Tethered Lean & Release System MATLAB GUI Instructions and Design Notes - LSL based

Tuesday, October 29, 2024 10:35 AM

App Design

Purpose

To display and plot the tether tension/estimated lean torque/estimated lean angle in real time. To provide researcher a clear indication of whether the test subject has lean enough/ready to be released.

Functionalities

Calculate lean angle and lean torque based on the force sensor readings and input parameters.

Plot results in real time.

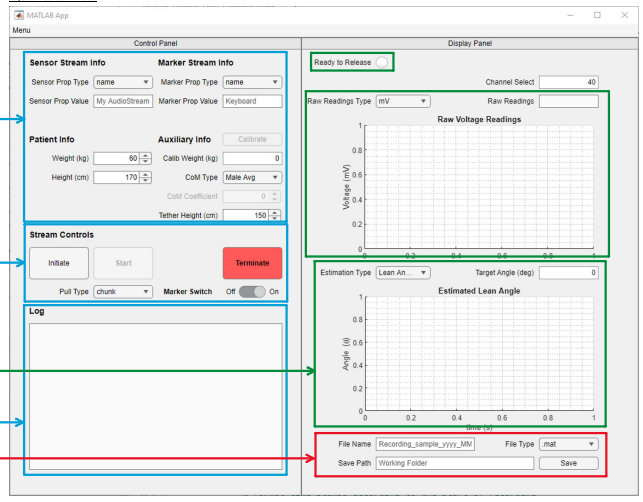
Stream Controls
- a) Initiate: initialize the library, obtain available channels based on channel properties
- b) Start: connect to the a channel inlet and start receiving data/time stamp vectors
- c) Stop: stop streaming and terminates the initialization

Info
- a) Stream Info
  - a. Prop type: the type of the stream property, such as "name", "type", "source_id", etc
  - b. Prop name: the name of the stream property, such as "Name" -> "My Keyboard"
  - c. Pull type: sample or chunk
  - d. Estimation type: either lean angle or lean torque
  - e. Target Value: toggle between target lean angle and target lean torque, depending on the estimation type
- b) Patient Info
  - a. Weight
  - b. height
- c) Auxiliary Info
  - a. height-CoM coefficient: the height of the CoM of patient over the height of the patient ratio, choose between established average or user-define
  - b. tether height
- d) Calibrate: calibrate the force sensor with know weights
  - a. Calib Weight: input the known weight
- e) Log: display detailed info and error messages

Display Panel
- a) Indicators: Release-Ready Indicator
- b) Estimates: Sensor force threshold, Lean Angle, Pivot-point Torque
- c) Sensor Info: tether tension, sensor force threshold, inclinometer angle read-out
- d) Channel number: the channel number to be displayed

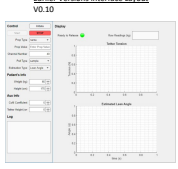
Save Panel: save recorded data.

Layout Breakdown v0.60

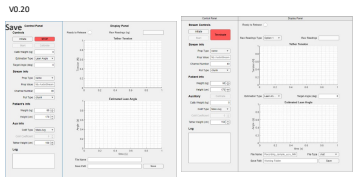


Earlier Versions Interface Layout

V0.10

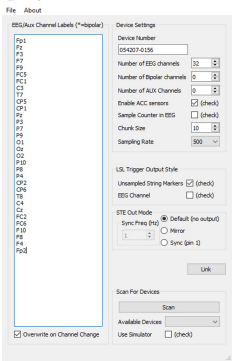


V0.20

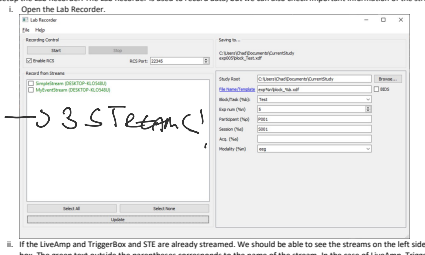


Use Instructions:

1. Preparations:
   - a. Setup the test subject.
     - i. Place the necessary sensors → specify
     - ii. Connect LiveAmp to BrainVision Recorder, etc.
     - iii. Put the harness on the subject
     - iv. Describe the experiment instructions to the subject.
     - v. Record the subject's height in cm.
     - vi. Record the subject's weight in kg.
     - vii. Note the subject's sex.
   - b. Setup the tethered-release system.
     - i. Adjust the metal plate height to about 3cm lower than the height of the metal clip on the subject's harness when the subject is standing straight with the harness on. Record the plate height in cm. → 2. → Harness is NOT ideal
     - ii. Clip the tether to the bow release and the clip on the subject's harness.
     - iii. Ask the tethered subject to walk away from the tethered-release system, following the red tape on the ground, until the tether is stretched straight.
     - iv. According to the experiment requirements, ask the subject to step back 1 or 2 steps to lean.
   - c. Setup the LiveAmp LSL Connector
     - i. Switch off the BrainVision Recorder.
     - ii. Switch on the LiveAmp LSL Connector on the Desktop of the MoBI Laptop.

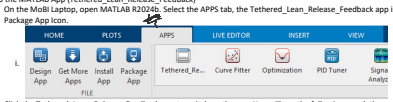


   - iii. Press Scan. The target LiveAmp should appear in the Available Devices Dropdown. We can confirm the LiveAmp by cross compare the Device Number to the brainVision Recorder's display. The number of EEG Channels will also automatically changed to 64 if we are using 2 LiveAmps.
   - iv. Specify Number of AUX Channels, for our STE box, we type in 8.
   - v. Check the Enable ACC sensors box.
   - vi. Keep the Chunk Size and Sampling Rate default. → 500Hz R 10 channels are
   - vii. Check the EEG Channel under LSL Trigger Output Style. This allows a standalone EEG channel-styled marker for de-jittering in later analysis. (However, this functionalities and its compatibility to the MATLAB App hasn't been fully checked as of 04/22)
   - viii. Press Link.
   - ix. If successfully linked, the Link button will change to Unlink.
   - d. Setup the Lab Recorder. The Lab Recorder is used to record data, but we can also check important information of the stream.
     - i. Open the Lab Recorder.

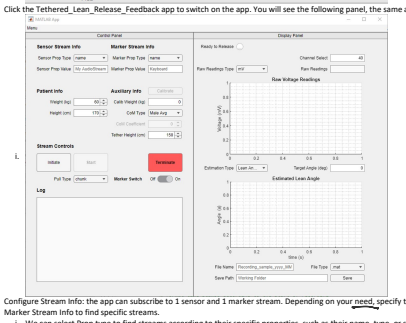


→ 3 STREAMS!

     - ii. If the LiveAmp and TriggerBox and STE are already streamed. We should be able to see the streams on the left side Record from Streams box. The green text outside the parentheses corresponds to the name of the stream. In the case of LiveAmp, TriggerBox, and STE connected, we should be able to see 3 streams. The name of the stream will be the device number followed by DeviceTrigger or STETriggerIn.
     - iii. Click Select All.
     - iv. If changes are made to the streams, click Update. The updated streams will show up in the Record from Streams box shortly.
     - v. Configure the Save path and save file name.
     - vi. Click Start to start recording. But you may want to start recording after using the MATLAB App to visualize all important sensors and triggers and make sure that they function properly.
2. Setup the MATLAB App (Tethered_Lean_Release_Feedback)
   - a. On the MoBI Laptop, open MATLAB R2024b. Select the APPS tab, the Tethered_Lean_Release_Feedback app is the 1st app to the right of the Package App icon.
     - i.



