UTILIZING NEURAL NETWORKS AND WEARABLES TO QUANTIFY HIP JOINT ANGLES AND MOMENTS DURING WALKING AND STAIR ASCENT

# Code Appendices Volume

by

MEGAN MCCABE

Bachelor of Arts Honors Thesis

Thayer School of Engineering
Dartmouth College
Hanover, New Hampshire

Date_____

Approved:_____
Advisor's Signature

Megan McCabe
Signature of Author

**Table of Contents**

1. APDM Pre-Processing ……. 3

1.1 Import APDM …… 3

1.1.1 Main Script ……. 3

1.1.2 Function to convert .apdm to .h5 to .mat …… 6

1.2 Filter APDM data …….. 8

1.3 Split logged APDM data by activity trial ……. 10

1.4 Identify "priority" trials for processing …… 11

2. Loadsol Pre-Processing …….. 12

2.1 Import Loadsol ……. 12

2.2 DC-shift Loadsol data …….. 14

2.3 Filter Loadsol data …… 16

2.4 Export Loadsol data into .mot file …… 18

2.4.1 Script ……. 18

2.4.2 Function to identify MOCAP markers by name …… 20

2.4.3 Function to write .mot files …… 21

3. MOCAP Pre-Processing ……. 23

3.1 Import MOCAP …… 23

3.2 Export MOCAP to .trc format …….. 25

3.4.1 Script …….. 25

3.4.2 Function to write .trc files …….. 27

4. Syncing …… 29

4.1 Main Script …… 29

4.2 Function to create gait percentage vector …….. 36

5. ANN Design …… 38

5.1 Create and train ANN (10 rounds) …… 38

5.2 Average performance metrics across rounds …… 41

5.3 Ensemble average W-A results ……. 44

5.4 Plot W-A and I-S ensemble averaged results …… 46

5.5 Create violin plots …….. 48

6. OpenSim Results – Data Analysis ……... 50

6.1 Import ID results (.sto files) ……... 50

6.1.1 Main script …….. 50

6.1.2 Function to convert .sto files to .mat …….. 52

6.2 Import IK results (.mot files) ……. 53

6.2.1 Main script …….. 53

6.2.2 Function to convert .mot files to .mat ……. 55

6.3 Consolidate hip ID and IK results …….. 56

6.4 Split data into gait cycles …… 58

6.4.1 Main script ……... 58

6.4.2 Function to split gait cycles ……. 60

6.5 Plot ensemble averaged results for just the I-S approach ……... 61

6.5.1 Main script ……... 61

6.5.2 Function to plot …… 64

6.6 Calculate gait parameters …… 66

## 1. APDM Pre-Processing

### *1.1 Import APDM*

#### *1.1.1 Main Script*

```
% A1) ImportAPDM_4IMUs.m
%
% This script loads 4 .apdm files and stores the data in a matlab
table.
% Note: At the end of the script, remember to save "rawAPDM".
%
% Megan McCabe
% Created Dec 2019
% Edited Feb 2020

%% Convert apdm data from .apdm format to .h5 format, then import into
matlab struct
addpath('/Users/megan_mccabe/Desktop/Thesis/Data Pre-Processing/APDM
Pre-Processing/RMC APDM to MAT');
[success,SensorData] = Thesis_IMU_H5toMatExtraction_4IMUs();

%% Store .h5 filenames
h5Dir = dir('*.h5');
filenames = struct('filename',{h5Dir(1:4).name});

% Use filename to determine which sensor it is
% 746 = shank, 711 = thigh, 2382 = waist, 2375 = marker
whichSensor = [746,0;711,0;2382,0;2375,0];
for c = 1:4
    currentFilename = filenames(c).filename;
    if contains(currentFilename,'746') == 1
        whichSensor(1,2) = c;
    else
        if contains(currentFilename,'711') == 1
            whichSensor(2,2) = c;
        else
            if contains(currentFilename,'2382') == 1
                whichSensor(3,2) = c;
            else
                whichSensor(4,2) = c;
            end
        end
    end
end

%% Organize data into recognizable variables

% In finalSensorData struct:
% 2 - acceleration
% 5 - ang velocity
% 8 - mag field
% 11 - quaternion
% 12 - marker status
```

```
% Time
dataLen = length(SensorData{1,2}(:,1));
time = zeros(dataLen,1);
for i = 2:dataLen
    time(i) = time(i-1) + 1/128;
end

% Shank - #746
shankData = SensorData(whichSensor(1,2),1:12);
AC_s = shankData{1,2};
AV_s = shankData{1,5};
MF_s = shankData{1,8};
QUAT_s = shankData{1,11};

% Thigh - #711
thighData = SensorData(whichSensor(2,2),1:12);
AC_t = thighData{1,2};
AV_t = thighData{1,5};
MF_t = thighData{1,8};
QUAT_t = thighData{1,11};

% Waist - #2382
waistData = SensorData(whichSensor(3,2),1:12);
AC_w = waistData{1,2};
AV_w = waistData{1,5};
MF_w = waistData{1,8};
QUAT_w = waistData{1,11};

% Marker - #2375
markerData = SensorData(whichSensor(4,2),1:12);
markerStatus = markerData{1,12};

rawAPDM =
table(time,AC_s,AV_s,MF_s,QUAT_s,AC_t,AV_t,MF_t,QUAT_t,AC_w,AV_w,MF_w,Q
UAT_w,markerStatus);

%% Save "rawAPDM" struct as "subjectID_rawAPDM.mat"
% in Processed Data/Subject Folder/APDM/MAT
```

