 0.555$; imagination: $t_i = 0.335$, $p_i = 0.743$; verbal cue: $t_v = 0.109$, $p_i = 0.915$). For the trials with biofeedback, the subjects were shown the signal from each muscle of interest individually - the triceps, biceps, and palm - to try to minimize the visible tremor. Figure A-1 in Appendix A shows an example of all 3 signals. Figure 5-2(b) displays how observing a different signal varied the TPM, similarly showing no significant difference in TPM (ANOVA $F = 0.525$, $p = 0.599$)

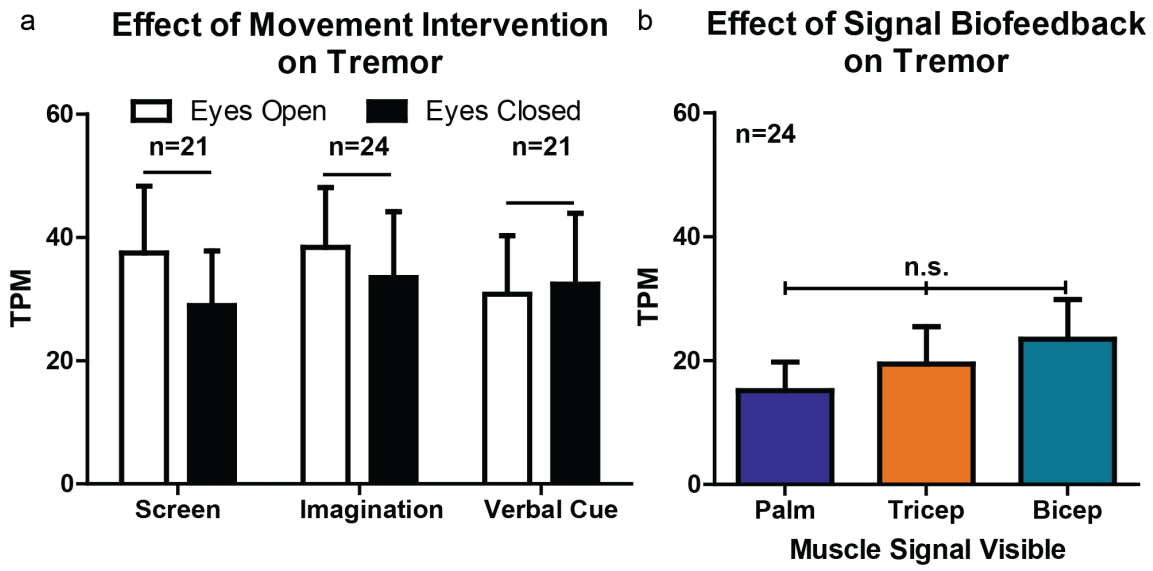


Figure 5-2: TPM variations (a) when subjects performed the movement with eyes open vs closed, and (b) when subjects observed the signals from different sensors during biofeedback.

This figure only highlights TPM from the triceps, but this pattern held for all 3 muscles.

5.1.2 Motion Capture

As mentioned in Section 3.3, it was also useful to look at motion-capture data for these tasks to determine whether tremors occurred more often during the extension or retraction. The average angular velocity while the filtered accelerometer signal was above the tremor-threshold was calculated for each muscle, and the results are shown in Figure 5-3(a).

5.1.3 EMG

The next signal of interest was EMG to better understand how muscle activation and TPM were related. Because tremors were significantly more prominent during retraction, the time derivative of EMG was similarly observed to determine whether tremors were more prevalent during contraction (positive derivative) or relaxation (negative derivative) of any particular muscle. Figure 5-3(b) highlights these results.

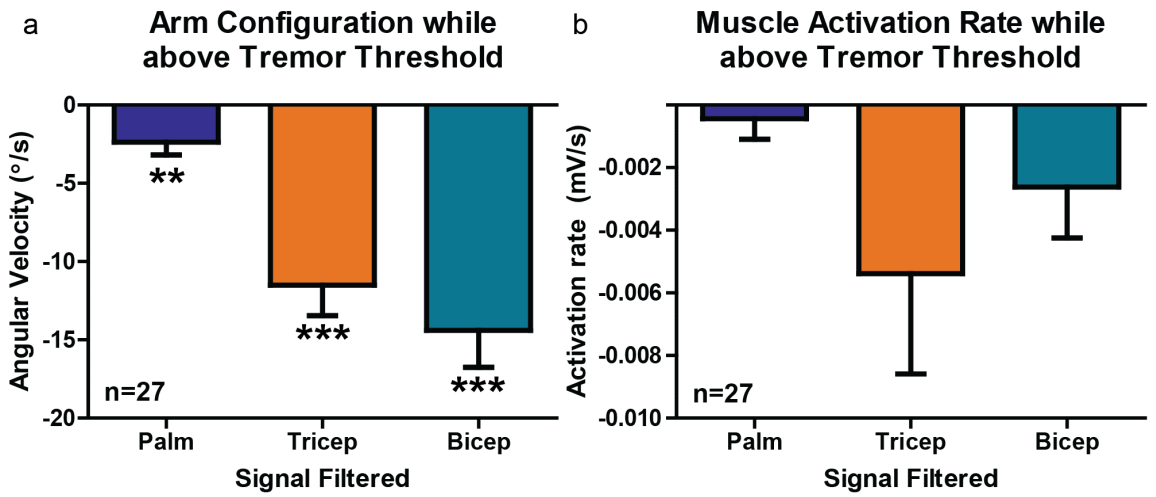


Figure 5-3: (a) Average angular velocity of the elbow angle while filtered triceps acceleration is above the tremor threshold. (b) Average EMG activation of each muscle as their filtered acceleration is above the tremor threshold.

Figure 5-3(a) illustrates that the overwhelming amount of tremors in all 3 muscles observed occurred during the retraction phase of the movement ($t_p = -2.941$, $p_p = 0.007$; $t_t = -5.888$, $p_t < 0.001$; $t_b = -6.080$, $p_b < 0.001$). From Figure 5-3(b), a similar trend exists regarding muscle relaxation, but no significant results emerged ($t_p = -0.662$, $p_p = 0.513$; $t_t = -1.683$, $p_t = 0.104$; $t_b = -1.616$, $p_b = 0.105$).

To understand the presence of tremors during retractions more thoroughly, each retraction was broken down into 8 equal time segments in which the triceps and biceps muscles were examined. First, the muscle EMG signals in each segment were averaged, grouping muscle activation during retractions into 8 segments. Then, the filtered acceleration signals within each segment were averaged, grouping tremor magnitude during retractions. Figure 5-4 highlights these results.

The elbow angle ranged from 149.4 ± 1.6^o when retractions began and from 84.5 ± 1.3^o when retractions ended. The duration of the retractions also ranged from 2.52 ± 0.07 s. Despite these variances, there was a distinctive and consistent movement pattern among subjects. Figure 5-4(a) illustrates that as the subjects first lowered their arms, their triceps relaxed to an activation comparable to the biceps. As the subjects finished retracting their arms, the activation of the biceps slowly decreased relative to the triceps. An initial hypothesis for the recurring presence of tremors during re-

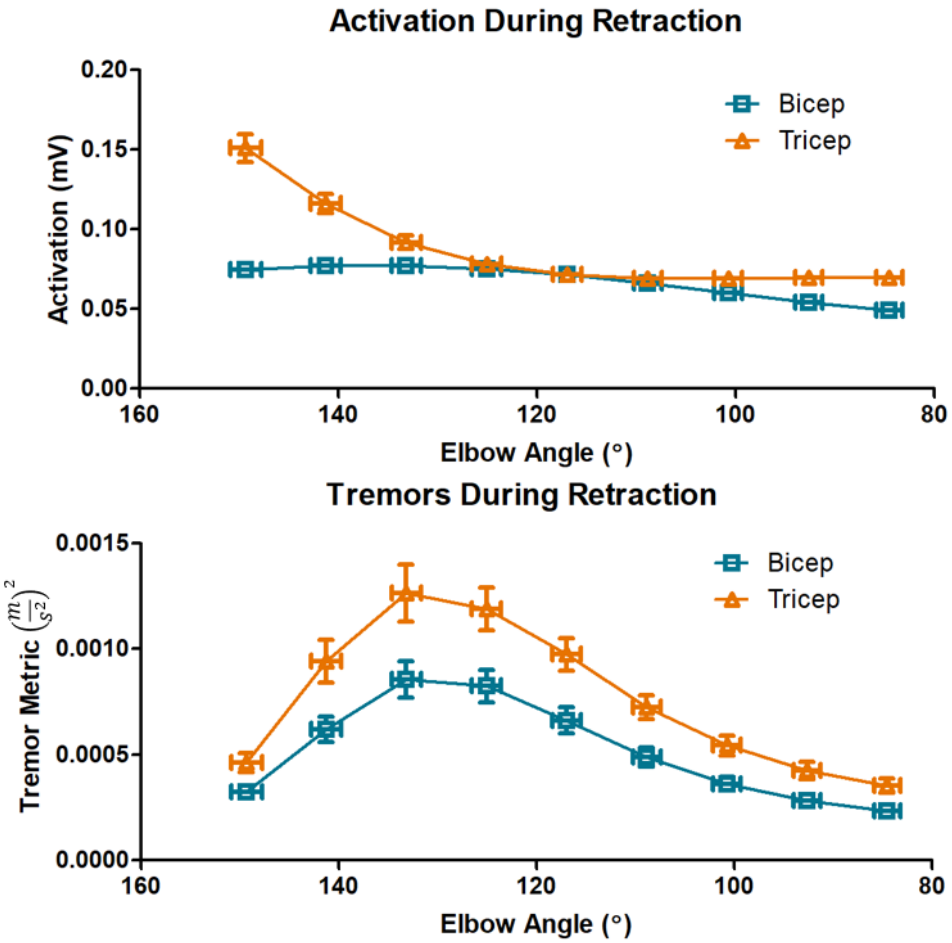


Figure 5-4: During the retraction of the arm during a slow reaching movement, (a) shows the activation of the triceps and biceps muscles, and (b) shows the tremor magnitude.

traction was that tremors occurred when antagonist muscles exchanged the need for activation. In other words, tremors would occur in this experiment when the biceps muscles took over load off the triceps muscles. Figure 5-4 highlights otherwise. During the relaxation of the triceps, the magnitude of the tremor quickly increased to a peak around 134^o in (b), while (a) does not show the muscles exchanging a loads around this angle.

5.2 Task Expertise - Future Work

One of the central motivators for this thesis was relating task expertise with the severity of a tremor. Before each data collection, subjects were presented with a survey in which they answered questions about their past experience with sports or other activities. Based on these responses, they were categorized into 6 levels of expertise involving arm movements:

- Level 0: Rarely active to not active
- Level 1: Currently somewhat active, not necessarily in a task involving arm movements.
- Level 2: Currently somewhat active in an arm movement task, or highly active in non-arm-movement task.
- Level 3: Currently highly active in arm-movement task, not much prior experience.
- Level 4: Currently active in arm-movement task, has competed locally.
- Level 5: Currently highly active in arm-movement task, has competed at a national or international level.

The goal of these categories was to group subjects and determine if a correlation between expertise and TPM was immediately obvious. For a fair comparison, only the subjects' control reaching trial (looking at the screen) was analyzed. Figure 5-5 shows these results.

Although a correlation appears present from this data, there were not enough subjects in the data collection to establish it as significant. Moreover, other confounding variables (a subject's uncertainty their in survey responses, unclear distinctions between expertise levels, a wide range of activities considered to involve arm movements, etc.) likely played a role in these results. In order for any relationship like this to be established, future work where a specific activity is chosen, clearer survey questions being asked, and more subjects participating will be required.

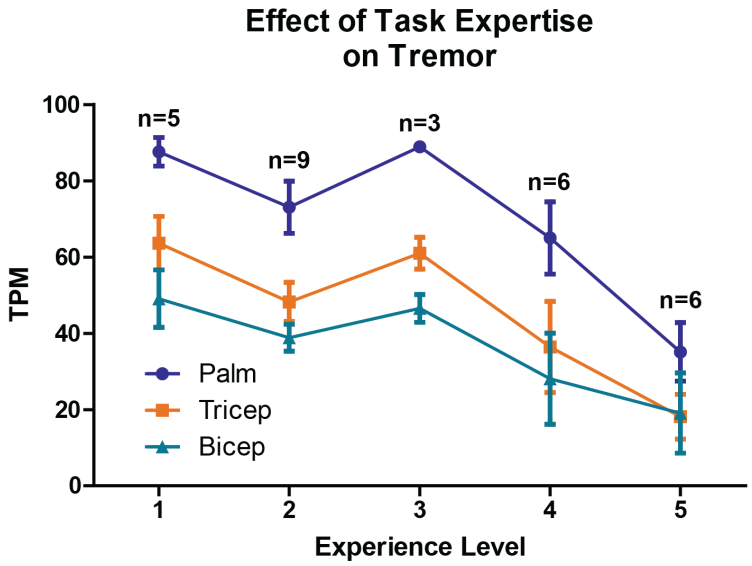


Figure 5-5: TPM based on rough levels of expertise in activities involving arm movements.

Finally, the results from the US analysis will be described.

5.3 Ultrasound Analysis

5.3.1 Cumulative Drifting

To first understand how cumulative drifting affected each frame-to-frame tracking algorithm, each tracker followed 3 distinct features starting at the first frame, tracking forward to N frames of a video, then tracking backward the same N frames. Since this process should start and end in the same frame, the error was simply the cumulative distances between the point locations in the first and last frames. Figure 5-6 depicts how drifting error accumulated as N increased, comparing both the various trackers in 5-6(a) and various movements performed in 5-6(b). Accuracy in all future US sections will refer to the summed distance between the labeled points and the tracked points in pixels, and for this application, 1 pixel is equivalent to 0.123mm.

From Figure 5-6(a), it was clear that the CSRT tracker experienced the most drifting over the course of a video ($\bar{x}_{CSRT} = 110.5$ pixels over 3200 frames), while all other trackers had low drifting ($\bar{x}_{KCF} = 42.4$ pixels, $\bar{x}_{MOSSE} = 36.6$ pixels, $\bar{x}_{LK} =$

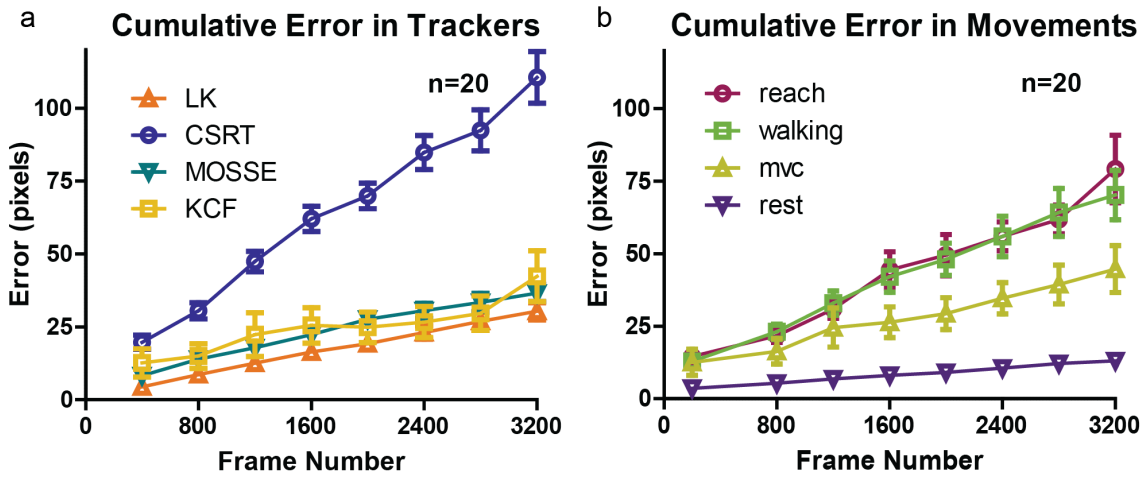


Figure 5-6: (a)Cumulative error between frame-to-frame trackers. The average error rate for the CSRT tracker, for example, is 0.0345 px/frame, or 4.25μm/frame. (b) Cumulative error between certain movement types.