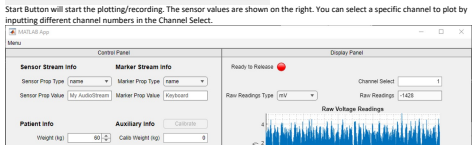
   - b. Click the Tethered_Lean_Release_Feedback app to switch on the app. You will see the following panel, the same as the layout shown above.
     - i.



   - c. Configure Stream Info: the app can subscribe to 1 sensor and 1 marker stream. Depending on your need, specify the Sensor Stream Info and the Marker Stream Info to find specific streams.
     - i. We can select Prop type to find streams according to their specific properties, such as their name, type, or source_id.
     - ii. For instance, we can find the LiveAmp EEG Stream by: (check the name of the streams via the Lab Recorder)
       1) Sensor Prop Type: type; Sensor Prop Value: EEG
       2) Sensor Prop Type: name; Sensor Prop Value: 054207-0156-xxx → Good: How to get the Marker stream
     - iii. The marker stream will automatically find the type marker stream. Specify the prop type and value similarly.
     - iv. The log will display the founded stream. Cross validate the stream information to make sure that we connected to the correct stream.
   - d. Configure Patient Info
     - i. Input the patient weight and height.
     - ii. Patient Info: Weight (kg): 60; Height (cm): 170
   - e. Configure Auxiliary Info
     - i. Input the Tether Height (which is the metal plate height from the ground) in the previous "b. Setup the tethered-release system." stage.
     - ii. Tether Height (cm): 150
     - iii. Configure the CoM Type according to the subject's sex. You can specify your own CoM ratio by selecting custom in the CoM Type and type in the CoM.
     - iv. CoM Type: Male Avg; CoM Coefficient: 0 → MALE: .65 FEMALE .55 → cite later!
     - v. CoM Type: Custom; CoM Coefficient: 0.65
   - f. Configure Stream Controls
     - i. Switch off Marker Switch if you do not have/want to stream a marker. If No marker is detected, the marker switch will automatically turn off.
     - ii. Marker Switch: Off On
     - iii. Initiate Button will check for streams that satisfy the prop type and values specified in the "c. Stream Info" section.
     - iv. Initiate
     - v. See Log for details of the connected stream or if the stream is connected.
     - vi. Log:

```
Loading the library...
Sensor name: MyAudioStream
Resolving Sensor Streams...
1 Stream(s) Found
Selected the Stream No.1
   Name: MyAudioStream
   Type: Audio
   Number of Channels: 1
   Nominal Sampling Rate: 11025 Hz
   Source ID: Microphone (6- Usb Audio Device)
```

     - vii. Start Button will start the plotting/recording. The sensor values are shown on the right. You can select a specific channel to plot by inputting different channel numbers in the Channel Select.



Notes - Commonly used functions

LSL-MATLAB Lib

1. Basics
   - a. lsl_loadlib(binarypath, debugging, keep_persistent) -> Load LSL library
2. Stream Receiving
   - a. lsl_resolve_all(lib, timeout) -> Resolve all visible /currently available streams
     - i. Returns: A cell array of stream info objects (excluding their desc field), any of which can subsequently be used to open an inlet. The full info can be retrieved from the inlet.
     - ii. disp('Resolving all streams...');

```
% resolve all streams...
results = lsl_resolve_all(lib);
```