### *1.1.2 Function to convert .apdm to .h5 to .mat*

```
function [success,finalSensorData] =
Thesis_IMU_H5toMatExtraction_4IMUs()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%  This program was created to open IMU .h5 files and convert to .mat
%%
%%
%%
%%
%%
%%  Created by: Ryan Chapman, Ph.D.
%%
%%  Created on: April 1, 2019
%%
%%
%%
%%  Edited by: Ryan Chapman, Ph.D.
%%
%%  Edited on: April 1, 2019
%%
%%  Edited by Megan McCabe in Feb 2020
%%
%%
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

%% Open directory of raw .apdm files to write .h5 files

codeDirectory = '/Users/megan_mccabe/Desktop/Thesis/Data Pre-
Processing/APDM Pre-Processing/RMC APDM to MAT';
codeFolder = strcat(codeDirectory,'/poi_library/');
jar_1 = strcat(codeFolder,'poi-3.8-20120326.jar');
jar_2 = strcat(codeFolder,'poi-ooxml-3.8-20120326.jar');
jar_3 = strcat(codeFolder,'poi-ooxml-schemas-3.8-20120326.jar');
jar_4 = strcat(codeFolder,'xmlbeans-2.3.0.jar');
jar_5 = strcat(codeFolder,'dom4j-1.6.1.jar');

format long;
javaaddpath(jar_1);
javaaddpath(jar_2);
javaaddpath(jar_3);
javaaddpath(jar_4);
javaaddpath(jar_5);

h1 = msgbox('Please select the .apdm folder.');
uiwait(h1);
apdmDirectory = uigetdir('/Users/megan_mccabe/Desktop/Thesis/Data
Capture');
cd(apdmDirectory);
h5Dir = dir('*.h5');
if length(h5Dir) < 4
    apdmDir = dir('*.apdm');
    newProcessRaw(apdmDir);
end
```

```
h5Dir = dir('*.h5');

%% Open appropriate .h5 files and combine data
count = 1;
for i = 1:length(h5Dir)
    HDF5FileNames{i,1} = h5Dir(i).name;
    HDF5FileInfo{i,1} = h5info(h5Dir(i).name);
    HDF5FileGroups{i,1} = HDF5FileInfo{i,1}.Groups;
    for j = 1:length(HDF5FileGroups{i,1})
        HDF5Names{count,1} = count;
        HDF5Names{count,2} = HDF5FileGroups{i,1}(j).Name;
        count = count + 1;
    end
end

%% Load sensor data
sensorData = importH5Data(HDF5FileNames,HDF5Names);
if length(HDF5FileNames) == 1;
    for i = 1:length(HDF5Names)
        sampleRate(i,1) =
double(HDF5FileGroups{1,1}(i).Attributes(2).Value);
        magneticField(i,1) =
double(HDF5FileGroups{1,1}(i).Groups.Datasets(3).Attributes(2).Value);
        gravity(i,1) =
double(HDF5FileGroups{1,1}(i).Groups.Datasets(1).Attributes(3).Value);
    end
else
    for i = 1:length(HDF5Names)
        sampleRate(i,1) =
double(HDF5FileGroups{i,1}.Attributes(2).Value);
        magneticField(i,1) =
double(HDF5FileGroups{i,1}.Groups(1).Datasets(3).Attributes(2).Value);
        gravity(i,1) =
double(HDF5FileGroups{i,1}.Groups(1).Datasets(1).Attributes(2).Value);
    end
end

finalSampleRate = mean(sampleRate(:,1));
finalMagField = mean(magneticField(:,1));
finalGravity = mean(gravity(:,1));

finalSensorData = syncedDataLoader_4IMUs(sensorData,finalSampleRate);

%% Convert data to doubles
[row,col] = size(finalSensorData);
for i = 1:row
    for j = 1:11
        finalSensorData{i,j} = double(finalSensorData{i,j});
    end
end

success = 'Woohoo you did it.'
```