30.4 pixels over 3200 frames). Additionally, Figure 5-6(b) shows that high-movement and high-impact activities such as walking and reaching resulted in the most drifting by the trackers ($\bar{x}_{walk} = 70.2$ pixels for walking, $\bar{x}_{reach} = 79.2$ px for reaching over 3200 frames), likely due to the high pixel movement within the US videos. Despite some trackers and movements resulting in lower cumulative errors than others, Figure 5-6 generally displays that cumulative drifting was a challenge present in frame-to-frame tracking across the board.

5.3.2 Drifting Correction

Sigmoid Tracking Correction (STC) required at least two frames with points labeled - one initial frame for the trackers to start and at least one correction frame with points for the tracker to reset to. Each tracker was run from the initial labeled frame to the first correction frame, the points were reset, and the process was continued repeated for an increasing number of correction frames all equally spaced apart. If more correction frames were used, the space between each one decreased accordingly. After tracking was complete, STC was applied as described in Section 4.5.1. Figure 5-7(a) summarizes these results.

Next, to use Reverse Tracking Correction (RTC) and Reverse Sigmoid Tracking

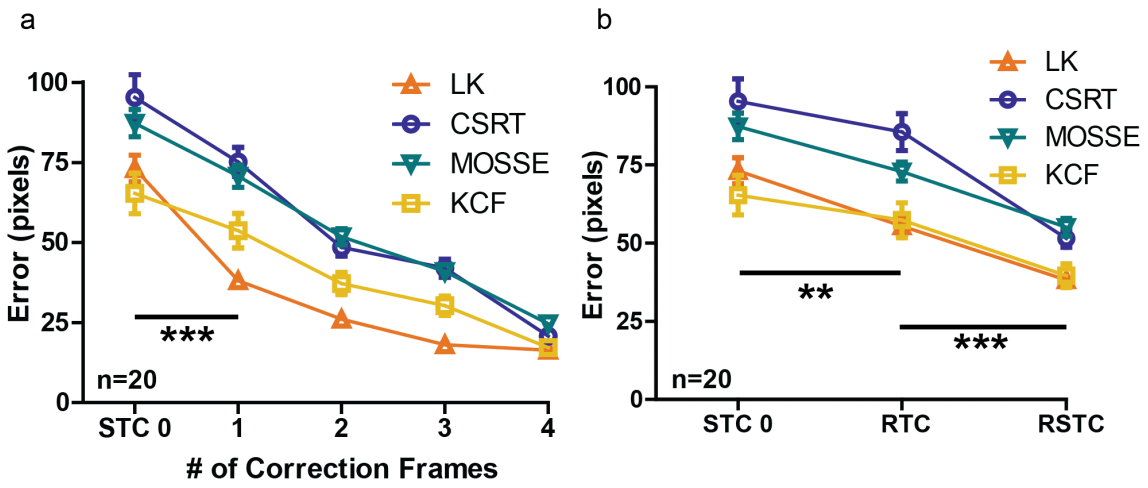


Figure 5-7: (a) Effect of increasing the number of correction frames on tracking accuracy, and tracking improvement due to STC across trackers. (b) Accumulated errors across trackers using RTC and RSTC. The left-most mean errors in (a) and (b) represent tracking error with no correction.

Correction (RSTC), each tracker was uninterruptedly run forward and backward. For RTC, the tracked paths were simply averaged. For RSTC, the contribution from the forward path near the initial labeled frame was increased, and the contribution from the backward path near the last labeled frame was increased. The error values when no drifting correction was used, when RTC was used, and when RSTC was used are compared in Figure 5-7(b).

As expected, using more correction frames with STC to guide any tracker led to a lower average error, with significant improvements with just 1 correction frame ($t = 5.22$, $p << 0.001$) among all trackers ($N = 80$). Using STC seems to provide an efficient way for any tracker to lower its error by providing sufficient labeled frames to pull the drifting points towards correct data. However, it is important to restate that error is measured at these frames, so when more correction frames are used, more errors equal to 0 will contribute to the average shown in the figure.

Using less labeled frames with RTC still significantly decreased error when compared to using the tracker normally ($t = 3.11$, $p = 0.002$ among all trackers), and using RSTC also significantly decreased error when compared to using RTC ($t = 7.01$, $p << 0.001$) among all trackers ($N = 80$). LK and KCF consistently resulted in the

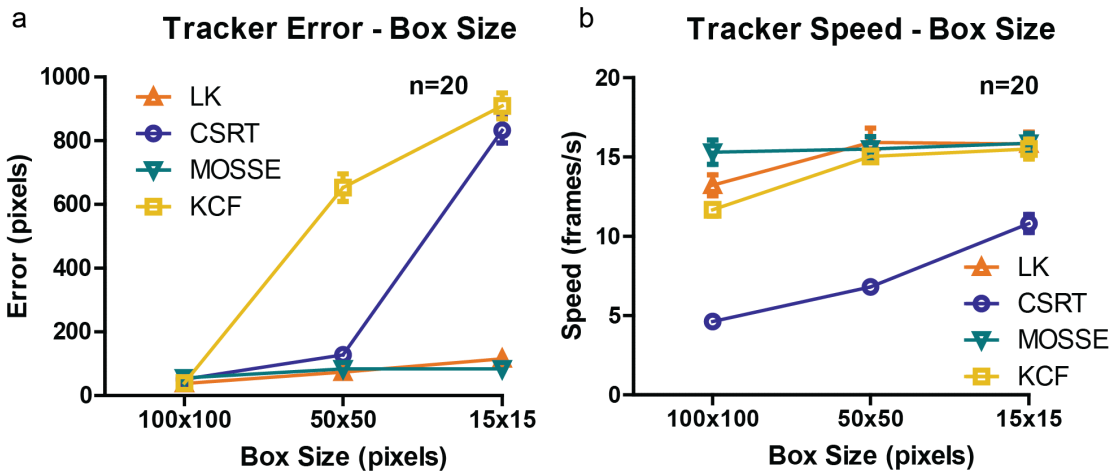


Figure 5-8: (a) Accuracy of frame-to-frame trackers using various bounding box sizes. (b) Speed of frame-to-frame trackers using various bounding box sizes.

lowest error when compared to CSRT and MOSSE.

The key takeaway for Figure 5-6 is that implementing these methods can have a significant impact in the accuracy of frame-to-frame trackers. Figures 5-6 and 5-7 also display the benefits of using trackers like KCF and LK in these applications.

5.3.3 Bounding Box Sizes

OpenCV trackers are initialized by drawing a box bounding the region of interest. These box sizes have the potential to alter tracker accuracy and speed, so the errors across all trackers using three different box sizes - 100x100 pixels, 50x50 pixels, and 15x15 pixels - are compared in Figure 5-8(a).

KCF performed well when tracking with a large box (100px), but it consistently ran into tracking failures consistently with a mid-sized or small box (50px, 15px). CSRT similarly performed poorly with a small box, but not as poorly as KCF using mid-sized boxes (50px). LK's error also increased with box size, but to a much lesser extent than KCF and CSRT. MOSSE resisted failures and maintained a relatively low error for all box sizes. Generally, in order to obtain the lowest error values when tracking, using a large box size was optimal.

There are a few distinctions to make when comparing the quality of the trackers

aside from the quantitative error metric found above. As shown in 5-8(a), KCF with a 100x100 pixel box had one of the lowest accumulated errors compared to its other frame-to-frame tracker counterparts, but the total average error for this tracker when using a 15x15 pixel box increased by an order of magnitude. When inspecting the tracking quality, we noticed this high error was due to tracking failures in which we reset the boxes to the origin coordinate. A large box size is recommended to prevent these failures from occurring.

Next, the speed of each tracker was calculated (in frames per second) using these bounding box sizes. CSRT was consistently the slowest of the algorithms while the MOSSE tracker did not significantly change speed as its box size changed. The speed of LK and KCF both plateaued as box size decreased. Figure 5-8(b) shows that one of the biggest caveats of using a larger box for improved tracking is computational cost. However, due to the substantial error increases relative to the minimal increases in speed for all trackers, it is still recommended that a larger box size is used. Although slower, it will prevent them from potential tracking failures and improve accuracy significantly.

5.3.4 DeepLabCut

Up to now, the frame-to-frame trackers were analyzed among one group of subject videos (Group 1) in which each subject's US videos were labeled with a unique template. Next, DeepLabCut (DLC) was used to train individual models for each subject in Group 1 using the same 4 correction frames used for STC, training for 500,000 iterations. The average error across all models was found to be 9.9 ± 0.4 pixels, shown as the leftmost point in Figure 5-9(a), significantly lower than that obtained with any frame-to-frame trackers using drifting correction.

It is likely these models performed this well because one model was trained per subject, meaning each model only learned the morphological characteristics of one bone-muscle group. To test the effectiveness of a generalized DLC model across multiple subjects, the second group of videos (Group 2) was used. A general template frame was chosen, three points along the edge of the humerus were labeled, and three

models were trained using 80% of the videos - one using 500,000 training iterations, one using 750,000, and one using 1,000,000 - to understand how overfitting could play a role in the accuracy of these generalized models. The remaining 20% of videos were randomly placed into 20 groups of 6 videos each, and the three models were applied to these groups. For each model (500K, 750K, 1M), the errors of all 20 groups were found, and their averages and standard errors are shown in Figure 5-9(a).

As expected, the error of all Group 2 models was significantly greater than that of the Group 1 model ($t = 7.18$, $p << 0.001$), as the Group 2 models learned the characteristics of multiple bone-muscle groups to track the given points. However, the average errors in Group 2 were not significantly different among the models with increasing training iterations (500K to 750K to 1M, ANOVA $F - value = 0.26$, $p = 0.77$), showing the robustness of DLC to common underfitting or overfitting issues in NNs.

An additional consideration when choosing a tracking method is computational cost. For all the models, the time to apply the models to each group of videos was found, and the speed was calculated by average number of frames processed per second. Figure 5-9(b) shows that once a model is trained, applying that model to track points in any video has similar computational cost regardless of the number of iterations that model took to train.

Computational speed was also important for DLC. The time to apply the models to each group of videos was found, and the speed was calculated by average number of frames processed per second. Figure 5-10(a) shows that once a model is trained, applying that model to track points in any video has similar computational cost regardless of the model accuracy in tracking or the number of iterations that model took to train.

More details about the computational power used to run these various algorithms are detailed in Section 6.6 of the main text, as DLC runs at a much higher speed on a computer with a GPU.

If DLC is chosen for tracking features in these types of US videos, it is recommended to use 500,000 training iterations as it generally results in faster training

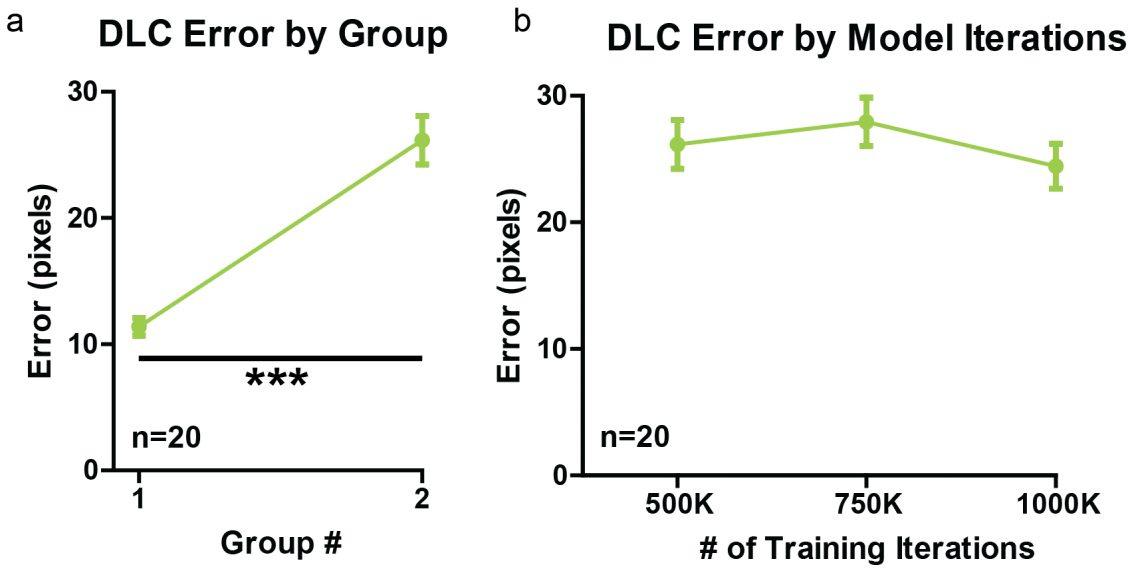


Figure 5-9: (a) Accumulated errors when using one DLC model per subject (Group 1) versus one model per group of subjects (Group 2). (b) Tracking errors when increasing the number of training iterations.

times with comparable accuracy to models using higher training iterations.

5.3.5 Method Combinations

Figures 5-7 and 5-9 show that tracking error when using DLC is relatively low compared to using uncorrected frame-to-frame tracking, but have yet to compare DLC to our correction methods directly.

Figure 5-6(a) shows that the average LK drifting error over the course of 200 frames (4.4 pixels) was lower than that seen using DLC in Group 1's models (11.4 pixels). With this in mind, these methods were combined by using every 200th frame tracked by DLC as pseudo-labeled frames, tracking forwards and backwards from each of these frames using LK, and applying RSTC to reduce drifting effects (**LK + DLC + RSTC**).

