Lab 5a and b: Closed-loop neural interfaces

0. Assignments associated with this lab

- Comprehension questions (section 4)
- You do not have to turn in your completed code. However, you are asked to provide the figures generated by running your analyses (last question of the comprehension questions).
  - All figures that must be provided are underlined in the lab manual and flagged in the code templates
- You must write a discussion section describing your findings in the context of existing scientific understanding, highlighting any caveats or data interpretation questions, and highlighting future directions and applications of your results..
  - You should use the IEEE conference paper template
  - Your discussion section should not exceed one page.
- Assignments cover both parts a and b of this lab, which you have two lab sessions to complete

1. Equipment required

- Hardware
  - Musclebox pro
    - Connection cable to connect to computer
    - Connection cable to EMG electrode (1)
  - Computer with USB connection
- Software
  - Backyard Brains Spike Recorder
  - Matlab
  - Lab Streaming Layer software library for Matlab
  - USB streaming software interface ("Backyard Brains")
  - Scripts outlines and functions for analysis

2. Background

This lab will give you hands-on experience with using electrophysiology signals to control a device in real-time. We will use surface electromyography (sEMG) to control a computer cursor in real-time as a subject receives feedback about cursor position, creating a closed-loop system. We will 1) write a script to stream sEMG data in real-time, running at a set loop rate, 2) build a script to map sEMG activity onto movements of a cursor (using a Kalman Filter) to perform a center-out task, and 3) test how two properties of our 'control system'—the time-lag between neural signal and device movement, and the variable being controlled (position or velocity)—influence neural interface performance.

2.1 types of neural interfaces

There are many different types of neural interfaces. Here, we define key terms to characterize these systems. Schematics of each system are shown in Figure 1.

Afferent vs. Efferent: Afferent and efferent describe the direction of information flow in a neural interface. The term efferent is used to describe information flowing from the brain to the periphery, while the term afferent is used to describe information flowing from the periphery into the brain. Neural interfaces that use signals from

the brain to control other devices are therefore efferent interfaces. Neural interfaces that enable external signals to access the brain through neural stimulation are therefore afferent interfaces. Examples of efferent interfaces include motor brain-machine interfaces where signals from the brain control a prosthetic arm. The most ubiquitous example of an afferent neural interface is deep brain stimulation, where electrical current is injected into neural tissue to modulate pathological neural activity.

Open-loop vs. closed-loop: In addition to differences in information flow, neural interfaces may be open-loop or closed-loop. Open-loop designs lack feedback in control of the external device, while closed-loop systems provide feedback within the system. Simple closed-loop control systems, for example, include an efferent interface where neural signals control a prosthetic arm and the user is provided feedback (e.g., through vision or hearing) to close the control loop. Afferent interfaces can also operate in closed-loop, for instance by using movement monitoring to detect a tremor and modify levels of deep brain stimulation.

Finally, systems where information out of and into the brain are both performed through an artificial interface are **bi-directional** neural interfaces. Bi-directional approaches record signals from the brain and ultimately send a transformed version of those neural signals back into the brain via an external pathway (i.e. separate from natural biological sensory systems). The key to this process is sensing of endogenous neural activity and using this in a control loop to precisely stimulate and/or regulate neural activity. Applications for this technology potentially include neural prostheses to restore both motor and sensory function, as well as functional cures for neural abnormalities like epilepsy that detect a seizure event and suppress onset via neuromodulation.

In this lab, we will create efferent closed-loop neural interfaces where the loop is closed with natural sensory feedback (vision), as shown in fig. 1c.