*1.2 Filter APDM data*

```
% A2) FilterAPDM.m
%
% This script filters raw APDM data and plots it with the marker "true"
% indicator.
%
% Megan McCabe
% Jan/Feb 2020
% Edited April 2020 to check remove NaNs and interpolate
%

%% Load subjectID_rawAPDM.m into workspace
% Pre-processed data/Subject folder/APDM/MAT

%% Check for NaNs, remove and interpolate
rawAPDM(:,2:13) = checkNaNs(rawAPDM(:,2:13));

%% Filter data
% Create desired filter (low pass butterworth)
cutoff = 10; % cutoff frequency = 10 Hz (same as Loadsol & used in
Sensors 2019 paper by KAIST)
sampleRate = 128;
Wn = cutoff / (sampleRate / 2);
[B,A] = butter(5,Wn);

numVars = width(rawAPDM) - 2; % subtract time & marker status
filtAPDM = rawAPDM;
for i = 2:numVars+1
    currentData = rawAPDM{:,i};
    newData = currentData;
    for j = 1:3
        newData(:,j) = filtfilt(B,A,currentData(:,j));
    end
    filtAPDM{:,i} = newData;
end

% Create lower cutoff filter for syncing data (2.3 Hz for shank AV
data)
cutoff2 = 2.3; % cutoff frequency = 2.3 Hz
Wn2 = cutoff2 / (sampleRate / 2);
[B2,A2] = butter(5,Wn2);

% Add to filtAPDM table:
shankAV = rawAPDM.AV_s;
syncAV =
[filtfilt(B2,A2,shankAV(:,1)),filtfilt(B2,A2,shankAV(:,2)),filtfilt(B2,
A2,shankAV(:,3))];
filtAPDM = addvars(filtAPDM,syncAV);

% Check Filter
% t = rawAPDM.time;
% raw = rawAPDM.AC_s;
% filt = filtAPDM.AC_s;
% plot(t,raw(:,1),'r',t,filt(:,1),'b');
% xlim([200,300]);
```

```
%% Plot data for splitting

% Find indices which correspond with me pushing the apdm button
markerTrue_indices = find(rawAPDM.markerStatus(:,1));

% First 3 digits of index tell you how to group the indices -
% but I only want the first of those groups

justFirst3Digits = floor(markerTrue_indices/1000);
firstIndices = zeros(50,1);
firstIndices(1) = markerTrue_indices(1);
count = 1;
for i = 2:length(markerTrue_indices)
    current3digs = justFirst3Digits(i);
    currentIndex = markerTrue_indices(i);
    if current3digs ~= justFirst3Digits(i-1)
        count = count + 1;
        firstIndices(count) = currentIndex;
    end
end

firstIndices = nonzeros(firstIndices);

% Break file up by activity type first
% plot thigh x acc, shank z av
thighXacc = filtAPDM.AC_t(:,1);
shankZav = filtAPDM.AV_s(:,3);
t = filtAPDM.time;

buttonPush = zeros(length(firstIndices),1);
yyaxis left
plot(t,thighXacc,t(transpose(firstIndices)),buttonPush,'ko','markerSize
',10,'MarkerFaceColor','k');
title('Differentiate by activities');
xlabel('time (s)');
ylabel('Thigh X AC (m/s)');

yyaxis right
plot(t,shankZav);
xlabel('time (s)');
ylabel('Shank Z AV (rad/s)');

buttonPush_times = t(firstIndices);

%% Save filtAPDM as "SubjectID_filtAPDM_date.mat"

%% Copy paste the buttonPush_times into an excel doc and label using
plot
% Then format that into the "SubjectID_formattedButtonTracker.xlsx" for
% input into A3_trialSplitter.m
```