   - b. lsl_resolve_byprop(lib, prop, value, minimum, timeout) -> Resolve streams among currently available streams by user-specified properties
     - i. Properties include: "name", "type", "source_id", or 'desc/manufature'
     - ii. Properties are usually user-configurable in the individual capture executables (AudioCapture.exe, etc.).
     - iii. result = lsl_resolve_byprop(lib,'name','MyAudioStrean');
   - c. lsl_inlet(info, maxbuffered, chunksize, recover, process) -> Using the handle generated by lsl_resolve_xx() to open an inlet of the streams to receive data.
     - i. inlet = lsl_inlet(result{1});
     - ii. Lsl_inlet.delete(self) -> Delete the current inlet instance
     - iii. Lsl_inlet.info(self, timeout) -> Retrieve info of this inlet
     - iv. Lsl_inlet.open_stream(self, timeout) -> Subscribe to the specified data stream from the moment this function is called. Samples pushed into this stream will be retrieved via pull_sample() or pull_chunk() calls.
     - v. Lsl_inlet.close_stream(self) -> Close the specified stream
     - vi. Lsl_inlet.time_correction(self, timeout) -> Retrieve an estimated time correction offset for the given stream
     - vii. Lsl_inlet.set_postprocessing(self, processing_flags) -> Set timestamp correction postprocessing flags.
     - viii. Lsl_inlet.pull_sample(self, timeout, binary_blobs) -> Pull a sample from the inlet and read it into an array of values
       1) Returns: data
       2) Returns: timestamp
     - ix. Lsl_inlet.pull_chunk(self) -> Pull a chunk of numeric samples and their timestamps from the inlet
       1) Returns: ChunkData. The chunk contents; this is a MxN matrix with one column per returned sample (and as many rows as the stream has channels).
       2) Returns: Timestamps. A vector of timestamps for the returned samples.
3. Stream Generation
   - a. lsl_streaminfo(lib, name, type, channelcount, samplingrate, channelformat, sourceid) -> Generate a stream with user-specified configurations
   - b. lsl_outlet(info, chunksize, maxbuffered) -> Open an outlet from the Info generated by lsl_streaminfo().
     - i. Lsl_outlet.delete(self) -> Delete the current outlet instance
     - ii. Lsl_outlet.push_sample(self, sampledata, timestamp, pushthrough) -> Push a sample into the outlet
     - iii. Lsl_outlet.push_chunk(self, chunkdata, timestamps, pushthrough) -> Push a chunk of sample into the outlet
     - iv. Lsl_outlet.info(self) -> Retrieve info of this outlet

MATLAB App Designer

1. Basics
   - a. Define properties -> app-use proprietary variables
   - b. Define functions -> app-use functions
   - c. Arrange app layout
2. Callback Functions
   - a. Connect callback functions to control/plot state-change/interactions
   - b. Loops inside a callback function and ways to terminate looping
     - i. Stop button callback changes ifStop variable
     - ii. Loop condition ~ifStop variable
     - iii. Callback set to interruptable
3. Plot immediately for every loop
   - a. Drawnow
   - b. Drawnow limitrate

Math

In-app Implementation v0.60

```
function CalculateForce(app)
    % According to the Force Sensor (LSB302) and Signal Conditioner
    % (IAA100) Settings, the conversion is around 30lbs/V
    if isequal(class(app.data_vec_F), 'double')
        app.force = (app.data_vec_F / 1e6) * 30 * 0.453592 * app.grav_acc; % uV to V to lbs to kg to N
    else
        app.force = -1 * ones(size(app.data_vec_F));
    end
end

function CalculateAngle(app)
    % calculates the angle from the derived force
    m = app.mass;
    H = app.patient_height / 100; % convert to meters
    g = app.grav_acc;
    alpha = app.CoMCoeff;
    F = app.force;
    h = app.tether_height;

    theta = abs(asin((F * h) / (m * g * H * alpha))); % in radians
    app.lean_angle = theta / pi * 180; % in degrees
end

function CalculateTorque(app)
    % calculate torque at the subject's anckle from the derived
    % force
    F = app.force;
    h = app.tether_height;

    app.lean_torque = F * h;
end
```

Lean-Release Physics Model

Assumptions:
1. Patient is rigid.
2. Patient is stationary.