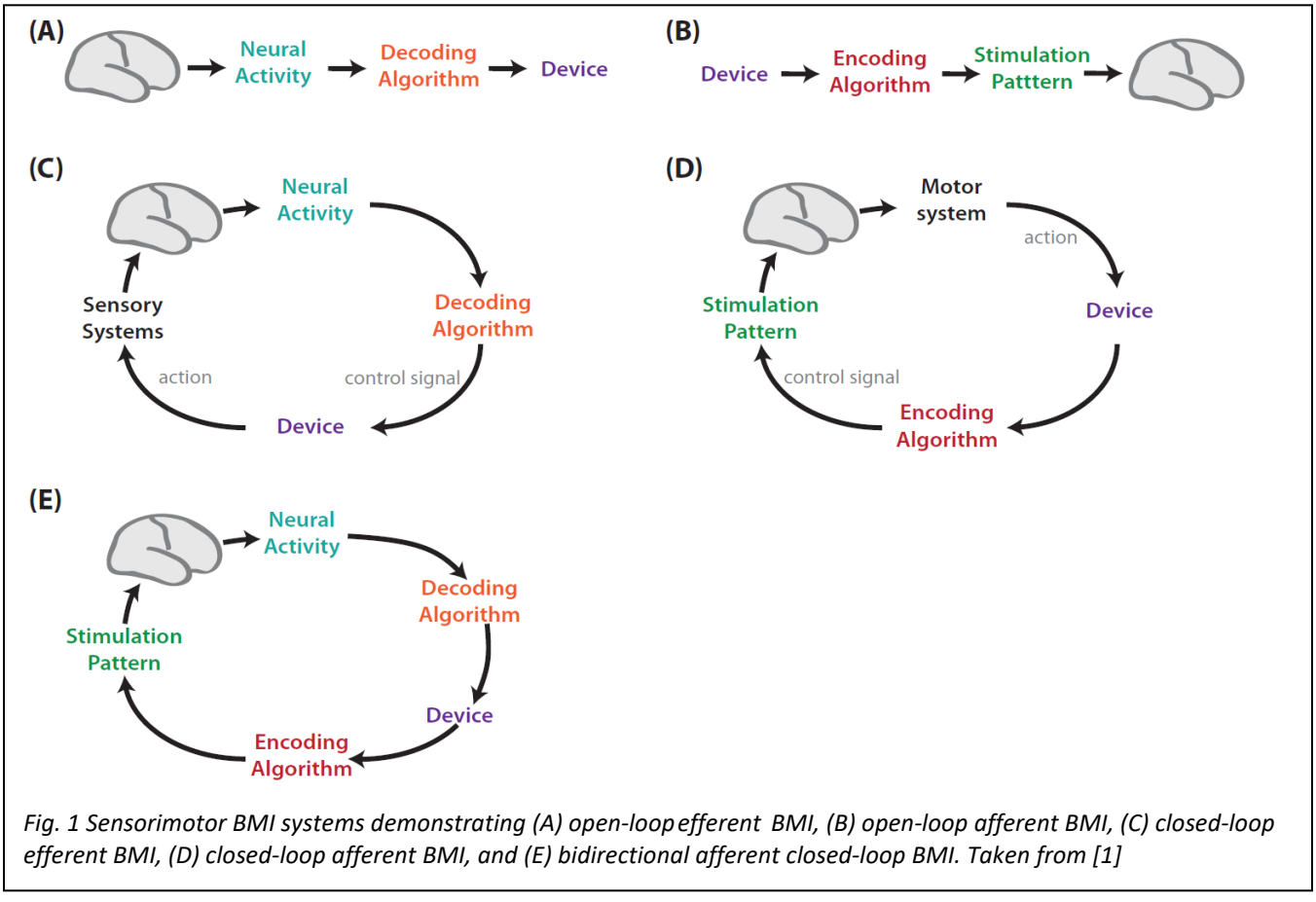

Fig. 1 Sensorimotor BMI systems demonstrating (A) open-loop efferent BMI, (B) open-loop afferent BMI, (C) closed-loop efferent BMI, (D) closed-loop afferent BMI, and (E) bidirectional afferent closed-loop BMI. Taken from [1]

2.2 Electromyography refresher

Recall from lab 3 that EMG is best recorded using a *differential amplifier*, which serves to better suppress the background electrical activity in a muscle. Our Muscle Spikerbox uses a differential amplifier. You will thus record sEMG by attaching two electrodes to the skin directly over a muscle and recording the potential difference between them as the muscle contracts. A third electrode is used as a ground, which may be attached anywhere nearby, usually not directly over a muscle (e.g. a boney portion of the body like the elbow).

As we saw in lab 3, raw EMG amplitude is not a useful proxy for muscle force. Signals that instead capture the magnitude of EMG (like root-mean square measures of variance) are more informative. In this lab, we will use another way to extract signal amplitude: rectification (i.e. the absolute value). This will be our neural 'feature'.

2.3 Kalman Filter refresher

In lab 4, we worked with the Kalman Filter. We will use this filter to map EMG activity into cursor movements. As a reminder, the KF assumes linear state evolution (1) and state observation models (2):

$$x_t = \boldsymbol{A}x_{t-1} + w_t \qquad (1)$$
$$y_t = \boldsymbol{H}x_t + q_t \qquad (2)$$

where x_t and y_t represent represents the movement variables and recorded neural activity at time t, respectively. $w_t \sim N(0, \mathbf{W})$ and $q_t \sim N(0, \mathbf{Q})$ are noise terms. The filter is thus specified by the matrices **A**, **W**, **H**, and **Q**.

Based on the models above, the Kalman Filter recursively estimates the current movement variables x_t based on both the past observed movements ($x_{t-1}, \ldots x_0$) and currently observed neural activity y_t. The recursive estimation scheme is as follows:

1) Estimate of movement variables based solely on state-evolution model (*a priori* estimate):
$$\hat{x}_{t|t-1} = \boldsymbol{A}x_{t-1}$$
2) Estimate the covariance (uncertainty) of the *a priori* estimate:
$$\boldsymbol{P}_{t|t-1} = \boldsymbol{A}\boldsymbol{P}_{t-1}\boldsymbol{A}^T + \boldsymbol{W}$$
3) Compute the expected neural activity corresponding to the *a priori* estimated movement, and the difference between the expected and observed neural activity (measurement residual)
$$\tilde{y}_t = y_t - \boldsymbol{H}\hat{x}_{t|t-1}$$
4) Compute the covariance (uncertainty) of this measurement residual
$$\boldsymbol{S}_t = \boldsymbol{H}\boldsymbol{P}_{t|t-1}\boldsymbol{H}^T + \boldsymbol{Q}$$
5) Compute an estimate of kinematics incorporating the recorded neural activity (*a posteriori* estimate)
$$\hat{x}_t = \hat{x}_{t|t-1} + \boldsymbol{K}\tilde{y}_t$$
$$\boldsymbol{K} = \boldsymbol{P}_{t|t-1}\boldsymbol{H}^T\boldsymbol{S}_t^{-1}$$
6) Compute the covariance (uncertainty) of the *a posteriori* estimate:
$$\boldsymbol{P}_t = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H})\boldsymbol{P}_{t|t-1}$$