Figure 5-6(a) shows that using STC with 4 correction frames resulted in a low average error of 19.7 pixels among all trackers, but the contribution to that average from LK was an error of 16.4 pixels (**LK + STC**). In an attempt to decrease this, STC and RSTC were combined by tracking from each of the 4 correction frames

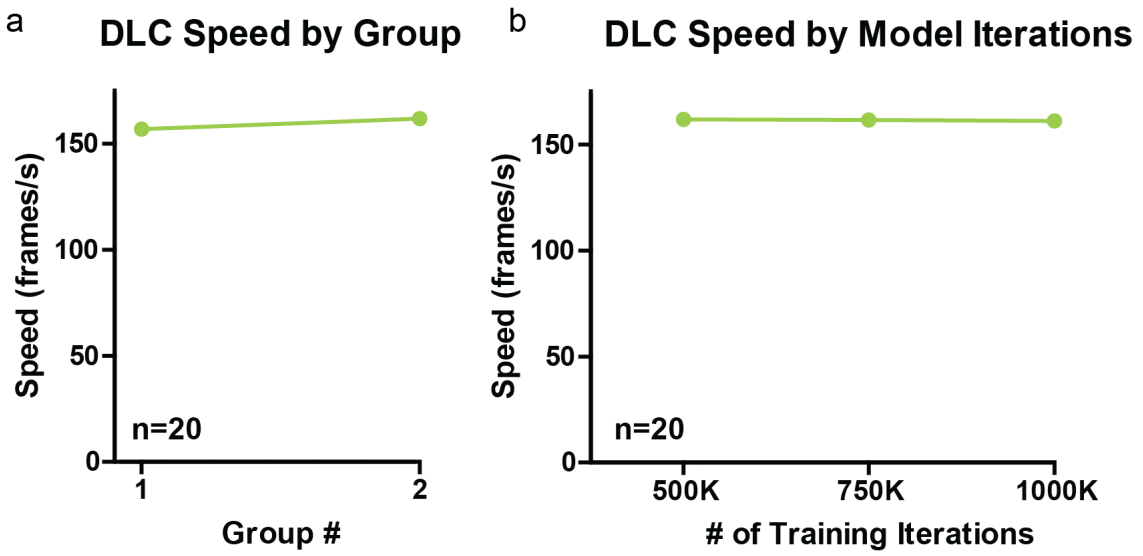


Figure 5-10: (a) Speed of applying the two respective DLC models to Groups 1 and 2. (b) Speed of applying the 3 DLC models trained with increasing training iterations to Group 2.

forwards and backwards with LK, and applying RSTC to these paths (**LK + STC + RSTC**).

Finally, the errors were grouped to compare each method's sensitivity to movement, shown in Figure 5-11.

This figure presents one of the main advantages of using DLC – robustness. The range of errors when using DLC was smaller than any other method here, showing it was less sensitive to large or fast tissue movements when tracking. However, using LK + RSTC + STC resulted in an error not significantly different than that obtained by DLC across movements ($n = 80$, $t = 1.768$, $p = 0.085$), highlighting that these correction methods have the potential to improve accuracy of frame-to-frame trackers to the level of NN models.

5.3.6 Labeling & Human Error

Up to now, error values have used one set of labels as ground truth; however, as discussed in Section 4.1, labeling often involved rough estimates of feature location. To understand human variability when labeling, the points were relabeled two more

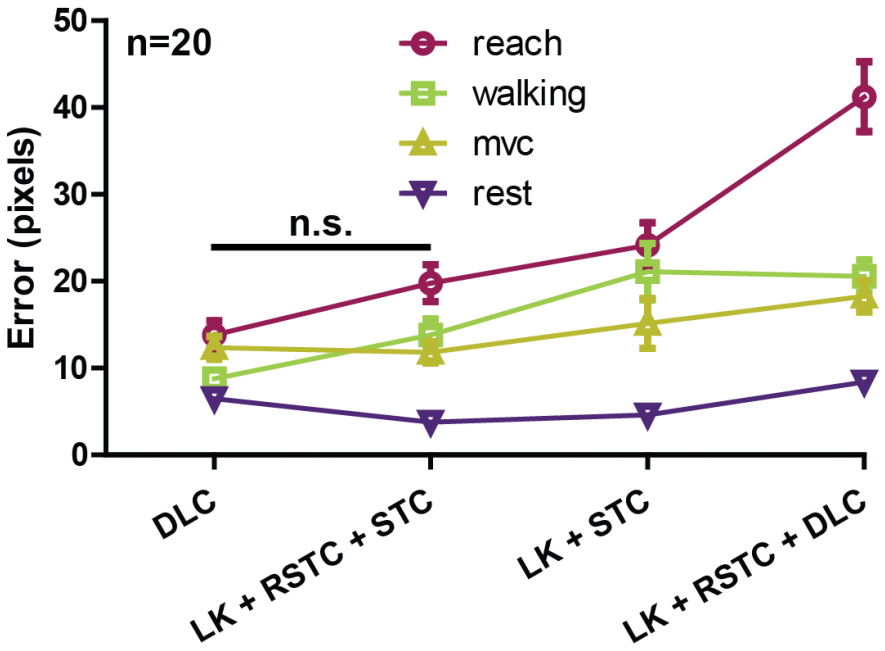


Figure 5-11: Accuracy of methods combining various existing LK and DLC tracking algorithms with proposed STC and RSTC methods of drifting correction.

times, finding their average distance to the original labels to be 37.9±2.6px. This error was high but expected due to the awkward deformation of skeletal tissue and fascia.

Figure 5-12 highlights how this variability, as well as the errors due to the various proposed algorithms from Figure 5-11 can partly be guided by the points selected as ground truth.

When tracking videos these videos, the points jittered more regularly when using DLC than using LK with RSTC. A few jitters seen were due to tracking outliers (described in Section 4.6) which can be mostly eliminated by adding more tracking information. The more consistent jitter likely came from the non-temporal tracking of CNN trackers that estimate the point location(s) at every frame independently of the frames right before or after it. LK, on the other hand, exploits spatial and temporal information by searching a point's location in the prior frame to determine its movement, leading to a smoother path. Figure 5-13 gives an example of this difference.

This Figure presents the only real caveat to using DLC. Though very accurate and

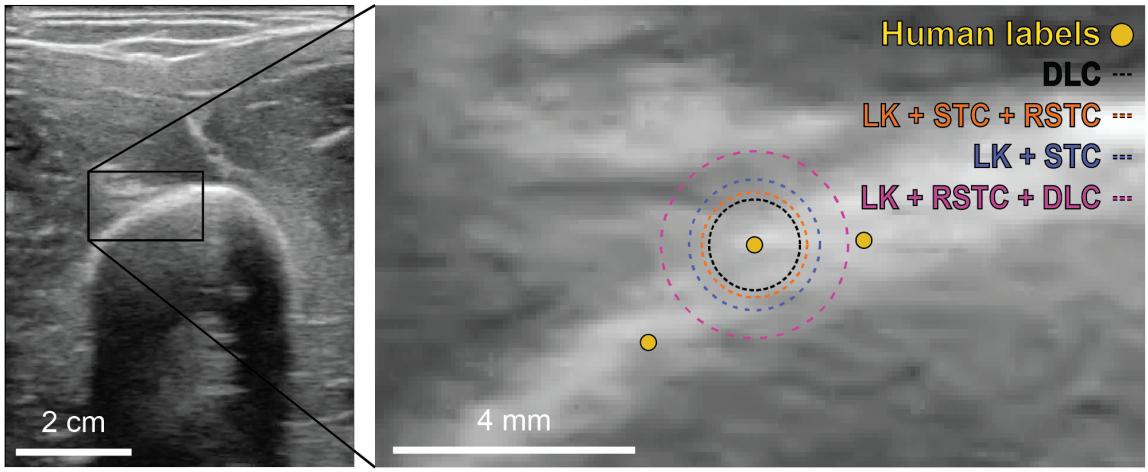


Figure 5-12: Highlights how variability of human labeling around ambiguous features (yellow points) in US can affect tracking and respective error values (dotted-line circles).

robust to video variability, DLC has tracking jitter, making it an unattractive choice if subtle or high frequency movements are of interest. In the context of this thesis, this means LK with these correction methods was used to further explore relationships between US and these tremors despite DLC having a slightly lower error.

5.4 Tremor in US

After establishing that using LK with RSTC and STC was the appropriate approach for tracking points in US for tremor detection, the points labeled in US videos of reaching tasks were tracked. First, the videos of Group 1 subjects each with different tracking templates were tracked, differentiating between the type of tissue where the points lied. The points tracked in Group 1 generally varied locations in the bone, muscle tissue, fascia, or adipose, as illustrated in Figure 1-1. After these points were tracked, and their x and y tracking paths were filtered and normalized as described in Section 3.4, and the root mean squared error (RMSE) between these filtered path signals and the filtered accelerometry signals was calculated to determine similarity to the tremor signature.

This process was repeated for the videos of Group 2 subjects, which all were

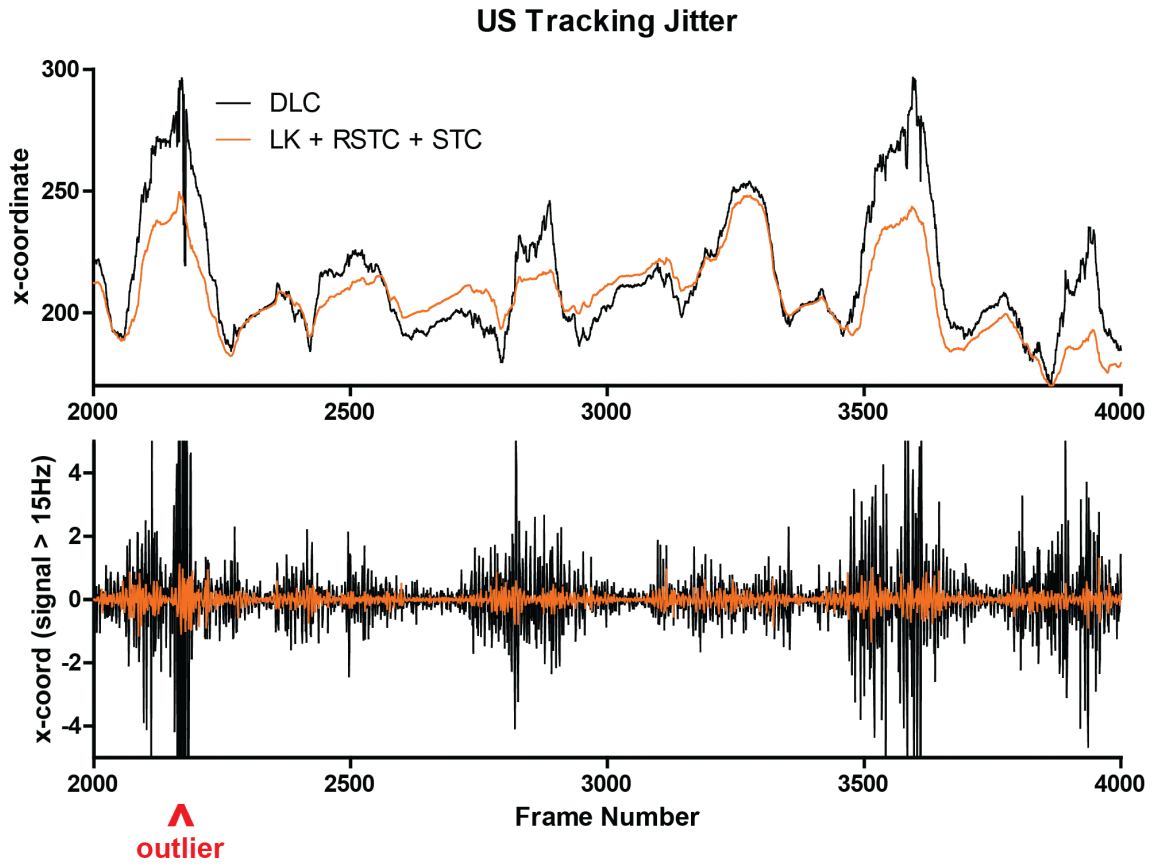


Figure 5-13: The upper plot shows an example point's x-coordinate as tracking occurs, both with small jitters throughout and a large tracking jump from an outlier at around frame 2200. The lower plot shows the resultant signal after bandpassing it above 15Hz with higher overall movement in DLC tracking.

labeled with the same 3 points on the surface of the bone. Figure 5-14 displays how both the type of tissue tracked and the coordinate of interest correlated to the "ground-truth" tremor signal from the biceps accelerometer, ensuring that the tremor signal from the tricep accelerometer produced the same results.

There are significant conclusions to draw from this Figure. For videos in Group 1, the smallest error among all tissue types was that of bone, hinting that the bone's movement during reaching presented the best indicator of tremors detection (ANOVA $F = 7.607$, $p < 0.001$). Additionally, the error in the y-coordinate was significantly lower than that of the x-coordinate ($t = 5.237$, $p < 0.001$), indicating that vertical movement in the image was a better indicator of tremor than horizontal movement. For videos in Group 2, the error in the y-coordinate was similarly lower than the

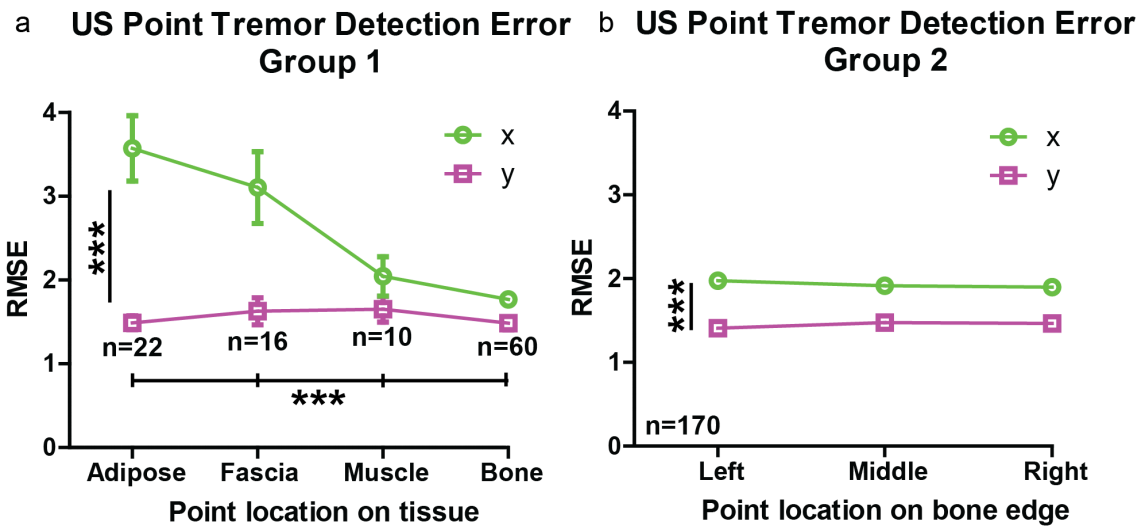


Figure 5-14: (a) This displays the RMSE between the tremor signal and the filtered US signal, showing a potential correlation to tissue rigidity. (b) This displays the RMSE between the tremor signal and the filtered US signal, showing a potential correlation to US axis movement.

x-coordinate's error ($t = 8.198$, $p << 0.001$). This figure also illustrates the RMSE values for all points on the bone were similar despite the differences in labeling processes for Groups 1 and 2.

Chapter 6

Conclusions & Future Work

This chapter will present the key takeaways and insights learned throughout this research process, including those from tremor and ultrasound analysis alone, as well as interpretations of their combined results in the context of physiologic tremors.

6.1 Tremor Characterization

When it comes to methods to decrease the magnitude of PTs, visual cues by far have the most impact compared to auditory or other sensory cues. Section 5.1 highlights that during the biofeedback trials, subjects were almost immediately able to change their movement in order to minimize their tremors. One subject in Group 2 described that to do this, they performed their reaching movements by focusing on shoulder rotation using their shoulder, back, and chest muscles as opposed to elbow rotation using their bicep and tricep muscles. Unfortunately, sensors were placed on the upper arm muscles, so no analysis of the others was done. Future work could involve attaching Trigno Avanti Sensors (or equivalent) on the deltoid, latissimus, trapezius, and infraspinatus muscles as these muscles can be measured by surface EMG and are central to the rotation of the shoulder joint. These signals could present a more holistic map of the muscles activated during reaching movements, and perhaps lead to a fuller understanding for the conditions required to activate a tremor.