*1.3 Split logged APDM data by activity trial*

```
% A3) TrialSplitter.m
%
% This script splits APDM data into separate activities/trials.
%
% Megan McCabe
% March 2020

% Create structure to store the trial/activity data separately
temp = {'stand';'sit';'sit2stand-1';'stand2sit-1';'sit2stand-2';...
    'stand2sit-2';'sit2stand-3';'stand2sit-3';'stepup-1';'stepdown-
1';...
    'stepup-2';'stepdown-2';'stepup-3';'stepdown-3';'walk-1.5-1';...
    'walk-1.5-2';'walk-1.5-3';'walk-2-1';'walk-2-2';'walk-2-3';...
    'walk-2.5-1';'walk-2.5-2';'walk-2.5-3';'stair-6-1';'stair-6-2';...
    'stair-6-3';'stair-8-1';'stair-8-2';'stair-8-3';'stair-10-1';...
    'stair-10-2';'stair-10-3'};

for i = 1:32
   apdmData(i).filename = temp{i};
end

% Import .xlsx file with marker times into matlab as well as matlab
% tabulated apdm data
h1 = msgbox('Please select the file with the formatted apdm marker
tracking data.');
uiwait(h1);
[file,path] = uigetfile('*.xlsx');
filepath = [path,file];
markerTimes = xlsread(filepath,'C5:D36');

h2 = msgbox('Please select the file with filtered apdm data.');
uiwait(h2);
[file2,path2] = uigetfile('*.mat');
filepath2 = [path2,file2];
load(filepath2);

time = filtAPDM.time;
for i = 1:32
    startTime = markerTimes(i,1);
    stopTime = markerTimes(i,2);

    startIndex = find(abs(time-startTime)<0.5/128);
    stopIndex = find(abs(time-stopTime)<0.5/128);

    apdmData(i).data = filtAPDM(startIndex:stopIndex,:);
end

%% Save apdmData as "SubjectID_splitAPDM.mat"
```

*1.4 Identify "priority" trials for processing*

```
% Load SubjectID_sepAPDM.mat

% Change indices depending on which mocap files you edited for this
subject
FiltAPDM = apdmData([1, 9, 19, 29]);

%% Save as FiltAPDM as "SubjectID_splitAPDM_date_priorityFiles.mat"
```

## 2. Loadsol Pre-Processing

### *2.1 Import Loadsol*

```
% A1) ImportLoadsol.m
%
% This script imports the force pad data from text files into matlab
and
% saves the data as .m files (tables).
%
% Megan McCabe
% Oct 22, 2019

%% Create folder for loadsol priority files before running code
% Only import those files

% Ask user to find folder with .txt files from force pads
h1 = msgbox('Please select the folder with the Loadsol data.');
uiwait(h1);
FolderPath = uigetdir('/Users/megan_mccabe/Desktop/Thesis/Data
Capture');
FolderContents = dir(FolderPath);

% Check for ghost files in directory - delete them
bytes = [FolderContents.bytes];
numFiles = length(bytes);
NumGhosts = zeros(5,0);
indexNumGhosts = 1;
for a = 1:numFiles
    fileName = FolderContents(a).name;
    if strcmp(fileName(1),' ') || strcmp(fileName(1),'.')
        NumGhosts(indexNumGhosts) = a;
        indexNumGhosts = indexNumGhosts + 1;
    end
end

[lastGhostIndex, junk] = max(NumGhosts);

% Pull Force data from folder and store it in struct, 'ForceFiles'
for b = lastGhostIndex+1:numFiles
    fileName = FolderContents(b).name;
    fileName(end-25:end) = [];
    fullFileName = strcat(FolderPath,'/',FolderContents(b).name);
    opts = detectImportOptions(fullFileName);
    data = readtable(fullFileName,opts);
    ForceFiles(b-lastGhostIndex).filename = fileName;
    ForceFiles(b-lastGhostIndex).data = data;
end

% Change the name of the table variables to make them more easily
% recognizable
[junk, lenStruct] = size(ForceFiles);

for a = 1:lenStruct
    T = ForceFiles(a).data;
```

```
    T.Properties.VariableNames =
{'L_Time_s','L_Force_N','R_Time_s','R_Force_N','junk'};
    ForceFiles(a).data = T;
end

%% Save ForceFiles as "SubjectID_rawLoadsol_date.mat"
```