In many neural interfaces, the A, W, H, and Q matrices are learned from a training dataset. **In this lab, we will instead use our understanding and intuition for the Kalman Filter model developed in lab 4c to specify the A, W, H, and Q matrices by hand to give the cursor desired dynamics and relationships with muscle activity.**

2.4 Real-time neural interfaces (rates, lags, and how they're implemented)

Closed-loop systems must be implemented in "real-time". This means that at every time-point, the full computation loop of the interface must run. We will be implementing an efferent closed-loop interface (Fig. 1c). This means that at each time-point, we must 1) collect new data, 2) process the data, 3) use data for prediction (through a decoder), and 4) update the state of the "device".

```
Define bin-size and loop time DELTA_T
Define time that loop will run RUN_TIME
Initialize variables to store inputs and predictions
Open data buffer stream
Pull data from the buffer to clear it
Initialize time counter i

While i < RUN_TIME
      Start a clock
      Read data from the buffer
      Average all received data from buffer (binning)
      Process data (e.g. signal conditioning)
      Kalman filter predictions
      Update cursor display
      Wait DELTA_T – time elapsed since loop started
      Increment i
end
```

Fig. 2 Pseudocode for real-time closed-loop efferent neural interface implementation

Rates and lags: As we saw in lab 4, discrete decoders (like the Kalman and Wiener filters) use neural activity averaged across a certain bin width. This bin width determines the **rate** of our closed-loop system, which is how frequently the loop computations are performed. The **lag** in a closed-loop system is the temporal delay between when neural activity occurs and when the corresponding movement of the device occurs.

Implementing real-time decoding: Our sEMG-controlled neural interface will be implemented by streaming data acquired by the Muscle SpikerBox into Matlab as it is collected (i.e. in "real-time"). To do this, we will use custom-written software that reads data from the computer's USB port and grabs it into Matlab . This software is provided to you, and full set-up/use information is provided below in section 3 and Appendix B.

While you do not need to know the full details of this software, basic knowledge of what it's doing is useful for understanding how we will implement the necessary computations for our closed-loop system. USB devices stream data into a running **buffer**, which collects all incoming data (from the device) until another computer process "pulls" data from the buffer (or a buffer size capacity is reached, at which point the buffer only stores the latest data and older data is overwritten). Pulling data from the buffer will empty the buffer. Our software simply opens a pipeline to allow Matlab to directly "pull" data from the USB data buffer.

In our neural interface, we will need to write code that reads data from the Muscle SpikerBox and "bins" it at the desired rate. Let's consider using a neural interface with a bin-width of 100 ms (10 Hz rate). Our computation loop must run every 100 ms (10 Hz). Within each loop, we want to read in 100 ms of data (current time point, looking up to 100 ms into the past), and then process and predict with that data. Figure 2 shows pseudocode of how we will implement this in our experiments. (You may notice that this implementation does not precisely control the bin size for the very first loop iteration. In practice, it is difficult to initialize the interface. A single time-bin of inaccuracy in bin size has no measurable impact on overall system performance.)

2.5 Center-out task structure and event-codes

We will use our sEMG neural interface to control a cursor in 1 dimension (moving left and right on the screen). We will use a behavioral task to test how well the user can control the cursor. The task we will perform is a center-out task, much like what the monkeys performed in the data we analyzed for labs 4 and 5. A successful trial requires a short hold at the center, moving to the peripheral target within a specified time-limit, and a brief hold at the target. The subject will be 'rewarded' with the target turning green when they successfully complete a trial.

Just like in the data we analyzed for lab 4, we will use task events and event-times to keep track of behavior. The event codes signify different types of events that could occur in the center-out task. Here is a list of event-code meanings:

- 1: center target appears (cuing start of a trial)
- 2: hand enters the center target
- 3: go-cue occurs (instructing the subject to start reaching)
- 5: hand enters the peripheral target
- 6: trial successful
- 7: trial failed
- 8: reach time-out error (did not enter the peripheral target before time-limit elapsed)
- 9: error holding at the center (hand left center before the go-cue)
- 10: error holding at the peripheral target (hand left before designated hold time elapsed)
- 11 – 12: code to note which target is presented (11 = target 1, 12 = target 2).