The second insight is that although Tremor Proportion in Movement (TPM) was

not found to be directly linked to a specific muscle's activation, tremors occurred most frequently in retraction as the tricep relaxed. Additionally, after the trials in which subjects were asked to maximize their tremors, all of them described the most efficient way for them to do so was to fully contract their arm, a clear signature in the EMG. [1] states that PTs occur due to normal muscle activation, but the results from this study demonstrate a further point that while activation is related, it is the specifically the change in activation from a controlled state that provokes them.

Much of the analysis of physiological signals resulted in non-significant results, but can be areas of future work. For example, tremors have been found to be related to fatigue [55] and in this study, subjects in Group 1 were tasked with performing full-arm a set of 12 maximum voluntary contractions (MVCs) for 6 seconds each. No significant difference in TPM was found in movements before and after the MVCs, but this trial might have not been sufficient to cause fatigue in the first place. Another factor hypothesized to have a correlation to TPM was heart rate, as PTs have also been linked to excitement or anxiety [1] - again, no correlation was found. If fatigue or heart rate were to be investigated further, a high-intensity workout targeting the arm muscles could be performed before and after a series of slow movements, measuring TPM accordingly. Other factors considered were spine curvature and breathing rate, but no relationships were found.

6.2 Ultrasound Analysis

US is an inexpensive and non-invasive tool to visualize skeletal muscle during movement, but tracking features in B-mode footage presented many issues including point drifting. In this thesis, drifting was shown to be a recurring challenge when tracking in US, and various correction algorithms to minimize this phenomenon were proposed. It was shown that using Lucas-Kanade (LK) for this application resulted in the lowest tracking error of the frame-to-frame trackers explored, and applying the correction algorithms described only further improved accuracy.

It was shown that using DeepLabCut (DLC) outperformed all non-modified frame-

to-frame trackers, not only being minimally sensitive to the type of movement but also generally resulting in lower error. The only caveat against DLC was the random tracking uncertainty that can sometimes resulted in tracking jitter.

In conclusion, DLC was the recommended approach when accuracy and robustness are the preferred qualities in tracking; however, if tracking the fine movements of a feature is more important, applying the proposed correction algorithms to LK was recommended. For this thesis, LK with these correction methods was used to further explore relationships between US and PTs.

6.3 Tremor Characteristics in Ultrasound

After the appropriate filtering of US and accelerometer signals, it was found that the points tracked on the bone had the most similar tremor signatures to tremors detected on the skin surface when compared to other tissues. It was also found that the filtered y-coordinate of the points tracked in the US footage had more similar tremor signatures than the x-coordinate of the points. The y-coordinate points into the probe, meaning most tissue vibrations at the frequency of PTs occurred perpendicular to the sagittal plane of the body.

One potential interpretation of this could be that the detected tremors originated from the joint socket and onto the bone, propagated out into the muscle where their magnitude is dampened, and reached the adipose and skin layers where they were detected by the accelerometers. However, the interactions between the shoulder and elbow joints were not imaged nor studied in the experiment, so it is challenging to confirm the impact of this had. Another interpretation for these results is that the tremors originated from the subtle but controlled contractions in the biceps and triceps muscles, causing other skeletal tissues like the brachialis to deform around the humerus, then appearing as the origin of the tremor from the perspective of the US.

6.4 Summary

To summarize these points:

1. Visual stimuli (biofeedback) is an efficient way to control one's physiologic tremors.

2. In reaching movements, arm retraction leads to significantly higher tremor signatures than arm extension.

3. In reaching movements, it is the change in muscle activation that leads to the highest magnitude tremors.

4. In reaching movements, the bone's movement imaged in US has more similar tremor signature to that seen in accelerometry than that of any other tissue.

Figure 6-1 roughly illustrates the intepretation of these results given the context of signals and images collected.

6.5 Future Work

There are many directions to explore for the future of this research.

To explore other Muscle Architecture Parameters (MAPs) than those observable for this thesis, a longitudinal view of the longest tricep and bicep heads could be observed via US. For subjects in Group 2, a few trials near the end were done while imaging the biceps longitudinally using a Telemed probe (Telemed Medical Systems, Milan, Italy) which was lighter and has a longer line of contact. Observing the muscle fibers in this manner can display other MAPs of interest like pennation angle, fiber lengths, and muscle thickness at multiple locations along the humerus, providing a different perspective on muscle activation.

As mentioned in Section 5.2, the question of whether or not tremors can be used as indicators of task expertise has yet to be determined. Future work in this space mainly involves continuing data collection with a narrowed scope of subject experience, and defining clear metrics on what distinguishes levels of expertise.

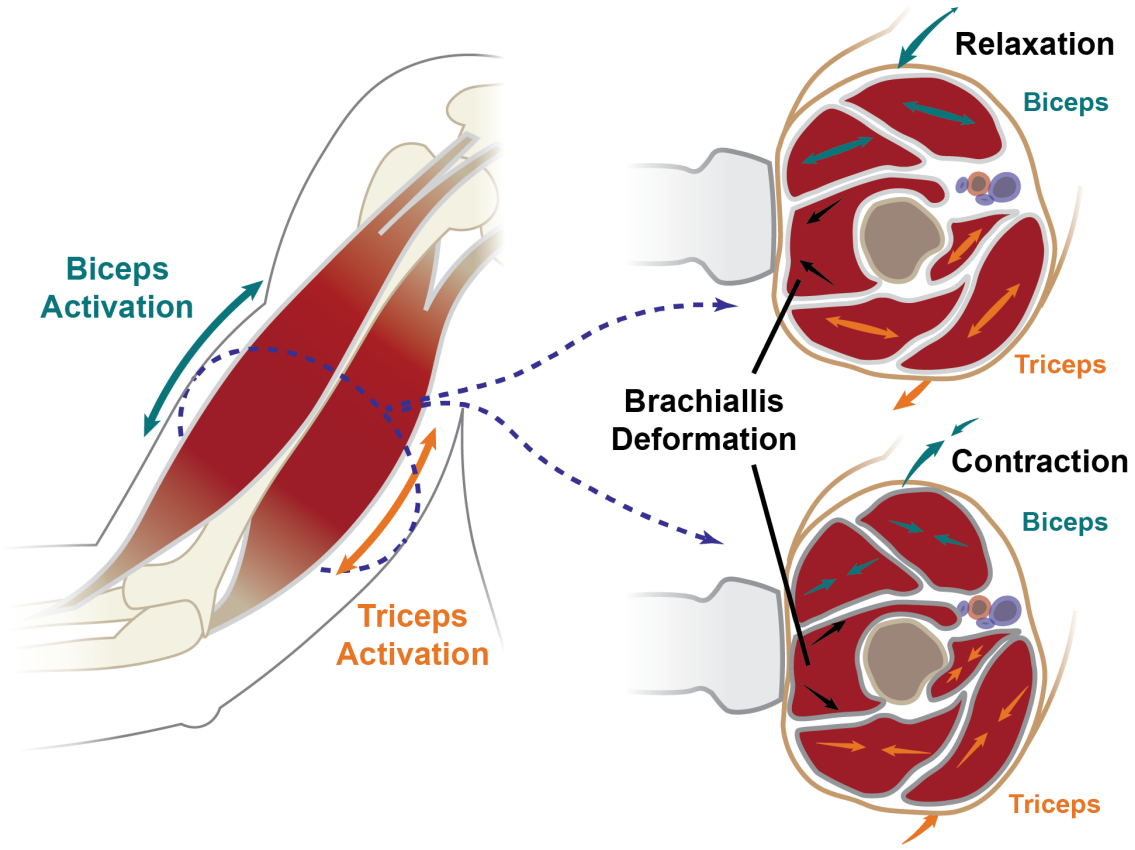


Figure 6-1: A potential interpretation of the biomechanics involved in the observed tremors. As the arm is lowered, the biceps and triceps muscles repeatedly (but subtly) contract and relax, leading to small changes in their shape. As the movement continues, the brachialis is deformed by the triceps and biceps muscles' contractions which, relative to the skin, appears as though the humerus bone is translating in the direction of the US probe.

The question of where these tremors originate is still not fully answered. The interpretation from Figure 6-1 only summarizes the observations captured via accelerometry, EMG, and US during the data collection for this thesis. However, muscle groups in the shoulder, back, and chest, as well as ball-socket interactions at the shoulder joint interface all likely contribute to the stability of these movements. Future studies will need to analyze these factors in more detail to expand the interpretation observed here.