*2.2 DC-shift Loadsol data*

```
% DCshift.m
%
% This script corrects for a small DC shift in one of the loadsol
sensors.
%
% Megan McCabe
% March 2020

%% Load the SubjectID_filtLoadsol.mat file (filtLoadsol)

oldLoadsol = FiltLoadsol;
[~,numFiles] = size(FiltLoadsol);

whichFoot = input('Which foot needs a DC shift, right (1) or left (0)?:
');

% Find index of standing file
for i = 1:numFiles
    if contains(FiltLoadsol(i).filename,'stand')
        standIndex = i;
    elseif contains(FiltLoadsol(i).filename,'walk')
        walkIndex = i;
    end
end

% Use unloaded portion of data as DC shift for each file (except
standing
% file)
for i = 1:numFiles
    if i ~= standIndex
        currentData = FiltLoadsol(i).data;
        time = currentData(:,1);
        if whichFoot == 0
            problemForce = currentData(:,2);
            correctforce = currentData(:,3);
        else
            problemForce = currentData(:,3);
            correctForce = currentData(:,2);
        end

        figure(1);
        plot(time,problemForce);
        title('Identify unloaded section of data');
        xlim([0 5]);
        clicks = ginput(2);
        startTime = clicks(1,1);
        stopTime = clicks(2,1);
        startIndex = find(abs(time - startTime(1))<0.5/128);
        stopIndex = find(abs(time - stopTime(1))<0.5/128);
        unloadedAvg = mean(problemForce(startIndex:stopIndex));

        if i == walkIndex
            walkUnloaded = unloadedAvg;
        end
```

```
        if whichFoot == 0
            currentData(:,2) = currentData(:,2) - unloadedAvg;
        else
            currentData(:,3) = currentData(:,3) - unloadedAvg;
        end

        FiltLoadsol(i).data = currentData;
    end
end

% Use unloaded portion of walking file to shift standing file
currentData = FiltLoadsol(standIndex).data;
time = currentData(:,1);
if whichFoot == 0
    problemForce = currentData(:,2);
    correctforce = currentData(:,3);
else
    problemForce = currentData(:,3);
    correctForce = currentData(:,2);
end

if whichFoot == 0
    currentData(:,2) = currentData(:,2) - walkUnloaded;
else
    currentData(:,3) = currentData(:,3) - walkUnloaded;
end

FiltLoadsol(standIndex).data = currentData;

% check code
for i = 1:numFiles
    currentData_shifted = FiltLoadsol(i).data;
    currentData_noShift = oldLoadsol(i).data;
    time = currentData_shifted(:,1);
    if whichFoot == 0
        force = currentData_shifted(:,2);
        forceNoShift = currentData_noShift(:,2);
        correctForce = currentData_noShift(:,3);
    else
        force = currentData_shifted(:,3);
        forceNoShift = currentData_noShift(:,3);
        correctForce = currentData_noShift(:,2);
    end

    figure(i+1);
    plot(time,forceNoShift,'r',time,force,'b',time,correctForce,'k');
    legend('old','new','correct');
    ylabel('Force (N)');
    xlabel('Time (s)');

end

%% Save FiltLoadsol
```