A successful trial will have a repeatable structure:

1 2 3 [target code] 5 6

3. Experiments

IMPORTANT NOTEs:

- **This lab will use two channels of sEMG (therefore 5 EMG electrodes). Please take care to avoid waste of electrodes. Select a designated subject and leave electrodes in place once you achieve good placement. Disconnecting the alligator clips but leaving electrodes in place can free up the subject during down-time/debugging.**
- **This lab relies heavily on the Muscle SpikerBoxes mixed with a lot of programming. To save batteries, when you are spending substantial time coding or debugging (that doesn't require testing streaming), please disconnect the streaming software and turn your spikerBox off.**

3.1 Experiment 1 – Real-time streaming of EMG data

All underlined plots should be included with your comprehension questions.

Set-up streaming software

We first need to set-up our software for streaming data from the Muscle SpikerBox into Matlab. Full instructions and requirements can be found in Appendix B. The appendix also includes troubleshooting information. Consult this troubleshooting throughout this lab if you are running into issues reading data from the SpikerBox.

Place electrodes to record EMG confirm signal quality

- Plug the MuscleBox into the computer, plug in the battery, and turn it on (via the 'Volume' knob)
- Plug in your electrode-connection cables to channel 1 and channel 2.
- Open the Spike Recorder software and confirm you are streaming channels 1 and 2 correctly
- Select one of your group members to be the research subject
- Select two forearm muscles to record from for your interface control. Some suggestions:
    - Before placing any electrodes, perform different wrist movements to feel where prominent muscles are located within the arm. Use this to identify good target placements.
    - Pick muscles that are active during clearly different actions. For instance, one that flexes the wrist and one that extends the wrist.
- Once you have identified target muscles and placements, place electrodes. A reminder that our EMG measurements are differential measurements between two recording electrodes on the target muscle, and a ground on a boney portion of the body (we recommend the elbow)
    - Recording electrode 1 (red lead)
    - Recording electrode 2 (red lead)
    - Ground electrode (black lead)
- Ask the subject to make movements as you observe the signals using the SpikeRecorder software. Assess your signal quality and electrode placement. Check whether you have muscle-specific recordings by having the subject:
    - make movements that should activate each muscle, and confirm that an EMG signal registers
    - make movements that should not activate each muscle, and confirm that you do not detect a significant EMG signal
    - Check whether there is different activity between channel 1 and channel 2 (i.e. you are recording different muscles in each channel). This is critical for success of our interface.

Exp. 1a: Create a script to plot streamed EMG data in Matlab

- Use the provided script 'bioen466_lab5_winter2023_executionExp1a.m' to build a loop that reads in EMG data and plots it in Matlab.

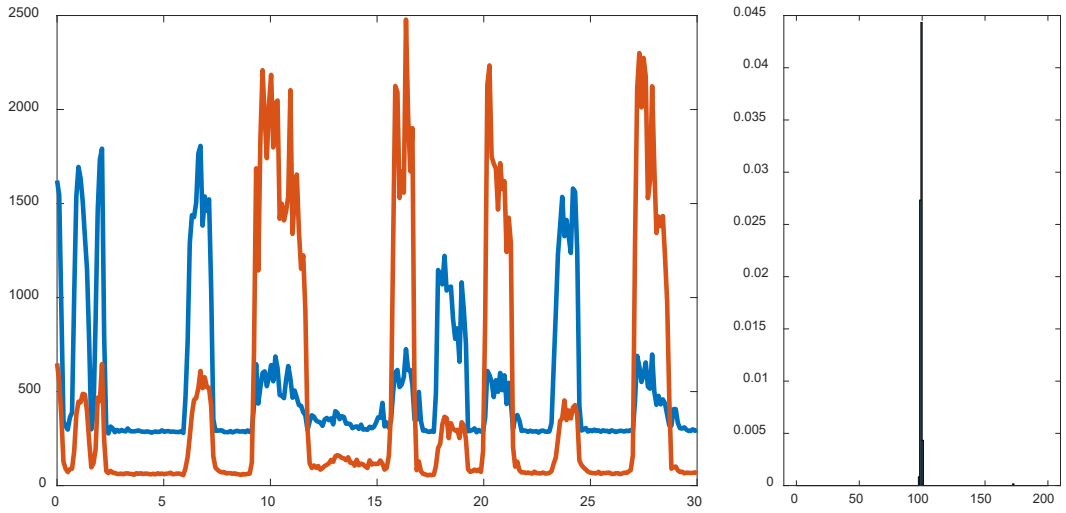

Fig. 3. Example expected output for experiment 1a

- This script contains the outlines of the pseudocode in Fig. 2 (excluding the "decoder" portion because we are just plotting the raw data itself)
- Fill in the script to implement your loop
- In order to run this script, we must have our streaming software connected. The order of operations should be:
    - Turn on SpikerBox
    - Run BackyardBrains.exe (within the 'BackyardBrains' folder)
    - 'Link' the detected device and confirm correct connection (see Appendix B)
    - Run your Matlab script
- An example of what your resulting streaming figure should look like is shown in Fig. 3. Once you have your streaming script running, ask the TA or Prof. Orsborn to check your results.
    - Save a figure with an example snap-shot of your streaming results.
- Once working properly, try running this at a few different rates (i.e. change DELTA_T to be smaller or larger)
    - Watch the histogram of elapsed time (i.e. how long our loop took to run)
    - Use the elapsed-time results to find the shortest bin size your system can execute well. On most reasonable computers, this should not be larger than 100 ms.