Chapter 7

Code

7.1 Tremor Analysis

```
"""
A compilation of some of the functions used to analyze accelerometer, EMG,
    and Optitrack data.
Log objects made for different applications (EMGLog, AccelLog, MarkerLog,
    OptitrackLog)
"""

import numpy as np
import pandas as pd
from scipy.signal import hilbert, firwin, filtfilt

class signalLog:
    def __init__(self,sig,sr,start=0):
        self._t = np.arange(start,len(sig)/sr,1/sr)
        self._sig = sig
        self.sr = sr
        self.start = start
        self.end = self._t[-1]
    def t(self):
        return self._t
    def __call__(self):
        return self._sig

class signalLog3D(signalLog):
    def __init__(self,sig,sr,start=0,axis=None):
        super().__init__(sig,sr,start=start)
        self.axis = np.argmax(np.shape(self._sig)) if axis is None else axis

    def x(self):
        return self._sig[0]
    def y(self):
        return self._sig[1]
    def z(self):
        return self._sig[2]

class EMGLog(signalLog):
    def __init__(self,sig,sr,start=0):
        super().__init__(sig,sr,start=start)

    def bandpass(self, low, high, order=None):
        if order is None:
```

```
            order = int(self.sr/2) + 1
        filt_pts = firwin(order, (low, high), fs=self.sr,
            pass_zero='bandpass')
        proc_sig = filtfilt(filt_pts, 1, self._sig, axis=1)
        return EMGLog(self._t,proc_sig,self.sr)
    def envelope(self):
        proc_sig = np.abs(hilbert(self._sig, axis=1))
        return EMGLog(self._t,proc_sig,self.sr)

class AccelLog(signalLog3D):
    def __init__(self,sig,sr,start=0,axis=None):
        super().__init__(sig,sr,start=start,axis=axis)

    def magnitude(self):
        proc_sig = np.linalg.norm(self._sig, axis=self.axis)
        return AccelLog(proc_sig,self.sr,start=self.start)
    def bandpass(self, low, high, order=None):
        if order is None:
            order = int(self.sr/2) + 1
        filt_pts = firwin(order, (low, high), fs=self.sr,
            pass_zero='bandpass')
        proc_sig = filtfilt(filt_pts, 1, self._sig, axis=self.axis)
        return AccelLog(proc_sig,self.sr,start=self.start)
    def envelope(self):
        proc_sig = np.abs(hilbert(self._sig, axis=self.axis))
        return AccelLog(proc_sig,self.sr,start=self.start)

    def tremor_sig(self, low = 7, high = 14, type='max', exp=2):
        """
        type = 'max' or 'mag'
        smooth_dt = None (set to sr/2) or n (int), smooth bandpassed signal
            over n sample points
        Returns AccelLog object
        """
        if smooth_dt is None:
            smooth_dt = int(self.sr/2)
        start_end = np.logical_and(self._t > self.start, self._t < self.end)
        new_t = self._t[start_end]
        if type=='max':
            new_acc = smooth(np.max(self.bandpass(low,...
                high).envelope()(),axis=1),smooth_dt)[start_end]
        else:
            new_acc = smooth(self.magnitude().bandpass(low,...
                high).envelope()(),smooth_dt)[start_end]
        return AccelLog(new_acc**exp,self.sr,start=new_t[0])

class MarkerLog(signalLog3D):
    def __init__(self,sig,sr,name,start=0,axis=None):
        super().__init__(sig,sr,start=start,axis=axis)
        self.name = name

class OptitrackLog:
    def __init__(self,marker_list):
        self._t = marker_list[0].t()
        self.marker_list = marker_list

    def t(self):
        return self._t
    def get_markers(self,name):
        markers_with_name = [marker for marker in self.marker_list if
            marker.name == name]
        if len(markers_with_name) == 0:
            return 'No markers found'
        else:
            return markers_with_name

def find_TPM_time(accel_log):
    """
    Returns TP (Tremor Percent) and TPM (Tremor Percentage in Movement)
```

```
    """
    tremor_sig = accel_log.tremor_sig()
    tremor_edges = get_edges(accel_log.t(),tremor_sig(),threshold = 0.015**2)
    movement_edges = get_movement_edges(accel_log())
    combined_edges = combine_edges(tremor_edges,movement_edges)

    time_tremor_mov = sum([i[1]-i[0] for i in combined_edges])
    time_mov = sum([i[1]-i[0] for i in movement_edges])
    time_tremor_percent = time_tremor_mov/(accel_log.end-accel_log.start)*100
    moving_time_tremor_percent = time_tremor_mov/time_mov*100

    return time_tremor_percent, moving_time_tremor_percent

def find_EMG_slopes(accel_log,emg_log):
    """
    Returns TP (Tremor Percent) and TPM (Tremor Percentage in Movement)
    """
    tremor_sig = accel_log.tremor_sig()
    tremor_edges = get_edges(accel_log.t(),tremor_sig(),threshold = 0.015**2)
    new_emg = emg_log.envelope()()

    slopes = np.zeros(len(tremor_edges))
    for i,(t_init,t_end) in enumerate(tremor_edges):
        if t_end < emg_log.t()[-1]:
            ind_init = np.where(emg_log.t()>t_init)[0][0]
            ind_end = np.where(emg_log.t()>t_end)[0][0]
            xs = emg_log.t()[ind_init:ind_end]
            m,b = fit_line(xs,new_emg[ind_init:ind_end])
            slopes[i]=m

    return np.average(slopes)

def find_tremor_elbow_angles(ot_log, muscle_accel_logs):
    """
    muscle_accels is list of 3 AccelLog objects, one for each muscle
    elbow_chain
    """
    t_ot = ot_log.t()
    pinky = ot_log.get_markers('pinky')
    elbow = ot_log.get_markers('elbow')
    shoulder = ot_log.get_markers('shoulder')

    elbow_angles = get_elbow_angles(pinky,elbow,shoulder)
    sr_ot = 1/(t_ot[1]-t_ot[0])
    elbow_speeds = smooth(differentiate(elbow_angles,1/sr_ot),int(sr_ot/4))

    for muscle_log in muscle_accel_logs:
        tremor_sig = muscle_log.tremor_sig()
        tremor_edges = get_edges(muscle_log.t(),tremor_sig(),threshold =
            0.015**2)
        speed_area = 0
        t_total = 0
        for pk in tremor_edges:
            if t_ot[0] < pk[0] and t_ot[-1] > pk[1]:
                init,end = np.where(t_ot>pk[0])[0][0],
                    np.where(t_ot>pk[1])[0][0]
                t_temp = pk[1]-pk[0]
                area = integrate(elbow_speeds[init:end],1/sr_ot)
                speed_area += area
                t_total += t_temp

def find_cep_RMSE(tracked_csv,accel_log,npoints_per_frame,features_labeled):
    """
    tracked_csv looks like:
    pt1_x_LK   pt1_y_LK   pt2_x_LK, ...
    175.4887362  108.0416328 281.3327573
    ...
    features_labeled = ['humerus','bicep_muscle',...]
    """
    if len(features_labeled) != npoints_per_frame:
```

```
            return "List of muscles_labeled must be npoints long"

        d = {}
        for body_part in features_labeled:
            for coord in ['x','y']:
                d[f'{body_part}_{coord}'] = []
        sr_cep = 60
        s, e = accel_log.start, accel_log.end
        df_US_data = pd.read_csv(tracked_csv)
        tremor_sig = accel_log.tremor_sig()
        for body_part,pt in zip(features_labeled,range(1,npoints_per_frame+1)):

            LK_axs_x =
                signalLog(np.array(df_US_data[f'pt{pt}_x']),sr_cep,start=s)
            LK_axs_y =
                signalLog(np.array(df_US_data[f'pt{pt}_y']),sr_cep,start=s)

            LK_bp_x =
                smooth(LK_axs_x.bandpass(7,14).envelope()(),int(sr_cep/2))**2
            LK_bp_y =
                smooth(LK_axs_y.bandpass(7,14).envelope()(),int(sr_cep/2))**2
            LK_axs_env_x = signalLog(LK_bp_x,sr_cep,start=s)
            LK_axs_env_y = signalLog(LK_bp_y,sr_cep,start=s)

            bp_b_LK, LK_interp_x = interp_same_t(tremor_sig,LK_axs_env_x)
            _, LK_interp_y = interp_same_t(tremor_sig,LK_axs_env_y)
            norm_val = (np.average(LK_interp_x())+np.average(LK_interp_y()))/2

            bp_b_norm = bp_b_LK()/np.average(bp_b_LK())
            LK_norm_x = LK_interp_x()/norm_val
            LK_norm_y = LK_interp_y()/norm_val

            rmse_x = np.sqrt(np.average((bp_b_norm-LK_norm_x)**2))
            rmse_y = np.sqrt(np.average((bp_b_norm-LK_norm_y)**2))
            d[f'{body_part}_x'].append(rmse_x)
            d[f'{body_part}_y'].append(rmse_y)

        df = pd.DataFrame(data=d)
        df.to_csv('RMSE_values.csv')

    # HELPER FUNCTIONS
    # Accelerometry
    def get_movement_edges(accel_sig, thresh = 0.15):
        """
        Finds "edges" list from get_edges during which the deriv in acceleration
            signal is high enough to be considered movement.
        """
        t = accel_sig.t()
        sr = accel_sig.sr
        dt = 1/sr

        start_end = np.logical_and(t > accel_sig.start, t < accel_sig.end)
        xs,ys,zs = [smooth(i,sr) for i in
            [accel_sig.x()[start_end],accel_sig.y()[start_end],accel_sig.z()[start_end]]]
        dxs, dys, dzs = [differentiate(sig,dt) for sig in [xs,ys,zs]]
        movement = np.sum(np.abs([dxs,dys,dzs]),axis=0)
        movement_edges = get_edges(t,movement,threshold=thresh)
        return movement_edges

    def combine_edges(edges1,edges2):
        """
        Takes 2 "edges" lists from get_edges, finds their overlaps
        """
        new_edges = []
        for a in edges1:
            for b in edges2:
                if a[1] < b[0]:
                    break
                if a[0] > b[0] and a[1] < b[1]:
                    new_edges.append((a[0],a[1]))
```

```
                break
            if a[1] > b[0] and a[1] < b[1]:
                new_edges.append((b[0],a[1]))
                break
            if a[0] < b[0] and a[1] > b[1]:
                new_edges.append((b[0],b[1]))
            if a[0] > b[0] and a[0] < b[1]:
                new_edges.append((a[0],b[1]))
    return new_edges

def get_edges(t,sig,threshold):
    """
    Returns list of tuples, each tuple having a (start,end) of when
    a signal surpasses threshold value, and then returns under it
    """
    edges = []
    marked = False
    for i in range(len(sig)):
        if sig[i] > threshold and not marked:
            marked = True
            point1 = t[i]

        if sig[i] < threshold and marked:
            marked = False
            point2 = t[i]
            edges.append((point1,point2))
    return edges

# Optitrack
def get_elbow_angles(pinky, elbow, shoulder):
    """
    pinky, elbow, and shoulder are MarkerLog objects
    returns elbow_angles array
    """
    t_ot = pinky.t()

    pinkyx,not_nans = ot_interp(t_ot,pinky.x()); pinkyx = pinkyx[not_nans]
    pinkyy,not_nans = ot_interp(t_ot,pinky.y()); pinkyy = pinkyy[not_nans]
    pinkyz,not_nans = ot_interp(t_ot,pinky.z()); pinkyz = pinkyz[not_nans]
    elbowx,not_nans = ot_interp(t_ot,elbow.x()); elbowx = elbowx[not_nans]
    elbowy,not_nans = ot_interp(t_ot,elbow.y()); elbowy = elbowy[not_nans]
    elbowz,not_nans = ot_interp(t_ot,elbow.z()); elbowz = elbowz[not_nans]
    shoulderx,not_nans = ot_interp(t_ot,shoulder.x()); shoulderx =
        shoulderx[not_nans]
    shouldery,not_nans = ot_interp(t_ot,shoulder.y()); shouldery =
        shouldery[not_nans]
    shoulderz,not_nans = ot_interp(t_ot,shoulder.z()); shoulderz =
        shoulderz[not_nans]

    v1 = np.array([pinkyx - elbowx, pinkyy - elbowy, pinkyz - elbowz])
    v2 = np.array([shoulderx - elbowx, shouldery - elbowy, shoulderz -
        elbowz])

    elbow_angles , not_nans =
        ot_interp(t_ot,np.rad2deg(angle_between3d(v1,v2)))
    return elbow_angles[not_nans]

def ot_interp(t,x,min_jump=2,min_time=2):
    """
    Takes a time array t and 1D x array from MarkerLog object, finds any
        missing points or discontinous jumps greater than min_jump mm
    that last less than min_time, and linearly interpolates between them.
    Returns the new x (array)
    """
    # Determining if initial few points should be high or low values
    dt = t[1]-t[0]
    next_values = int(min_time//dt)
    init_mean = np.mean(x[:next_values])
    if abs(init_mean - x[0]) > abs(init_mean - x[1]):
```

```
            x_ = [x[1]]
        elif pd.isna(init_mean):
            x_ = [0]
        else:
            x_ = [x[0]]

        # Interpolating
        currently_interp = False
        curr_i = 0
        i = 1
        while i < len(x):
            if pd.isna(x[i]):
                x[i] = 0
            if currently_interp:
                k = i-curr_i+1
                if k == j+1:
                    currently_interp = False
                x_.append(x[curr_i-1] + k*slope)
                i += 1

            # if the next point is too far away, check the next min_time seconds
                for any point within feasible range
            elif abs(x[i] - x[i-1]) > min_jump:
                vals_left_to_check = min(next_values,len(x)-i)
                if vals_left_to_check == 1:
                    x_.append(x[i-1])
                    i+=1
                for j in range(1,vals_left_to_check):
                    # if point is found, interpolate between i'th and i+j'th point
                    if abs(x[i+j] - x[i-1]) < j*min_jump/20:
                        currently_interp = True
                        slope = (x[i+j]-x[i-1])/(j+1)
                        curr_i = i
                        break
                    # if no point is found, add the previous point
                    elif j == vals_left_to_check-1:
                        x_.append(x[i-1])
                        i+=1

            # if point is ok
            else:
                x_.append(x[i])
                i+=1

        x_ = np.array(x_)
        not_nans = np.logical_not(pd.isna(x_))

        return x_, not_nans

    # General
    def smooth(sig,n):
        """
        Takes a signal and takes real-time moving average, smoothing by n sample
            points.
        Returns new signal with the same length as sig
        """
        new_sig = np.zeros_like(sig())
        assert n > 1

        back = n // 2
        for i in range(len(sig())):
            start = max(i - back,0)
            end = min(i + back + 1 if n % 2 != 0 else i + back, len(sig()))
            new_sig[i] = np.average(sig()[start:end])

        return new_sig

    def interp_same_t(sig1,sig2):
        """
        Takes 2 signalLog objects and cross-correlates them:
        """
        t_init = max([sig1.t()[0],sig2.t()[0]])
```

```
    t_end = min([sig1.t()[-1],sig2.t()[-1]])
    if sig1.sr > sig2.sr:
        x_base = sig1
        sr = sig1.sr
    else:
        x_base = sig2
        sr = sig2.sr
    t_vals = x_base.t[np.logical_and(x_base.t>t_init,x_base.t<t_end)]

    x1_new = np.interp(t_vals, sig1.t(), sig1())
    x2_new = np.interp(t_vals, sig2.t(), sig2())
    return
        signalLog(x1_new,sr,t0=t_vals[0]),signalLog(x2_new,sr,t0=t_vals[0])

def differentiate(sig,dt):
    deriv = [(sig[i+1]-sig[i])/dt for i in range(len(sig)-1)]
    deriv.extend([0]) # keeps len(sig) = len(deriv(sig))
    return np.array(deriv)

def integrate(sig,dt):
    area = 0
    for i in range(len(sig)-1):
        area += dt*(sig[i]+sig[i+1])/2
    return area

def fit_line(x,y):
    """
    Input x and y, returns best fit line slope and y intercept
    """
    A = np.vstack([x,np.ones(len(x))]).T
    m,b = np.linalg.lstsq(A,y,rcond=None)[0]
    return m,b

def angle_between3d(v1,v2):
    xa,ya,za = v1
    xb,yb,zb = v2
    dir = 1
    if np.array_equal(za,np.zeros_like(xa)) and
        np.array_equal(zb,np.zeros_like(xb)):
        if np.arctan2(ya,xa) < np.arctan2(yb,xb):
            dir = -1

    return dir*np.arccos((xa*xb + ya*yb +
        za*zb)/(np.sqrt(xa**2+ya**2+za**2)*np.sqrt(xb**2+yb**2+zb**2)))
```