Variables:

Symbol	Description
theta	Lean Angle
m	Person's Body Mass
R	Tether length
H*	Person's Chest Height
h	Height of Force Sensor
H	Person Height
l	Length from ground to CoM, should be estimated
F	Expected force sensor reading or tether tension
alpha	CoM to Height Ratio

Geometrics:
R = h sin(theta)

From Equation of Motions:
Angle:
m g l sin(theta) = F h
m g H alpha sin(theta) = F h

Torque at ankle:
tau = F h

Version History

Update 11/20 v0.10

Tested functionalities
- a. Initiate -> can initiate library anytime
- b. Stop -> can stop streaming and delete stream info anytime.
- c. Start -> can successfully create inlet and receive data.
- d. Pull type toggle between chunk and sample, should use chunk by default to avoid missing data points
- e. Log is correctly displayed
- f. Raw data is correctly displayed
- g. Raw data is correctly plotted
- h. Estimated data is correctly plotted
- i. Estimation type is correctly toggled, also changes displayed value.
- j. Scrolling display

To be tested
- a. Patients's info -> weight and height is correctly recorded and used
- b. Aux Info CoM coefficient is correctly selected and used
- c. Calibration is correct -> needs to build a rack for the known weight to directs tension to gravity direction, see "Force Sensor Zero and Span Calibration Setup"

To be implemented
- a. Channel selection -> with a multi-channel input, find the correct channel with data, need to be tested with the EEG cart ActiChamp
- b. Indicator status determination -> based on avg data values over a time period instead? To ensure stability
- c. Display target lean angle and target lean torque threshold?
- d. Calibtration
- e. Save data
- f. Stream data?

Update 12/12 v0.45

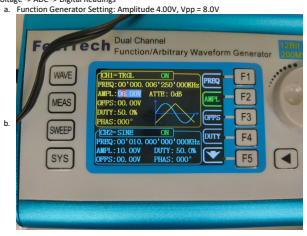
Timeseries object causes crashes, consider implementing with the newer time table object instead.

To Do:

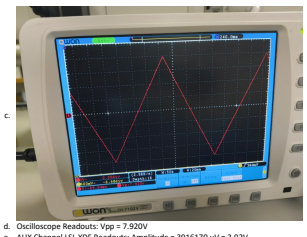
Implement time table:
- a. Object declaration/initialization
- b. Append new data
- c. Test for performance, latency and consistency, etc.

Update 04/21 v0.60
1. Checked the voltage to force conversion
2. Checked the force to angle conversion
3. Swaped Controls and Infos Panel for more intuitive use of the app.
4. Added calibration function.

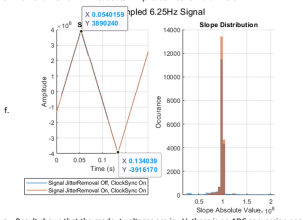
ADC Conversion and App Code Check:
1. Voltage -> ADC -> Digital Readings
   - a. Function Generator Setting: Amplitude 4.00V, Vpp = 8.0V
   - b.



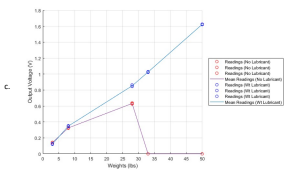
   - c.



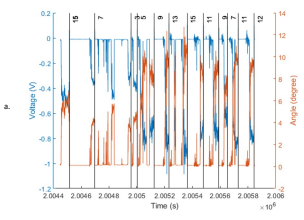
   - d. Oscilloscope Readouts: Vpp = 7.920V
   - e. AUX Channel LSL XDF Readouts: Amplitude = 3916170 uV = 3.92V
   - f.



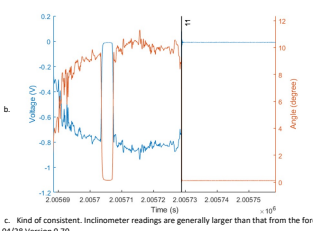
   - g. Result shows that the readout voltages are in uV, there is no ADC conversion needed. The ADC conversion could be done by the LiveAmp or ActiChamp System.
2. Weight -> Tether Tension (force) -> Voltage -> ADC -> Digital Readings
   - a. 300lbs -> 136.078kg -> 10V -> 30lbs/V or 13.61kg/V
   - b. Experimental Data with some friction, about 33lbs/V
   - c.