Exp. 1b: Add signal conditioning to our EMG streaming

- In the step above, you may notice that the EMG channels have very different DC offsets from each other, and somewhat arbitrary scaling such that maximum contraction for each muscle produces different amplitudes. (Why is that?)
- In order to get well-conditioned signals for our neural interface, we need to modify our script from step 1a to add offset-removal and scaling of our EMG inputs.
    - Instead of simply plotting X = mean (abs(emg data), we want to plot X' = A(X – B) where A and B are vectors (length 2 for our 2 channels).

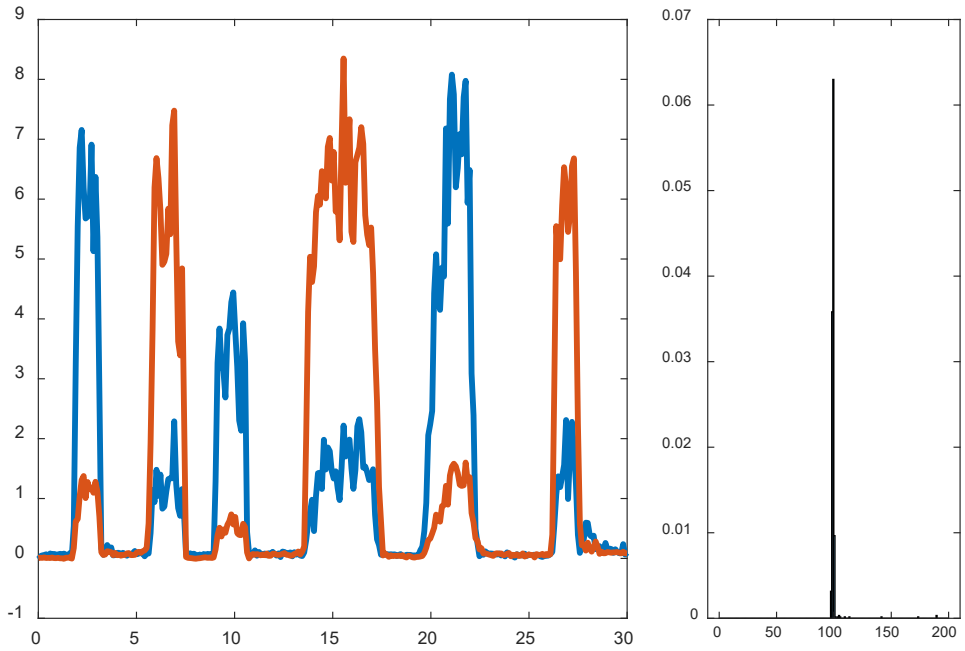

Fig. 4. Example expected output for experiment 1b (note different scaling of EMG signals)

- Because A and B are hard to know ahead of time, we will instead learn A and B by recording data for some time and then calculating them from this training data
- We have provided a script 'bioen466_lab5_winter2023_executionExp1b.m' which modifies our computations from 1a to add this step
    - Note: MOST of this script is the same as 1a. You can copy-paste the pieces you filled in on that experiment over to this script.
- Fill in the script to implement your loop
- An example of what your resulting streaming figure should look like is shown in Fig. 4. Once you have your streaming script running, ask the TA or Prof. Orsborn to check your results.
    - Save a figure with an example snap-shot of your streaming results.
- Note: To achieve best results, during the 'baseline estimation' period, the subject should perform several large contractions of both muscles.

There is no additional data analysis required for this experiment.

3.2 Experiment 2 – Use sEMG activity to move a cursor in a task

We will now use our streamed, conditioned EMG activity to control the movements of a cursor. The cursor will be controlled to play a simple behavioral task (a 'center-out' task, see section 2.5).

- We have provided a script 'bioen466_lab5_winter2023_executionExp2and3.m' that you will use for both this experiment and experiment 3.
- The script builds on the things you already implemented in experiment 1—much of what you need to fill in can be copied from experiment 1b
- The code includes a significant amount of programming that implements the task. You do not need to modify or add to any of this (**please don't**, it is complex to debug!). Task-related code is marked as such with notes where it begins and ends.
- The pieces you need to fill in are tagged with 'FILL IN' comments. Use this to find all the missing steps.

Set the matrices of the Kalman filter to specify a desired EMG - cursor mapping

We will map EMG activity into cursor movements using a Kalman Filter. We will start with having EMG activity control the *velocity* of the cursor. One muscle will move the cursor right (positive x velocity); the other muscle will move the cursor left (negative x velocity).