7.2 Ultrasound Analysis

```
"""
A compilation of some of the functions used to track, perform analysis on,
    and compare methods on videos.
This code has been modified to apply to one video file at a time.
    Preprocessing includes:
1. Creating a Log object for a specific video file (some_video_usb_001.mp4)
2. From that video, screenshot a frame, highlight the points you want to
    track, and save as tracking_template.png
3. Label a select number of frames, 1 per ~2000 frames is recommended, but
    depends highly on the type of movement observed.
3. Run any tracking algorithm!
"""

import os
import random
import subprocess
import cv2 as cv
import numpy as np
import pandas as pd
from math import sqrt, atan2,cos,sin, acos
```

```
from skimage.restoration import denoise_wavelet

class Log:
    def __init__(self, video_file, tracker_type_input = 'LK', type =
        'original'):
        """
        video_file = 'your_vidfile_002.mp4'
        Will choose Lucas Kanade (LK) Tracker automatically unless specified
            otherwise.
        Other tracker options: 'LK','BOOSTING', 'MIL','KCF', 'TLD',
            'MEDIANFLOW', 'GOTURN', 'MOSSE', 'CSRT'
        type (str) parameter only useful to ensure cep.Log.name is consistent
            example -  'original' - cep.Log(None,
                video_file=us_b_001.mp4).name = b_001
                        'backward' - cep.Log(None,
                            video_file=usb_b_001.mp4).name = b_001_bwd
                        'mirrored' - cep.Log(None,
                            video_file=usb_b_001.mp4).name = b_001_mir
                        'dlc'      -  cep.Log(None,
                            video_file=usb_b_001.mp4).name = b_001_dlc
        """

        assert os.path.splitext(video_file)[-1] == '.mp4'
        data = cv.VideoCapture(video_file)
        if type == 'backward':
            self.name =
                os.path.split(video_file)[-1].split('.')[0][4:]+'_bwd'
        elif type == 'mirrored':
            base = os.path.split(video_file)[-1].split('.')[0].split('_')
            init_frame,final_frame = base[3][5:],base[5]
            self.name = f'{base[1]}_{base[2]}_mir{init_frame}_{final_frame}'
        # only used in automate_DLC_labeling_opr02() of dlc_wobble
        elif type == 'dlc':
            self.name = os.path.split(video_file)[-1].split('.')[0]
        else:
            self.name = os.path.splitext(video_file)[-1].split('.')[0][3:]
        # dat,tim = os.path.splitext(video_file)[0].split('\\')[-1].split()
        frames = data.get(cv.CAP_PROP_FRAME_COUNT)
        self.sr = int(data.get(cv.CAP_PROP_FPS))
        self.desired_times = [i/self.sr for i in range(int(frames))]
        self.real_times = None
        self.start = 0
        self.end = frames / self.sr
        self.vid_file = video_file
        self.tracker_type = tracker_type_input
        self.video = None

    def set_tracker_type(self,tracker):
        self.tracker_type = tracker

    def tracker(self):
        if self.tracker_type == 'BOOSTING':
            tracker = cv.legacy.TrackerBoosting_create()
        if self.tracker_type == 'MIL':
            tracker = cv.legacy.TrackerMIL_create()
        if self.tracker_type == 'KCF':
            tracker = cv.legacy.TrackerKCF_create()
        if self.tracker_type == 'TLD':
            tracker = cv.legacy.TrackerTLD_create()
        if self.tracker_type == 'MEDIANFLOW':
            tracker = cv.legacy.TrackerMedianFlow_create()
        if self.tracker_type == 'GOTURN':
            tracker = cv.TrackerGOTURN_create()
        if self.tracker_type == 'MOSSE':
            tracker = cv.legacy.TrackerMOSSE_create()
        if self.tracker_type == "CSRT":
            tracker = cv.legacy.TrackerCSRT_create()
        return tracker
```

```
def update_tracker(self, video, denoised, old_pts, tracker, slength,
    multi=None, return_boxes = False):
    """
    Function to be used in 'while' loop of openCV
    Args:
        video - from cv.VideoCapture
        old_frame - from video.read()
        old_pts - nx2 points
        tracker - from ['LK', 'BOOSTING', 'MIL','KCF', 'TLD',
            'MEDIANFLOW', 'GOTURN', 'MOSSE', 'CSRT']
        multiTracker - None if tracker in ['LK'], else multiTracker
            object from cv.legacy.MultiTracker_create()
    Returns:
        new_frame - next frame
        new_pts - nx2 updated tracked points
    """
    success, new_frame = video.read()
    if not success:
        video.release()
        return None

    # old_gray = cv.cvtColor(old_frame, cv.COLOR_BGR2GRAY)
    new_gray = cv.cvtColor(new_frame, cv.COLOR_BGR2GRAY)

    denoised_gray = cv.cvtColor(denoised, cv.COLOR_BGR2GRAY)
    denoised_new = self.denoise_frame(new_frame)
    denoised_new_gray = cv.cvtColor(denoised_new, cv.COLOR_BGR2GRAY)

    new_pts = np.zeros((len(old_pts),2))
    H,W = self.h(),self.w()

    if tracker == 'LK':
        edge_offset = 0.5
        p0 = old_pts.reshape((len(old_pts),1,2))
        lk_params = dict(
            winSize=(slength, slength),
            maxLevel=2,
            criteria=(cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10,
                0.03),
        )

        p1, st, err = cv.calcOpticalFlowPyrLK(denoised_gray,
            denoised_new_gray, p0, None, **lk_params)

        new_pts = p1[:,0,:]
        old_pts = p0[:,0,:]

        for i, (new, old) in enumerate(zip(new_pts, old_pts)):
            new_x = old_pts[i][0] + new_pts[i][0]-old_pts[i][0]
            if new_x > W - edge_offset:
                new_x = min(new_x,W-edge_offset)
            elif new_x < edge_offset:
                new_x = max(new_x,edge_offset)

            new_y = old_pts[i][1] + new_pts[i][1]-old_pts[i][1]
            if new_y > H - edge_offset:
                new_y = min(new_y,H-edge_offset)
            elif new_y < edge_offset:
                new_y = max(new_y,edge_offset)

            new_pts[i] = np.array([new_x,new_y])

        if return_boxes:
            new_boxes = Boxes(boxes=[])
            for i in range(len(old_pts)):
                temp_box = Box((new_pts[i][0],new_pts[i][1],0,0))
                new_boxes.add(temp_box.centered_box(slength=slength))

    elif tracker in ['BOOSTING', 'MIL','KCF', 'TLD', 'MEDIANFLOW',
        'GOTURN', 'MOSSE', 'CSRT']:
        if tracker == 'KCF':
```

```
            success, boxes = multi.update(denoised_new)
        else:
            success, boxes = multi.update(denoised_new_gray)
        new_boxes = Boxes([Box(box).centered_box(slength=slength) for
            box in boxes])
        for i, newbox in enumerate(new_boxes.get()):
            a, b = newbox.center()
            new_pts[i] = np.array([a,b])

    if return_boxes:
        return new_frame, new_pts, new_boxes

    return new_frame, new_pts

def method_errors(self, nframes_reset, df, method, slength = 100, show =
    True, save = False, backward = False):
    """
    nframes_reset (int) 1-4, how many correction frames to include from
        training to help track
    df (DataFrame), from tracking_points.csv file, to get training data
    method (str), from ['LK', 'BOOSTING', 'MIL','KCF', 'TLD',
        'MEDIANFLOW', 'GOTURN', 'MOSSE', 'CSRT']
    """

    assert nframes_reset <= 4

    reset = False
    nframes = self.frame_count()
    all_points = np.zeros((nframes,3,2))
    step = nframes//9
    if not backward:
        counter = step - 1
    else:
        offset = nframes - step*9
        counter = step + offset - 1

    frame_ind = 0
    if nframes_reset == 0:
        frames = [1]
    if nframes_reset == 1:
        frames = [1, 5]
    if nframes_reset == 2:
        frames = [1, 4, 7]
    if nframes_reset == 3:
        frames = [1, 3, 5, 7]
    if nframes_reset == 4:
        frames = [1, 2, 4, 6, 8]

    if not backward:
        frame_nums = [i*step for i in frames]
    else:
        frame_nums = [i*step+offset for i in frames]

    video = cv.VideoCapture(self.vid_file)
    video.set(1,counter)
    success, old_frame = video.read()

    colors = self.get_colors(3,type=('Multi',None),randomized=False)
    colors_mask = [(255,10,10),(10,255,25),(10,10,235)]

    p0_ = []

    # GETTING ALL TESTING DATA
    boxes = Boxes(boxes=[])
    comparing_points = {}
    testing_points = {}

    color_ind = 0
    for f in range(1,9):
        comparing_points[f] = []
        if f in frames:
            testing_points[f] = []
```

```
    for i,pt in enumerate(df.Points):
        if pt[4:] in self.name:
            if f == frames[0]:
                if not backward:
                    init_point = (df['f1_x'][i],df['f1_y'][i])
                else:
                    init_point = (df['f8_x'][i],df['f8_y'][i])

                p0_.append(np.array([np.array([init_point[0],...
                           init_point[1]],dtype=np.float32)]))
                bbox = Box((init_point[0],...
                           init_point[1],0,0)).centered_box(slength=slength)
                boxes.add(bbox)
                cv.circle(old_frame,(round(init_point[0]),...
                           round(init_point[1])),4,colors[color_ind],-1)
                if show:
                    cv.imshow(f"{method} {slength}x{slength}
                        Img",old_frame)
                color_ind+=1
            if f in frames:
                if not backward:
                    testing_points[f].append((df[f'f{f}_x'][i],df[f'f{f}_y'][i]))
                else:
                    testing_points[f].append((df[f'f{9-f}_x'][i],df[f'f{9-f}_y'][i]))

            comparing_points[f].append((df[f'f{f}_x'][i],df[f'f{f}_y'][i]))

p0 = np.array(p0_)
old_pts = p0[:,0,:]

mask = np.zeros_like(old_frame)

# Making multitracker object
# blurred = cv.GaussianBlur(old_frame,(21,21),0)
denoised_asint = self.denoise_frame(old_frame)
multiTracker = cv.legacy.MultiTracker_create()
for bbox in boxes.get():
    multiTracker.add(self.tracker(), denoised_asint, bbox.get_int())

# to save
if save:
    cep_folder = os.path.split(self.vid_file)[0]
    opencv_folder = 'opencv'
    if not backward:
        tracker_folder = f'{method}_{slength}x{slength}'
    else:
        tracker_folder = f'{method}_{slength}x{slength}_bwd'
    if not os.path.exists(os.path.join(cep_folder,opencv_folder)):
        os.mkdir(os.path.join(cep_folder,opencv_folder))
    if not
        os.path.exists(os.path.join(cep_folder,opencv_folder,tracker_folder)):
        os.mkdir(os.path.join(cep_folder,opencv_folder,tracker_folder))
    saved_video =
        os.path.join(cep_folder,opencv_folder,tracker_folder,...
            f'us_{self.name}_{method}_{slength}x{slength}_{nframes_reset}reset.avi')
    out_vid = cv.VideoWriter(saved_video,
        cv.VideoWriter_fourcc(*"MJPG"), 60, (self.w(), self.h()))

# tracking
while True:
    out = self.update_tracker(video, denoised_asint, old_pts,
        method, slength, multi=multiTracker)
    if out is None:
        break

    new_frame, new_pts = out

    old_frame = new_frame.copy()
    denoised_asint = self.denoise_frame(old_frame)
    counter += 1
```

```
            # reseetting points at correcting frames
            if counter == frame_nums[frame_ind]:
                reset = True
            for i in range(len(new_pts)):
                a, b = new_pts[i]
                cv.circle(new_frame, (round(a), round(b)), 5, colors[i], -1)
                cv.rectangle(new_frame,(round(a-slength//2),round(b-slength//2)),
                    (round(a+slength//2),round(b+slength//2)), colors[i], 1,
                    1)
                if reset:
                    actual_points = testing_points[frames[frame_ind]][i]
                    new_pts[i] = np.array([actual_points[0],actual_points[1]])
                    # set reset back to False, and look at next frame_ind at
                        which to reset
                    if i == len(new_pts) -1:
                        reset = False
                        if frame_ind + 1 < len(frames):
                            frame_ind += 1

            all_points[counter] = new_pts
            old_pts = new_pts.copy()

            # img for display
            img = cv.add(new_frame, mask)
            cv.putText(img, f"Next reset at Frame {frame_nums[frame_ind]}",
                (80,535), cv.FONT_HERSHEY_SIMPLEX, 0.75, (255,255,255), 2)
            cv.putText(img, f"{self.name}, {nframes_reset} Corr Frame",
                (80,560), cv.FONT_HERSHEY_SIMPLEX, 0.75, (255,255,255), 2)
            cv.putText(img, f"{round(counter/nframes*100)}% Complete",
                (130,585), cv.FONT_HERSHEY_SIMPLEX, 0.75, (255,255,255), 2)
            if show:
                cv.imshow(f"{method} {slength}x{slength} Img", img)
            if save:
                out_vid.write(img)

            # If ESC pressed
            k = cv.waitKey(25) & 0xFF
            if k == 27:
                break

        if save:
            out_vid.release()
            subprocess.call(['ffmpeg', '-i', saved_video,
                f'{saved_video[:-4]}.mp4'])
            os.remove(saved_video)
        if show:
            cv.destroyWindow(f"{method} {slength}x{slength} Img")

        return all_points, comparing_points, testing_points

    def method_errors_combined(self, method, frame_ind, init_points, step =
        200, slength=100, show = True):
        """
        Uses 'method' tracker to track from frame_ind to frame_ind + step
            starting from init_points

        method (str), from ['LK', 'BOOSTING', 'MIL','KCF', 'TLD',
            'MEDIANFLOW', 'GOTURN', 'MOSSE', 'CSRT']

        """

        nframes = self.frame_count()
        all_points = np.zeros((nframes,3,2))
        if 'bwd' in self.name:
            counter = frame_ind - 1
        else:
            counter = frame_ind

        video = cv.VideoCapture(self.vid_file)
        video.set(1,counter)
        success, old_frame = video.read()
```

```
colors = self.get_colors(3,type=('Multi',None),randomized=False)
title = f"{method} {slength}x{slength} Img"

p0_ = []

# GETTING ALL TESTING DATA
boxes = Boxes(boxes=[])

color_ind = 0

for (x,y) in init_points:
    p0_.append(np.array([np.array([x,y],dtype=np.float32)]))
    bbox = Box((x,y,0,0)).centered_box(slength=slength)
    boxes.add(bbox)
    cv.circle(old_frame,(round(x),round(y)),4,colors[color_ind],-1)
    if show:
        cv.imshow(title,old_frame)
    color_ind+=1

p0 = np.array(p0_)
old_pts = p0[:,0,:]
all_points[counter] = old_pts

mask = np.zeros_like(old_frame)

# Making multitracker object
denoised_asint = self.denoise_frame(old_frame)
multiTracker = cv.legacy.MultiTracker_create()
for bbox in boxes.get():
    multiTracker.add(self.tracker(), denoised_asint, bbox.get_int())

# tracking
while counter < frame_ind + step:
    out = self.update_tracker(video, denoised_asint, old_pts,
        method, slength, multi=multiTracker)

    # if not success
    if out is None:
        break

    new_frame, new_pts = out

    old_frame = new_frame.copy()
    denoised_asint = self.denoise_frame(old_frame)
    counter += 1

    for i in range(len(new_pts)):
        a, b = new_pts[i]
        # c, d = old_pts[i]
        # mask = cv.line(mask, (round(a), round(b)), (round(c),
            round(d)), colors_mask[i], 1)
        cv.circle(new_frame, (round(a), round(b)), 5, colors[i], -1)
        cv.rectangle(new_frame,(round(a-slength//2),round(b-slength//2)),
            (round(a+slength//2),round(b+slength//2)), colors[i], 1,
            1)

    all_points[counter] = new_pts
    old_pts = new_pts.copy()

    # img for display
    img = cv.add(new_frame, mask)
    cv.putText(img, f"{self.name}", (150,560),
        cv.FONT_HERSHEY_SIMPLEX, 0.75, (255,255,255), 2)
    if 'bwd' in self.name:
        cv.putText(img, f"{nframes - counter} -> {nframes - frame_ind
            - step}", (130,585), cv.FONT_HERSHEY_SIMPLEX, 0.75,
            (255,255,255), 2)
    else:
        cv.putText(img, f"{counter} -> {step + frame_ind}",
            (130,585), cv.FONT_HERSHEY_SIMPLEX, 0.75, (255,255,255),
            2)
    if show:
```

```
            cv.imshow(title, img)

        # If ESC pressed
        k = cv.waitKey(25) & 0xFF
        if k == 27:
            break

    if show:
        cv.destroyWindow(title)

    return all_points

def follow_points(self, all_points_list, start_ind = None, save = False):
    """
    Function that inputs an f x n x 2 matrix (first output of
        methods_error) where f is number of frames,
    n is the number of points per frame, and 2 is the x and y location
        of each point.

    Displays these points in the video.
    """
    nframes = self.frame_count()
    assert all_points_list[0].shape[0] == nframes

    colors = self.get_colors(3,type=('Multi',None),randomized=True)

    counter = 0
    video = cv.VideoCapture(self.vid_file)
    if start_ind is not None:
        counter = start_ind - 1
        video.set(1,counter)
    success, frame = video.read()
    if save:
        cep_folder = os.path.split(self.vid_file)[0]
        saved_video =
            os.path.join(cep_folder,f'us_final_{self.name}.avi')
        out_vid = cv.VideoWriter(saved_video,
            cv.VideoWriter_fourcc(*"MJPG"), 60, (self.w(), self.h()))

    while True:
        success, frame = video.read()
        if not success:
            break
        counter += 1
        for p,points in enumerate(all_points_list):
            pts = points[counter]
            for i,(x,y) in enumerate(pts):
                cv.circle(frame, (round(x), round(y)), 5, colors[i], -1)
                if p == 1:
                    cv.circle(frame, (round(x), round(y)), 7, (0,255,0), 1)
        cv.putText(frame, "Frame: " + str(counter) + " of " +
            str(nframes), (80,585), cv.FONT_HERSHEY_SIMPLEX, 0.75,
            (255,255,255), 2)
        cv.imshow(f"Img", frame)
        if save:
            out_vid.write(frame)
        k = cv.waitKey(25) & 0xFF
        if k == 27:
            cv.destroyWindow('Img')
    if save:
        out_vid.release()
        subprocess.call(['ffmpeg', '-i', saved_video,
            f'{saved_video[:-4]}.mp4'])
        os.remove(saved_video)
    cv.destroyWindow('Img')

def frame_count(self):
    """ Video frame count """
    self.video = cv.VideoCapture(self.vid_file)
    return int(self.video.get(cv.CAP_PROP_FRAME_COUNT))
```

```
def denoise_frame(self, frame, gray = False, asint = True):
    denoised = denoise_wavelet(frame, channel_axis=-1,
        convert2ycbcr=True,
        method='BayesShrink',mode='soft',rescale_sigma=True)
    if asint:
        denoised = (denoised*255).astype(np.uint8)
    if gray:
        denoised = cv.cvtColor(denoised, cv.COLOR_BGR2GRAY)
    return denoised

def watch_backward(self,show=True,save=False):
    video = cv.VideoCapture(self.vid_file)
    success, frame = video.read()
    if not success: return
    all_frames = [frame]
    if save:
        cep_folder = os.path.split(self.vid_file)[0]
        saved_video = os.path.join(cep_folder,f'usb_{self.name}.avi')
        out_vid = cv.VideoWriter(saved_video,
            cv.VideoWriter_fourcc(*"MJPG"), 60, (self.w(), self.h()))

    while video.isOpened():
        success, frame = video.read()
        if not success:
            break
        all_frames.append(frame)

    all_frames.reverse()
    for frame in all_frames:
        if save:
            out_vid.write(frame)

        if show:
            cv.imshow("Backward",frame)
            # If ESC pressed
            k = cv.waitKey(25) & 0xFF
            if k == 27:
                break
    if save:
        out_vid.release()
        subprocess.call(['ffmpeg', '-i', saved_video,
            f'{saved_video[:-4]}.mp4'])
        os.remove(saved_video)
    if show:
        cv.destroyWindow(f"Backward")
    return

def get_colors(self,num, type =
    ('Multi',None),randomized=False,interval=(0,255)):
    """
    Makes sequence of color(s) based on parameters
    num -> number of colors in returned list
    type -> ('Multi',None) or ('Single','color') -> 'color' options are
        r,o,y,g,b,i(indigo),v(violet),p(pink),n(brown),k(black),w(white)
    randomized (bool), only set to True if type[0] = 'Multi'

    Returns list(tuples(len=3)) of RGB type colors
    """
    colors =
        {'r':(255,0,0),'o':(255,149,0),'y':(255,255,0),'g':(0,255,10),...
              'b':(0,220,255),'i':(0,0,255),'v':(100,20,200),'p':(255,0,255),...
              'n':[110,50,10],'k':(0,0,0),'w':(255,255,255)}
    color_num_key =
        {'1':'r','2':'rg','3':'rgi','4':'rogi','5':'rogiv','6':'rogivp',...
        '7':'roygivp','8':'roygbivp','9':'roygbivpn','10':'roygbivpnk','11':'roygbivpnkw'}
    min_,max_ = interval
    if min_ > 0:
        for col in colors.keys():
            colors[col] =
                (max(colors[col][0],min_),max(colors[col][1],min_),...
```

```
                max(colors[col][2],min_))
        if max_ < 255:
            for col in colors.keys():
                colors[col] =
                    (min(colors[col][0],max_),min(colors[col][1],max_),...
                    min(colors[col][2],max_))

        # Multicolor
        if type[0] == 'Multi':
            options = list(color_num_key[str(num)])
            if randomized:
                random.shuffle(options)
            return [colors[c] for c in options]

        # Single color
        return [colors[type[1]] for i in range(num)]

class Box(object):
    def __init__(self,box_tuple):
        """
        box_tuple: tuple like (123,341,20,30) describing (x,y,width,height)
            of box
        """
        self.box = box_tuple
    def get(self):
        return self.box
    def get_int(self):
        return tuple([int(round(i)) for i in self.box])
    def centered_box(self,slength=100):
        (p1,p2) = self.center()
        return Box((p1-slength/2,p2-slength/2,slength,slength))
    def points(self):
        p1 = (self.box[0], self.box[1])
        p2 = (self.box[0] + self.box[2], self.box[1] + self.box[3])
        return p1, p2
    def width(self):
        return self.box[2]
    def height(self):
        return self.box[3]
    def center(self):
        """
        Returns the center point of the box (x,y)
        """
        return (self.box[0]+self.box[2]/2,self.box[1]+self.box[3]/2)
    def __str__(self):
        return str(self.box)

class Boxes(object):
    def __init__(self,boxes = []):
        """
        boxes: list of all Box objects in one video frame
        """
        self.boxes = boxes

    def add(self,box):
        self.boxes.append(box)

    def get(self):
        return self.boxes

    def set(self,newboxes):
        self.boxes = newboxes.get()

    def __len__(self):
        return len(self.boxes)
    def __str__(self):
        tups = [str(box) for box in self.boxes]
        return str(tups)

def dist(p1,p2):
```

```
    return sqrt((p2[0]-p1[0])**2+(p2[1]-p1[1])**2)

def angle_between2d(p0,p1,p2):
    th1 = atan2(p2[1]-p1[1],p2[0]-p1[0])
    th1_p = atan2(p1[1]-p0[1],p1[0]-p0[0])
    th_large = np.pi - th1
    return th_large+th1_p

# Preprocessing
def create_training_labels(lf, labels = 1):
    """
    Within the Log File (lf), 8 evenly distributed frames are chosen.
    The points highlighted in 'tracking_template.png' are labeled on those 8
        frames for all 6 videos,
    and the point values are exported to 'tracking_points.csv'
    Set labels = 1 unless video has been labeled already and is being
        re-labeled, in which case choose 2,3 etc.
    """
    data_dir = os.path.split(lf.vid_file)
    template =
        cv.imread(os.path.join(data_dir,'cep','tracking_template.png'),0)
    cv.imshow('Template',template)

    d = {}
    d['Points'] = []
    frames = [1,3,5,7]
    nframes = lf.frame_count()
    step = nframes//9
    cap = cv.VideoCapture(lf.vid_file)
    print('Log File',lf.name)

    for i in frames:
        print(f'Frame {i}/9 - {i*step}/{nframes}')
        cap.set(1,i*step)
        success, frame = cap.read()
        cv.putText(frame, f"Frame: {i*step} of {nframes}", (80,585),
            cv.FONT_HERSHEY_SIMPLEX, 0.75, (255,255,255), 2)
        xcol_name = f'f{i}_x'
        ycol_name = f'f{i}_y'
        d[xcol_name] = []
        d[ycol_name] = []
        for j in range(1,4):
            _box = Box(cv.selectROI('Img', frame))
            x,y = _box.center()
            cv.circle(frame,(round(x),round(y)),4,(0,0,255),-1)
            cv.imshow('Img',frame)
            if i == frames[0]:
                pt_name = f'pt{j}_{lf.name[2:]}'
                d['Points'].append(pt_name)
            d[xcol_name].append(x)
            d[ycol_name].append(y)

    csv_dir = os.path.join(data_dir,'cep',f'tracking_points{labels}.csv')

    dframe = pd.DataFrame(data=d)
    dframe.to_csv(csv_dir)

# Tracking algorithms
def combined_RSTCxRTC(lf_fwd,lf_bwd, save=False):
    """
    Params: lf_fwd: a Log video played forwards and lf_bwd: a Log video
        played backwards
    To create a backwards video, run lf_fwd.watch_backwards(save=True)
    Set save=True if you want to save point data to {lf_fwd_name}_points.csv

    Runs RSTC between each of the 4 correction frames instead of between
        start and end frames of the entire video
    """
    epsilon = 0.001
    tracker = 'LK'
```

```
# Labeled data
data_dir = os.path.split(lf_fwd.vid_file)[0]
tracking_dir = os.path.join(data_dir,'cep','tracking_points.csv')
labeled = pd.read_csv(tracking_dir)

nframes = lf_fwd.frame_count()
all_points_combined = np.zeros((nframes,3,2))

labeled_sub = labeled[labeled.video==lf_fwd.name]
x_vals = list(labeled_sub.x)
y_vals = list(labeled_sub.y)
frames = sorted(list(set(labeled_sub.frame_num)))

# Tracking before first labeled point
b_points = []
for pt in range(3):
    b_points.append((x_vals[pt],y_vals[pt]))
path_bwd = lf_bwd.method_errors_combined(tracker,nframes-frames[0],...
        b_points,step=frames[0],show=False)
path_bwd = path_bwd[::-1]
all_points_combined[:frames[0]] = path_bwd[:frames[0]]

# Tracking between first and last labeled point
for j,frame in enumerate(frames[:-1]):
    f_points, b_points = [], []
    dframe = frames[j+1]-frames[j]
    f_points, b_points = [], []
    for pt in range(3):
        f_points.append((x_vals[j*3+pt],y_vals[j*3+pt]))
        b_points.append((x_vals[(j+1)*3+pt],y_vals[(j+1)*3+pt]))

    path_fwd = lf_fwd.method_errors_combined(tracker, frame, f_points,
        step = dframe,show=False)
    path_bwd = lf_bwd.method_errors_combined(tracker,
        nframes-frame-dframe, b_points, step = dframe,show=False)
    path_bwd = path_bwd[::-1]

    xs = np.array([i+frame for i in range(dframe)])
    B = -2*np.log(1/epsilon-1)/(xs[-1]-xs[0])
    C = (xs[0]+xs[-1])//2
    sig1 = 1/(1+np.exp(-B*(xs-C)))
    sig2 = 1/(1+np.exp(B*(xs-C)))
    for pt in range(3):
        for xy in range(2):
            sig_array = path_fwd[xs][:,pt,xy]*sig1 +
                path_bwd[xs][:,pt,xy]*sig2
            all_points_combined[frame:frame+dframe,pt,xy] = sig_array

# Tracking after last labeled point
f_points = []
for pt in range(3):
    f_points.append((x_vals[-3+pt],y_vals[-3+pt]))
path_fwd =
    lf_fwd.method_errors_combined(tracker,frames[-1],f_points,step=frames[0],show=False)
all_points_combined[frames[-1]:] = path_fwd[frames[-1]:]
lf_fwd.follow_points(all_points_combined, start_ind = 0, save=True)

if save:
    d = {}
    for pt in range(all_points_combined.shape[1]):
        d[f'pt{pt+1}_x'] = all_points_combined[:,pt,0]
        d[f'pt{pt+1}_y'] = all_points_combined[:,pt,1]
    df = pd.DataFrame(data=d)
    df.to_csv(f'us_final_{lf_fwd.name}.csv')
```