3. Lean Angle (Weight, Height ,etc.) -> Tether Tension -> Voltage -> ADC -> Digital Readings -> Conversion Formula -> Actual Lean Angle
   - a.



   - b.

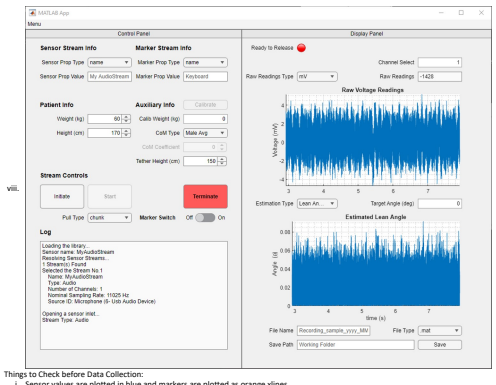


   - c. Kind of consistent. Inclinometer readings are generally larger than that from the force sensor.

Update 04/28 Version 0.70

New features
1. Added percent weight display

Trouble shoot
1. Cannot start stream, error message "vertcat(app.TT_all; TT_new);" Dimensions of arrays being concatenated are not consistent.

viii. 

g. Things to Check before Data Collection:
   i. Sensor values are plotted in blue and markers are plotted as orange xlines.
   ii.

   iii. You want to select the appropriate channel for the accelerometer of the force sensor. The channel numbers follow the following convention.
      1) 32 or EEG channels are channel 1 : 32 or 1 : 64.
      2) User specified number of AUX channels (we are using ActiCHamp or LiveAmp STE, both have 8 AUX channels) are 32 + 8 or 64 + 8 => 33 : 40 or 65 : 72. The number of AUX channels is specified in the LiveAmp connector App. The order is from AUX 1 to AUX 8.
      3) The LiveAmp onboard accelerometer, if enabled, will be additional 3 channels following the AUX channels, thus 41 : 43 or 73 : 75. the order follows x, y, z.
   iv. After selecting the appropriate channel, we can check if we have the correct signal.
      1) If we are plotting the force sensor, try pull the tether, the readouts should change correspondingly.
      2) If we are plotting the accelerometer, try moving the LiveAmp or the standalone sensor.
      3) If we are also plotting the trigger button, click the button and looks for the orange xlines, as shown in the figure above.
   v. If anything is wrong, try looking at the Log outputs and/or restart the MATLAB App, LiveAmp Connector, and the Lab Recorder.
h. Record: now go back to the Lab Recorder and click Start. We are ready for the experiment.
i. Indicator Lights: the indicator light indicate whether the subject has lean to a sufficient angle (specified by the Target Angle (deg)) and ready to release. Red means the angles are too large. White means angles are too small. Green means just right. The same rule can apply for the torque by switching the Estimation Type drop down to the Lean Torque.

```
if value >= target_value + app.tolerance * target_value % if value is too large, outside of tolerance
    app.ReadytoReleaseLamp.Color = 'r';
elseif value < target_value % if value is too small
i.      app.ReadytoReleaseLamp.Color = 'w';
else % if value is sufficient and within tolerance.
        app.ReadytoReleaseLamp.Color = 'g';
end
```

j. Calibrate: Not tested yet. You should be able to calibrate the sensor using a known weight.
   i. Connects the known weight (could be nothing) to the tethered lean-release system.
   ii. Input the calib weight to the edit field.
   iii. Start the stream.
   iv. The Calibrate Button should be enabled.
   v. Click Calibrate.
   vi. An Offset will be calculated according to the difference between the known weight force and the force sensor force.

```
% Button pushed function: CalibrateButton
function CalibrateButtonPushed(app, event)
vii.    % the difference between the measured force and the calibration weight force should be the offset.
    app.offset = (app.CalibWeightkgEditField.Value * app.grav_acc) - app.force;
end
```