*2.3 Filter Loadsol data*

```
% A2) FilterLoadsol.m
%
% This script 1) Filters data using a low pass Butterworth filter
(cutoff freq =
% 10 Hz - recommended for loadsol 100 Hz says Sensor 2019 article by
% Renner) 2) Resamples force data to 128 Hz to match IMUs and MOCAP
%
% Megan McCabe
% created Dec 2019 mod March 2020

% Ask user to identify raw force file
h1 = msgbox('Please select the SubjectID_rawLoadsol.mat file:');
uiwait(h1);
cd '/Users/megan_mccabe/Desktop/Thesis/Data Pre-Processing/Processed
Data'
[filename,path] = uigetfile('*.mat');
fullfilename = [path,filename];
load(fullfilename);
cd '/Users/megan_mccabe/Desktop/Thesis/Data Pre-Processing/Loadsol
Processing Code'

%% Add in code here to delete rows with NaNs from the force data

[junk, numFiles] = size(ForceFiles);

for i = 1:numFiles
    currentData = ForceFiles(i).data;
    currentData = currentData{:,1:4};

    findNaNs = isnan(currentData);
    L = findNaNs(:,2);
    R = findNaNs(:,4);

    % Check that you do not need to interpolate (all NaNs occur at the
end
    % or beginning of the time series)
    NaNindices{i,1} = find(L);
    NaNindices{i,2} = find(R);

    currentDataLen = length(L);
    for j = currentDataLen:-1:1
        if (L(j) == 1 || R(j) == 1)
            currentData(j,:) = [];
        end
    end

    ForceFiles(i).data = [];
    ForceFiles(i).data = currentData;

end

%% Convert ForceFile table variable names

for a = 1:numFiles
```

```
    currentData = ForceFiles(a).data;
    L_Time_s = currentData(:,1);
    L_Force_N = currentData(:,2);
    R_Time_s = currentData(:,3);
    R_Force_N = currentData(:,4);
    T = table(L_Time_s,L_Force_N,R_Time_s,R_Force_N);
    ForceFiles(a).data = T;
end


%%  Filter and resample force data to 128 Hz

FiltLoadsol = ForceFiles;
for b = 1:numFiles
    currentdata = ForceFiles(b).data;
    currentfile = ForceFiles(b).filename;
    time = currentdata.L_Time_s;
    Fl = currentdata.L_Force_N;
    Fr = currentdata.R_Force_N;

    cutoff = 10; % cutoff freq 10 Hz
    sampleRate = 100; % sampling freq 100 Hz
    Wn = cutoff / (sampleRate / 2);
    [B,A] = butter(5,Wn);
    Fl_filt = filtfilt(B,A,Fl);
    Fr_filt = filtfilt(B,A,Fr);

    desiredTime = linspace(0,time(end),round(time(end)*128,0));
    desiredTime = transpose(desiredTime);
    L_splined = spline(time,Fl_filt,desiredTime);
    R_splined = spline(time,Fr_filt,desiredTime);
    FiltLoadsol(b).data = [desiredTime,L_splined,R_splined];

%     % Code to plot raw versus filtered data
%     figure(b);
%     plot(time,Fl_filt,time,Fr_filt);
%     title(currentfile);
%     legend('Left','Right');
%     xlabel('Time (s)');
%     ylabel('Force (N)');

end

%% ** Save "FiltLoadsol" as "SubjectID_filtForceFiles.m" **
% filePath: Thesis/Data Pre-Processing/Processed Data/Subject
Folder/Loadsol/MAT
```

## *2.4 Export Loadsol data into .mot file*

### *2.4.1 Script*

```
% A3) ExportGRFs.m
%
% This script determines an appropriate application point for the force
% data (midpoint x-axis position between ankle MOCAP markers) and
exports
% .mot files for import into OpenSim.
%
% Megan McCabe
% Jan 2020

%% ** Load "SubjectID_SyncedData.m" (aka SyncedData in workspace)
% in Processed Data/Subject Folder/

%% Determine application point for GRFs (~ midpoint of talus, like
CalPoly Masters students)

[~,numFiles] = size(SyncedData);
for i = 1:numFiles
    MOCAP = SyncedData(i).MOCAPdata.data;
    mrkrNames = SyncedData(i).MOCAPdata.markerNames;

    RAnkleIn = mrkrDataFinder(mrkrNames,'RAnkleIn',MOCAP);
    RAnkleOut = mrkrDataFinder(mrkrNames,'RAnkleOut',MOCAP);
    LAnkleIn = mrkrDataFinder(mrkrNames,'LAnkleIn',MOCAP);
    LAnkleOut = mrkrDataFinder(mrkrNames,'LAnkleOut',MOCAP);

    AP_l = LAnkleIn/2+LAnkleOut/2;
    AP_r = RAnkleIn/2+RAnkleOut/2;

    SyncedData(i).AP = table(AP_l,AP_r);
end

%% Write GRF .mot files for import into OpenSim

subject_id = input('Please enter subject identifier: ','s');

%% Write .trc files for each file
[~,numFiles] = size(SyncedData);
for i = 1:numFiles
    currentFileName = SyncedData(i).filename;
    currentData = SyncedData(i).Loadsoldata;
    AP = SyncedData(i).AP;
    AP = AP{:,:};
    currentData(:,3:8) = AP;

    time = SyncedData(i).time;

    motFilename = strcat(subject_id,'_',currentFileName,'_grf.mot');
    fileID = fopen(motFilename,'w');

    success = GRFmotWriter(fileID,motFilename,time,currentData);
```

```
end

%% Move .mot files that result from current code folder to:
% Processed Data/Subject Folder/Loadsol

% Create xml files to match each mot file
```