Appendix A

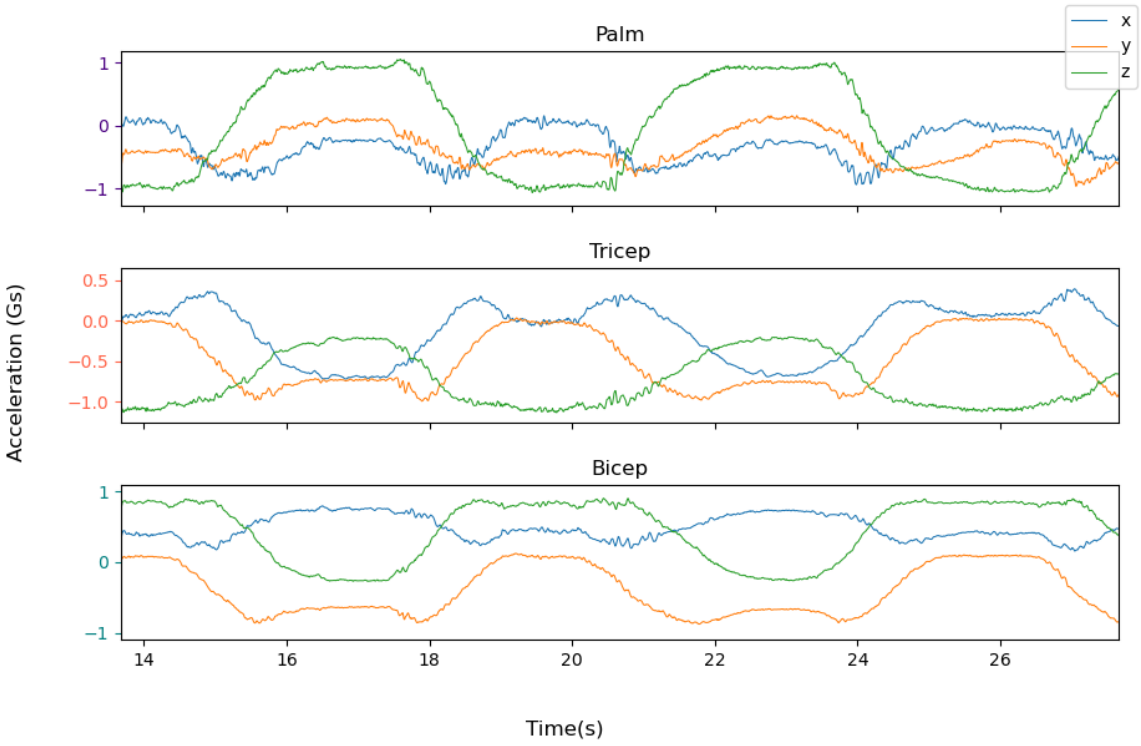


Figure A-1: The signals from the triceps and biceps sensors are visibly similar while the signal from the palm sensor is noisier. Because of this, the palm signal also results in a higher TPM.

Bibliography

[1] Paul Crawford and Ethan E. Zimmerman. Differentiation and diagnosis of tremor. *American Family Physician*, 83(6):697–702, March 2011.

[2] Günther Deuschl, Jan Raethjen, Michael Lindemann, and Paul Krack. The pathophysiology of tremor. *Muscle & Nerve*, 24(6):716–735, 2001. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/mus.1063.

[3] Paul Crawford and Ethan E. Zimmerman. Tremor: Sorting Through the Differential Diagnosis. *American Family Physician*, 97(3):180–186, February 2018.

[4] Günther Deuschl, Peter Bain, Mitchell Brin, and Ad Hoc Scientific Committee. Consensus Statement of the Movement Disorder Society on Tremor. *Movement Disorders*, 13(S3):2–23, 1998. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/mds.870131303.

[5] Andreas Puschmann and Zbigniew K. Wszolek. Diagnosis and Treatment of Common Forms of Tremor. *Seminars in neurology*, 31(1):65–77, February 2011.

[6] Not Available Habib-ur Rehman. Diagnosis and Management of Tremor. *Archives of Internal Medicine*, 160(16):2438–2444, September 2000.

[7] R. Bhidayasiri. Differential diagnosis of common tremor syndromes. *Postgraduate Medical Journal*, 81(962):756–762, December 2005. Publisher: The Fellowship of Postgraduate Medicine Section: Review.

[8] M. C. Akbostanci, K. V. Slavin, and K. J. Burchiel. Stereotactic ventral intermedial thalamotomy for the treatment of essential tremor: results of a series of 37 patients. *Stereotactic and Functional Neurosurgery*, 72(2-4):174–177, 1999.

[9] Joseph Jankovic and Eduardo Tolosa. Clinical Rating Scale for Tremor. In *Parkinson's Disease and Movement Disorders*, page 644. Williams & Wilkins, Baltimore-Munich, sixth edition, May 2015.

[10] Rodger J. Elble. The Essential Tremor Rating Assessment Scale. *Journal of Neurology & Neuromedicine*, 1(4), July 2016.

[11] Joseph Jankovic and Jau-Shin Lou. Essential tremor. *Neurology*, 42(7):1433–1433, July 1992. Publisher: Wolters Kluwer Health, Inc. on behalf of the American Academy of Neurology Section: Correspondence.

[12] Adriano O. Andrade, Adriano Alves Pereira, Maria Fernanda Soares de Almeida, and Guilherme Lopes Cavalheiro. Human Tremor: Origins, Detection, and Quantification. In *Practical Applications in Biomedical Engineering*. IntechOpen, London, 2013.

[13] Laura V. Schaefer and Frank N. Bittmann. Mechanotendography: description and evaluation of a novel method for investigating the physiological mechanical oscillations of tendons using a piezo-based measurement system. *European Journal of Translational Myology*, 31(1), March 2021. Number: 1.

[14] Mark Hallett. Overview of Human Tremor Physiology. *Movement Disorders*, 13(S3):43–48, 1998. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/mds.870131308.

[15] Laura V. Schaefer, Nils Löffler, Julia Klein, and Frank N. Bittmann. Mechanomyography and acceleration show interlimb asymmetries in Parkinson patients without tremor compared to controls during a unilateral motor task. *Scientific Reports*, 11(1):2631, January 2021. Number: 1 Publisher: Nature Publishing Group.

[16] Justin J. Kavanagh and Hylton B. Menz. Accelerometry: A technique for quantifying movement patterns during walking. *Gait & Posture*, 28(1):1–15, July 2008.

[17] Phillip Peng, Roderick Finlayson, Sang Hoon Lee, and Anuj Bhatia. Ultrasound for Interventional Pain Management. In *Ultrasound for Interventional Pain Management*, page 362. Springer Cham, 1 edition, September 2019.

[18] Amir Manbachi and Richard S C Cobbold. Development and Application of Piezoelectric Materials for Ultrasound Generation and Detection. *Ultrasound*, 19(4):187–196, November 2011. Publisher: SAGE Publications.

[19] Dale Ensminger and Bond Leonard. Medical Applications of Ultrasonic Energy. In *Ultrasonics: Fundamentals, Technologies, and Applications,*, pages 583–620. CRC Press, Boca Raton, 3rd edition edition, 2012.

[20] Paul E. Bigeleisen, Michael Gofeld, and Steven L. Orebaugh. *Ultrasound-Guided Regional Anesthesia and Pain Medicine*. Lippincott Williams & Wilkins, 2 edition, November 2009. Google-Books-ID: Urxj_8dIpcsC.

[21] Barys Ihnatsenka and André Pierre Boezaart. Ultrasound: Basic understanding and learning the language. *International Journal of Shoulder Surgery*, 4(3):55–62, 2010.