- Our state will be defined as $X(t) = [\ P_x(t)\ V_x(t)\ 1]^T$ where $P_x(t)$ is the x position, and $V_x(t)$ is the x velocity.
- Set A (the state transition model) so that:
    - $P_x(t)$ is the integral of velocity (i.e. $P_x(t+1) = P_x(t) + V_x(t)*DELTA\_T$)
    - $V_x(t)$ is related to itself with a small 'decay' of 0.9 (i.e. $V_x(t+1) = 0.9* V_x(t)$)
    - The constant term (last state) is always 1
- Set W (the state covariance model) so that:
    - It is a diagonal matrix
    - $P_x(t)$ is assumed to be fully known (i.e. fully specified by $V_x(t)$, so no covariance)
    - $V_x(t)$ variance is set to 0.9
- Set H (the observation model) so that:
    - EMG channel 1 is mapped to +X velocity with a scaling of 1 (i.e. $V_x(t) = emg1(t)$)
    - EMG channel 2 is mapped to -X velocity with a scaling of 1
    - The offset term is zero

- Set Q (the observation covariance model) so that:
    - It is a diagonal matrix
    - Each EMG channel has a variance of 0.2

Run your center-out task

- Once you have filled in and debugged your script, ask the TA or Prof. Orsborn to observe your cursor control to confirm it looks correct.
- Have your subject practice with the interface for a little bit (at least 20 trials) to learn how to use it
- Collect and save data as your subject completes 20 trials using the above Kalman filter settings and bin width of 100 ms.

There is no additional data analysis required for this experiment.

3.3 Experiment 3 – Quantify the effects of control system properties

We will now use our closed-loop sEMG interface to quantify the influence of two properties of the control loop: lags (experiment 3a) and the movement variable controlled (experiment 3b). **All underlined plots should be included with your comprehension questions.**

Experiment 3a: impact of lag

- Save a copy of your execution script completed for experiment 2, re-named appropriately
    - Keep the Kalman filter and bin size the same as experiment 2
- Modify this loop to add the ability to lag cursor position displayed on the screen (cursor_position) relative to the 'decoded' position (X_hat)
    - Define a constant LAG_T to represent the lag duration in seconds
    - Define a constant LAG_LENGTH that represents the lag duration in units of loop iterations (can be calculated from LAG_T and DELTA_T)
    - Modify the line: `cursor_pos(1:NUM_DIMS,i) = X_hat(1:NUM_DIMS,i)` to shift the cursor position plotted on the screen (cursor_pos) by LAG_LENGTH relative to the variable X_hat
- collect and save data as your subject completes 20 trials with 3 different levels of lags (you already have data for zero lag, collected in experiment 2)
    - LAG_T = 100 ms
    - LAG_T = 300 ms
    - LAG_T = 500 ms
- Use the provided table in Appendix A to keep track of your experimental metadata

Experiment 3b: impact of control variable

- Save a copy of your execution script completed for experiment 2, re-named appropriately
    - bin size the same as experiment 2, and lag at zero
- Modify the Kalman filter mapping to use EMG to control position, rather than velocity:
    - Update the W (the state covariance model) so that:
        - It is a diagonal matrix
        - $P_x(t)$ and $V_x(t)$ are set to 0.9
    - Set H (the observation model) so that:
        - EMG channel 1 is mapped to +X position with a scaling of 1 (i.e. $P_x(t)$ = emg1(t))

        - EMG channel 2 is mapped to -X position with a scaling of 1
        - There is no EMG contribution to velocity
    - A and Q do not need to be changed.
- collect and save data as your subject completes 20 trials with position control, rather than velocity
- Use the provided table in Appendix A to keep track of your experimental metadata

Data analysis

- We have provided an outline data analysis script called 'bioen466_lab5_winter2023_analysisExp3.mat'. Use this script to work through your data analysis
- The key steps of your analysis will be:
    - Load your data files (.mat files)
    - Trial-sort the events and event-times
    - Compute the success rate (# correct trials / #attempted trials) for each condition
    - Compute the reach time (time between the go-cue and entering the target) for successful trials
- The key plots you will generate through this script include
    - Success rate vs. loop lag
    - Reach time vs. loop lag
    - Success rate for position vs. velocity control (no lag in either condition)
    - Reach time for position vs. velocity control (no lag in either condition)

4. Comprehension Questions

1. What part of our neural interface makes it a closed-loop system? (1 sentence)

2. The rate of a neural interface system is set by the _____ of the neural feature used.

3. What other implementation details place constraints on the rate of a neural interface loop?

4. Describe the relationship you observed between control loop lags and performance of the neural interface (1 sentence). What part of the task appeared to be most influenced most by the lags (i.e. what did the subject struggle the most with)?

5. Which control variable (position or velocity) was easier for the subject to control? Briefly (2-3 sentences) explain why there is such a notable difference between these two systems. Hint: think about the temporal dynamics of the decoder (i.e. the role of the A matrix in the Kalman Filter).

6. In experiment 2, we used a state transition matrix with $V_x(t)$ related to itself with a small 'decay' of 0.9. What would you predict would happen if we reduced this term to 0.5? Describe how this would impact the cursor dynamics.

7. In our experiments, we used 2 muscles to control movements along a single direction (1 "degree of freedom"). We could have, instead, used each muscle to control a separate direction of cursor movement (making a 2D cursor).
   a. What property of the neural signals we have (our EMGs) makes the 2D interface infeasible with our existing approach? (Hint: think about how 1 muscle would control positive and negative velocity.)

   b. What part of our neural interface pipeline (data acquisition, signal pre-processing, decoding) would need to change to implement a 2D cursor interface? **Note:** there are several possible solutions—just list one option.

8. Append all requested figures to your comprehension questions.

5. References

[1] http://brain.ieee.org/wp-content/uploads/sites/52/2019/11/Roadmap-White-Paper-Version1-7Nov.pdf

Appendix A: Data logging table

Recording #	Lag (ms)	Control variable	Notes and observations

Appendix B: Streaming software installation steps and debugging (sEMG interface)

Installing:

All files needed for software installation can be found on the 'g' drive on the bioe computers: 466/lab6/softwareinstall/ or on Canvas (zip folders of each are in the Modules) . **This must be run on a Windows machine.**

If you downloaded zip-files, un-zip them. Then, copy the (unzipped) software folders into your /Documents/Matlab/ folder on your computer:

1) BackyardBrains
    a. This is a custom application to create a streaming port between the SpikerBox USB interface and Matlab
2) LSL
    a. This is a software package called 'Lab Streaming Layer' (https://github.com/sccn/labstreaminglayer) which has utilities to stream data from devices into software like Matlab, Python, etc.
3) liblsl-Matlab
    a. This is a software library within Lab Streaming Layer that is specialized for Matlab-based operations.

Testing the USB App:

Once you have installed the above software and libraries, we now need to test the USB streaming app ('BackyardBrains').

- Turn on your spikerBox and confirm it is detected by your computer
    - You can do this using the SpikeRecorder software (i.e. do you see a USB icon available in that software) or by listening for the 'device connected' sound on your computer
    - **Important Note:** You CANNOT have SpikeRecorder connected to your spikerBox when running the BackyardBrains app. The SpikerBox can only communicate with one piece of software at a time.
- Go into /Documents/Matlab/BackyardBrains/ and open 'BackyardBrains.exe'
    - If any security warnings pop up, tell Windows to trust the software
- You will now see a black command window open, as well as a grey small window titled 'BackyardBrains LSL'. (see Fig. B1)
    - If your SpikerBox is being detected correctly, the 'Device' box will be populated with a long string that contains the hardware address of the box (it's not important what it is, just that there is something in that box)
    - If your dialog box is empty, there is an issue detecting your hardware
- Select 'Link' on the gray window to connect your hardware to the app. If the connection is successful, you will see output in the command window describing the data stream properties (see Fig. B2).

Troubleshooting and common problems:

Here are a few likely/possible problems with streaming and their causes.

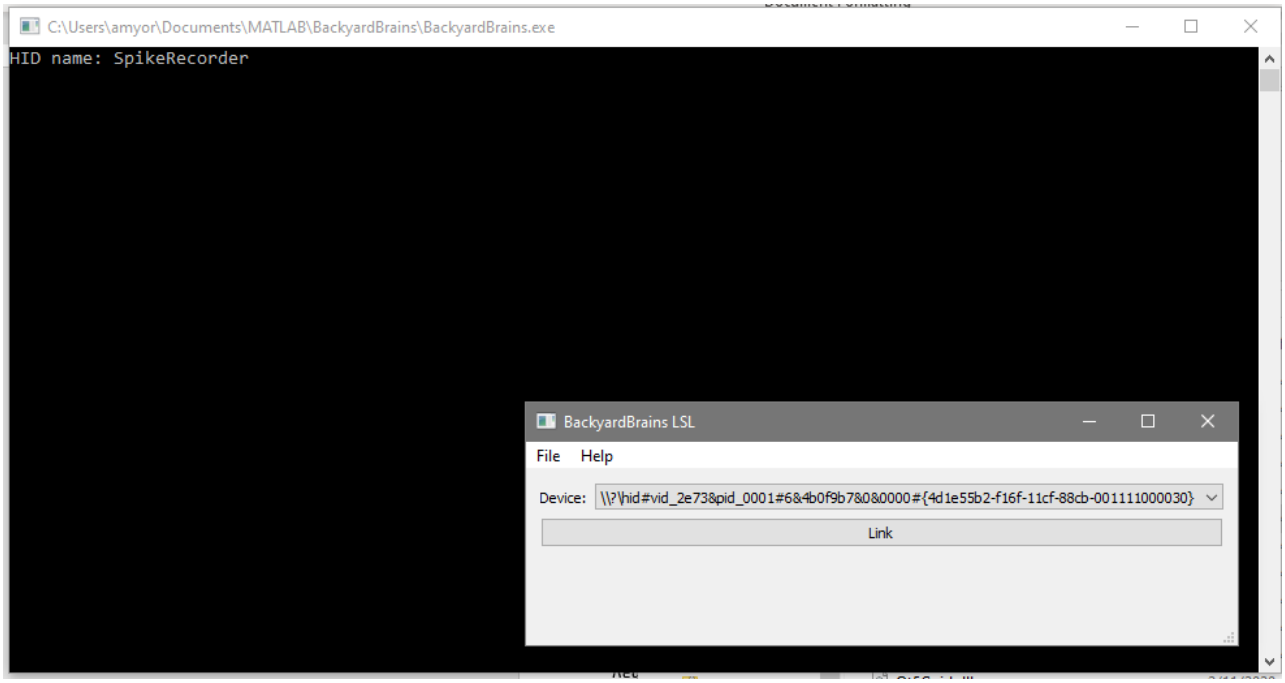

Fig. B1. Screenshot of Backyard Brains app display for a correctly detected device.