[22] Aladin Carovac, Fahrudin Smajlovic, and Dzelaludin Junuzovic. Application of Ultrasound in Medicine. *Acta Informatica Medica*, 19(3):168, 2011.

[23] Joe B. Woodhouse and Eugene G. McNally. Ultrasound of skeletal muscle injury: an update. *Seminars in ultrasound, CT, and MR*, 32(2):91–100, April 2011.

[24] Xiaoli Wang and Mei Yang. The Application of Ultrasound Image in Cancer Diagnosis. *Journal of Healthcare Engineering*, 2021:8619251, November 2021.

[25] Jae-Young Yu, Jin-Gyu Jeong, and Byung-Hun Lee. Evaluation of muscle damage using ultrasound imaging. *Journal of Physical Therapy Science*, 27(2):531–534, February 2015.

[26] David C. Nieman, R. Andrew Shanely, Kevin A. Zwetsloot, Mary Pat Meaney, and Gerald E. Farris. Ultrasonic assessment of exercise-induced change in skeletal muscle glycogen content. *BMC sports science, medicine & rehabilitation*, 7:9, 2015.

[27] Pan Li, Xuebing Yang, Guanjun Yin, and Jianzhong Guo. Skeletal Muscle Fatigue State Evaluation with Ultrasound Image Entropy. *Ultrasonic Imaging*, 42(6):235–244, November 2020. Publisher: SAGE Publications Inc.

[28] G. R. Adams, M. R. Duvoisin, and G. A. Dudley. Magnetic resonance imaging and electromyography as indexes of muscle function. *Journal of Applied Physiology*, 73(4):1578–1583, October 1992. Publisher: American Physiological Society.

[29] G. R. Adams, R. T. Harris, D. Woodard, and G. A. Dudley. Mapping of electrical muscle stimulation using MRI. *Journal of Applied Physiology*, 74(2):532–537, February 1993. Publisher: American Physiological Society.

[30] C. J. Zuurbier and P. A. Huijing. Changes in geometry of activily shortening unipennate rat gastrocnemius muscle. *Journal of Morphology*, 218(2):167–180, 1993. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jmor.1052180206.

[31] M V Narici, T Binzoni, E Hiltbrand, J Fasel, F Terrier, and P Cerretelli. In vivo human gastrocnemius architecture with changing joint angle at rest and during graded isometric contraction. *The Journal of Physiology*, 496(1):287–297, 1996. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1996.sp021685.

[32] Guang-Quan Zhou, Phoebe Chan, and Yong-Ping Zheng. Automatic measurement of pennation angle and fascicle length of gastrocnemius muscles using real-time ultrasound imaging. *Ultrasonics*, 57:72–83, March 2015.

[33] Micha Feigin, Bryan J. Ranger, and Brian W. Anthony. Statistical consensus matching framework for image registration. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 1827–1832, December 2016.

[34] A. Sarti, C. Corsi, E. Mazzini, and C. Lamberti. Maximum likelihood segmentation of ultrasound images with Rayleigh distribution. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 52(6):947–960, June 2005. Conference Name: IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control.

[35] Ting Yun, Yi Qing Xu, and Lin Cao. Semi-Supervised Ultrasound Image Segmentation Based on Curvelet Features. *Applied Mechanics and Materials*, 239-240:104–114, 2013. Conference Name: Measurement Technology and its Application ISBN: 9783037855454 Publisher: Trans Tech Publications Ltd.

[36] Ehsan Jokar and Hossein Pourghassem. Kidney Segmentation in Ultrasound Images Using Curvelet Transform and Shape Prior. In *2013 International Conference on Communication Systems and Network Technologies*, pages 180–185, April 2013.

[37] Rishu Gupta, Irraivan Elamvazuthi, Sarat Chandra Dass, Ibrahima Faye, Pandian Vasant, John George, and Faizatul Izza. Curvelet based automatic segmentation of supraspinatus tendon from ultrasound image: a focused assistive diagnostic method. *BioMedical Engineering OnLine*, 13(1):157, December 2014.

[38] Yuichi Notomi, Takahiro Shiota, Zoran B. Popović, Joan A. Weaver, Stephanie J. Oryszak, Neil L. Greenberg, Richard D. White, James D. Thomas, Randolph M. Setser, Richard D. White, Peter Lysyansky, and Maureen G. Martin-Miklovic. Measurement of Ventricular Torsion by Two-Dimensional Ultrasound Speckle Tracking Imaging. *Journal of the American College of Cardiology*, 45(12):2034–2041, June 2005.

[39] Bo-I Chuang, Jian-Han Hsu, Li-Chieh Kuo, I-Ming Jou, Fong-Chin Su, and Yung-Nien Sun. Tendon-motion tracking in an ultrasound image sequence using optical-flow-based block matching. *BioMedical Engineering OnLine*, 16(1):47, April 2017.

[40] Mohammad Hassan Jahanandish, Nicholas P. Fey, and Kenneth Hoyt. Lower Limb Motion Estimation Using Ultrasound Imaging: A Framework for Assistive Device Control. *IEEE Journal of Biomedical and Health Informatics*, 23(6):2505–2514, November 2019. Conference Name: IEEE Journal of Biomedical and Health Informatics.

[41] Elif Ayvali and Jaydev P. Desai. Optical Flow-Based Tracking of Needles and Needle-Tip Localization Using Circular Hough Transform in Ultrasound Images. *Annals of Biomedical Engineering*, 43(8):1828–1840, August 2015.

[42] Daniel Tenbrinck, Sönke Schmid, Xiaoyi Jiang, Klaus Schäfers, and Jörg Stypmann. Histogram-Based Optical Flow for Motion Estimation in Ultrasound Imaging. *Journal of Mathematical Imaging and Vision*, 47(1):138–150, September 2013.

[43] Guillaume Zahnd, Maciej Orkisz, André Sérusclat, Philippe Moulin, and Didier Vray. Evaluation of a Kalman-based block matching method to assess the bidimensional motion of the carotid artery wall in B-mode ultrasound sequences. *Medical Image Analysis*, 17(5):573–585, July 2013.

[44] Ting-Yu Lai, Hsiao-I Chen, Cho-Chiang Shih, Li-Chieh Kuo, Hsiu-Yun Hsu, and Chih-Chung Huang. Application of a novel Kalman filter based block matching method to ultrasound images for hand tendon displacement estimation. *Medical Physics*, 43(1):148–158, 2016. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1118/1.4937932.

[45] Kiros Karamanidis, Artemis Travlou, Peter Krauss, and Uwe Jaekel. Use of a Lucas–Kanade-Based Template Tracking Algorithm to Examine In Vivo Tendon Excursion during Voluntary Contraction Using Ultrasonography. *Ultrasound in Medicine & Biology*, 42(7):1689–1700, July 2016.

[46] José Seabra, Francesco Ciompi, Petia Radeva, and João Miguel Sanches. A Rayleigh Mixture Model for IVUS Imaging. In Joao Miguel Sanches, Andrew F. Laine, and Jasjit S. Suri, editors, *Ultrasound Imaging: Advances and Applications*, pages 25–47. Springer US, Boston, MA, 2012.

[47] Ashish Khare, Manish Khare, Yongyeon Jeong, Hongkook Kim, and Moongu Jeon. Despeckling of medical ultrasound images using Daubechies complex wavelet transform. *Signal Processing*, 90(2):428–439, February 2010.

[48] Deep Gupta, R. S. Anand, and Barjeev Tyagi. Despeckling of ultrasound medical images using ripplet domain nonlinear filtering. *Signal, Image and Video Processing*, 9(5):1093–1111, July 2015.

[49] Fan Zhang, Yang Mo Yoo, Liang Mong Koh, and Yongmin Kim. Nonlinear Diffusion in Laplacian Pyramid Domain for Ultrasonic Speckle Reduction. *IEEE Transactions on Medical Imaging*, 26(2):200–211, February 2007. Conference Name: IEEE Transactions on Medical Imaging.

[50] K Anders Ericsson, Ralf Th Krampe, and Clemens Tesch-Romer. The Role of Deliberate Practice in the Acquisition of Expert Performance. 100:363–406, 1993.

[51] K. Anders Ericsson and Kyle W. Harwell. Deliberate Practice and Proposed Limits on the Effects of Practice on the Acquisition of Expert Performance: Why the Original Definition Matters and Recommendations for Future Research. *Frontiers in Psychology*, 10, 2019.

[52] Fredrik Ullén, David Zachary Hambrick, and Miriam Anna Mosing. Rethinking expertise: A multifactorial gene–environment interaction model of expert performance. *Psychological Bulletin*, 142(4):427–446, 2016.

[53] Steven J. Chatfield, Donna H. Krasnow, Amanda Herman, and Glenna Blessing. A Descriptive Analysis of Kinematic and Electromyographic Relationships of the Core During Forward Stepping in Beginning and Expert Dancers. *Journal of Dance Medicine & Science*, 11(3):76–84, September 2007.

[54] Munenori Uemura, Morimasa Tomikawa, Ryuichi Kumashiro, Tiejun Miao, Ryota Souzaki, Satoshi Ieiri, Kenoki Ohuchida, Alan T. Lefor, and Makoto

Hashizume. Analysis of hand motion differentiates expert and novice surgeons. *Journal of Surgical Research*, 188(1):8–13, May 2014.

[55] Paul S. Slack, Chris J. Coulson, X. Ma, P. Pracy, S. Parmar, and K. Webster. The effect of operating time on surgeon's hand tremor. *European Archives of Oto-Rhino-Laryngology*, 266(1):137–141, January 2009.

[56] Christian Gölz, Claudia Voelcker-Rehage, Karin Mora, Eva-Maria Reuter, Ben Godde, Michael Dellnitz, Claus Reinsberger, and Solveig Vieluf. Improved Neural Control of Movements Manifests in Expertise-Related Differences in Force Output and Brain Network Dynamics. *Frontiers in Physiology*, 9, 2018.

[57] Wen-Tzu Tang, Wen-Yu Zhang, Chien-Chun Huang, Ming-Shing Young, and Ing-Shiou Hwang. Postural tremor and control of the upper limb in air pistol shooters. *Journal of Sports Sciences*, 26(14):1579–1587, December 2008. Publisher: Routledge _eprint: https://doi.org/10.1080/02640410802287063.

[58] Nađa Dardagan, Adnan Brđanin, Džemil Džigal, and Amila Akagic. Multiple Object Trackers in OpenCV: A Benchmark, October 2021. arXiv:2110.05102 [cs].

[59] Peter Janku, Karel Koplik, Tomáš Dulík, and Istvan Szabo. Comparison of tracking algorithms implemented in OpenCV. *MATEC Web of Conferences*, 76:04031, January 2016.

[60] Olfa Haggui, Matossouwé Agninoube Tchalim, and Baptiste Magnier. A Comparison of OpenCV Algorithms for Human Tracking with a Moving Perspective Camera. In *2021 9th European Workshop on Visual Information Processing (EUVIP)*, pages 1–6, June 2021. ISSN: 2471-8963.

[61] V. De Luca, T. Benz, S. Kondo, L. König, D. Lübke, S. Rothlübbers, O. Somphone, S. Allaire, M. A. Lediju Bell, D. Y. F. Chung, A. Cifor, C. Grozea, M. Günther, J. Jenne, T. Kipshagen, M. Kowarschik, N. Navab, J. Rühaak, J. Schwaab, and C. Tanner. The 2014 liver ultrasound tracking benchmark. *Physics in Medicine & Biology*, 60(14):5571, July 2015. Publisher: IOP Publishing.

[62] Jeffrey Stoll and Pierre Dupont. Passive Markers for Ultrasound Tracking of Surgical Instruments. In James S. Duncan and Guido Gerig, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2005*, Lecture Notes in Computer Science, pages 41–48, Berlin, Heidelberg, 2005. Springer.

[63] Yun Lei, Xiaoqing Ding, and Shengjin Wang. AdaBoost Tracker Embedded in Adaptive Particle Filtering. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 4, pages 939–943, August 2006. ISSN: 1051-4651.

[64] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online Multiple Instance Learning. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 983–990, June 2009. ISSN: 1063-6919.

[65] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-Backward Error: Automatic Detection of Tracking Failures. In *2010 20th International Conference on Pattern Recognition*, pages 2756–2759, August 2010. ISSN: 1051-4651.

[66] David S. Bolme, J. Ross Beveridge, Bruce A. Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2544–2550, June 2010. ISSN: 1063-6919.

[67] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-Learning-Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1409–1422, July 2012. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[68] João F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-Speed Tracking with Kernelized Correlation Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, March 2015. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[69] David Held, Sebastian Thrun, and Silvio Savarese. Learning to Track at 100 FPS with Deep Regression Networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 749–765, Cham, 2016. Springer International Publishing.

[70] Alan Lukežič, Tomáš Vojíř, Luka Čehovin Zajc, Jiří Matas, and Matej Kristan. Discriminative Correlation Filter Tracker with Channel and Spatial Reliability. *International Journal of Computer Vision*, 126(7):671–688, July 2018.

[71] Bruce Lucas and Takeo Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision (IJCAI). In *7th international joint conference on Artificial intelligence*, volume 81, April 1981.

[72] Simon Baker and Iain Matthews. Lucas-Kanade 20 Years On: A Unifying Framework. *International Journal of Computer Vision*, 56(3):221–255, February 2004.

[73] Ebrahim Karami, Mohamed S. Shehata, and Andrew Smith. Tracking of the internal jugular vein in ultrasound images using optical flow. In *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–4, April 2017.

[74] Mohammad Wasih and Mohamed Almekkawy. Motion Tracking of Carotid Artery in Ultrasound Images Using Lucas Kanade Method with Advanced Siamese Neural Networks. In *2021 IEEE International Ultrasonics Symposium (IUS)*, pages 1–4, September 2021. ISSN: 1948-5727.

[75] Alexander Mathis, Pranav Mamidanna, Kevin M. Cury, Taiga Abe, Venkatesh N. Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. *Nature*

Neuroscience, 21(9):1281–1289, September 2018. Number: 9 Publisher: Nature Publishing Group.

[76] Eldar Insafutdinov, Leonid Pishchulin, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. DeeperCut: A Deeper, Stronger, and Faster Multi-person Pose Estimation Model. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 34–50, Cham, 2016. Springer International Publishing.

[77] Alan Wrench and Jonathan Balch-Tomes. Beyond the Edge: Markerless Pose Estimation of Speech Articulators from Ultrasound and Camera Images Using DeepLabCut. *Sensors*, 22(3):1133, January 2022. Number: 3 Publisher: Multidisciplinary Digital Publishing Institute.

[78] Rebecca L. Krupenevich, Callum J. Funk, and Jason R. Franz. Automated analysis of medial gastrocnemius muscle-tendon junction displacements in heathy young adults during isolated contractions and walking using deep neural networks. *Computer Methods and Programs in Biomedicine*, 206:106120, July 2021.

[79] Joan K.-Y. Ma and Alan A. Wrench. Automated assessment of hyoid movement during normal swallow using ultrasound. *International Journal of Language & Communication Disorders*, 57(3):615–629, 2022. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1460-6984.12712.

[80] Tanmay Nath, Alexander Mathis, An Chi Chen, Amir Patel, Matthias Bethge, and Mackenzie Weygandt Mathis. Using DeepLabCut for 3D markerless pose estimation across species and behaviors. *Nature Protocols*, 14(7):2152–2176, July 2019. Number: 7 Publisher: Nature Publishing Group.