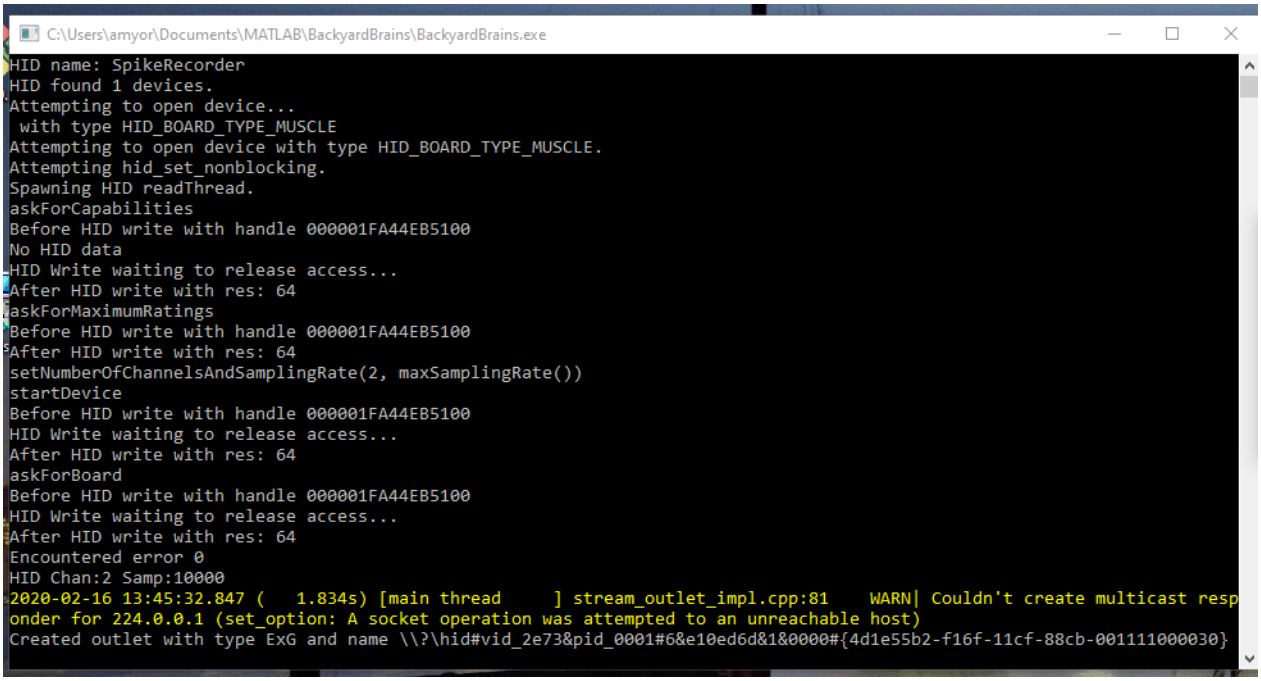

Fig. B2. Screenshot of Backyard Brains app display for a correctly connected device.

- An error message upon trying to run the BackyardBrains streaming app saying a particular file (usually a .dll file) is missing
    - This is an installation error and is unlikely to happen. If it does, please let Prof. Orsborn or the TA known immediately.
- Your SpikerBox is not detected by the BackyardBrains streaming app
    - Your device may not be on or is not properly recognized by your computer. Check that you can connect to it with SpikeRecorder
    - You may have other software (e.g. SpikeRecorder) already connecting to your SpikerBox, making the USB connection unavailable for the streaming app.
        - Suggested solution: Close any SpikeRecorder instances running, turn the SpikerBox off and back on (re-sets connections)
- When in Matlab trying to initialize your connection socket, you receive an error message about a function missing (a function with 'lsl' in the name)
    - You have not properly added the LSL and liblsl-Matlab directories to your Matlab path. Be sure that those folders were copied fully from what we provided and that the directories are in your Matlab path
- When in Matlab trying to initialize your connection socket, you receive an indexing error message at line '`info = streams{1};`' because the variable 'streams' is empty.
    - The streaming software ('BackyardBrains') is not able to connect to your device to open a data stream
    - Check that you have opened the BackyardBrains streaming app and linked to your device, and that your streaming app shows messages similar to what's shown in figure B2.

Appendix S: Lab Safety Considerations for Lab 5

This appendix describes the safety considerations for the procedures to be performed in this laboratory session. Read this appendix in full ***prior to beginning your experiments****. If you have any questions regarding the lab procedures or safety considerations, ask the TA or Professor Orsborn prior to starting your experiments.*

Once you understand these considerations in full, sign this form to indicate that you have read it and understand the procedures. Return the signed form to Professor Orsborn or the TA ***prior to beginning your experiments****.*

There are no safety considerations for this laboratory. No safety form submission is required.