### *2.4.2 Function to identify MOCAP markers by name*

```
function [mrkrData] = mrkrDataFinder(mrkrNames,findName,data)

for m = 1:19
    if (strcmp(mrkrNames{m},findName) == 1)
        index = m;
        start = (index-1)*3+1;
        stop = start+2;
        mrkrData = data(:,start:stop);
    end
end

end
```

### *2.4.3 Function to write .mot files*

```
function [success] = GRFmotWriter(fileID,filename,time,data)
%
% This function writes GRF .mot files for import into OpenSim
%
% Megan McCabe
% Jan 2020
%

    % data = L_force,R_force,L_AP(x,y,z),R_AP(x,y,z)

    % Determine size of file
    nRows = length(data(:,1));

    % Create file header
    fprintf(fileID, '%s\n', filename);
    fprintf(fileID,'version=1\n');
    fprintf(fileID,'nRows=%d\n',round(nRows,0));
    fprintf(fileID,'nColumns=19\n');
    fprintf(fileID,'inDegrees=yes\n');
    fprintf(fileID,'endheader\n');

fprintf(fileID,'time\tground_force_vx\tground_force_vy\tground_force_vz
\t');

fprintf(fileID,'ground_force_px\tground_force_py\tground_force_pz\t');


fprintf(fileID,'1_ground_force_vx\t1_ground_force_vy\t1_ground_force_vz
\t');

fprintf(fileID,'1_ground_force_px\t1_ground_force_py\t1_ground_force_pz
\t');


fprintf(fileID,'ground_torque_x\tground_torque_y\tground_torque_z\t');

fprintf(fileID,'1_ground_torque_x\t1_ground_torque_y\t1_ground_torque_z
\n');

    % Write data into file
    L_force = data(:,1);
    R_force = data(:,2);
    AP_L = data(:,3:5);
    AP_R = data(:,6:8);

    for b = 1:nRows

fprintf(fileID,'%f\t0\t%f\t0\t%f\t0\t%f\t',time(b),R_force(b),AP_L(b,1)
,AP_L(b,3));

fprintf(fileID,'0\t%f\t0\t%f\t0\t%f\t',L_force(b),AP_R(b,1),AP_R(b,3));
        fprintf(fileID,'0\t0\t0\t0\t0\t0\n');
    end
```

```
    fclose(fileID);

    success = 'success';

end
```

## 3. MOCAP Pre-Processing

### *3.1 Import MOCAP*

```
% A1) ImportMOCAP_csvVersion_2_13.m
%
% This script imports MOCAP data from .trc files and stores it in a
structure.
%
% Megan McCabe
% Jan 2020

%% All .csv files to be processed should be stored in one folder
% Separate priority files in their own folder!

% Ask user to identify folder with .csv files
h1 = msgbox('Please select the folder with the .csv files');
uiwait(h1);
FolderPath = uigetdir('/Users/megan_mccabe/Desktop/Thesis/Data
Capture/Raw Data');
FolderContents = dir(FolderPath);

% Check for ghost files in directory - delete them
bytes = [FolderContents.bytes];
numFiles = length(bytes);
NumGhosts = zeros(5,0);
indexNumGhosts = 1;
for a = 1:numFiles
    fileName = FolderContents(a).name;
    if strcmp(fileName(1),' ') || strcmp(fileName(1),'.')
        NumGhosts(indexNumGhosts) = a;
        indexNumGhosts = indexNumGhosts + 1;
    end
end

[lastGhostIndex, junk] = max(NumGhosts);

% Ask user for subject ID:
subjectID = input('Please enter subject ID: ','s');
IDlen = length(subjectID);

% Pull MOCAP data from folder and store it in struct, 'RawMOCAP'
for b = lastGhostIndex+1:numFiles

    fileName = FolderContents(b).name;
    fileName(end-3:end) = [];
    fullFileName = strcat(FolderPath,'/',FolderContents(b).name);

    trcData.data = csvread(fullFileName,7,0);
    tableData = readtable(fullFileName);

    % Make sure we know which row has the marker labels
    markersStart = tableData{:,3};
    lenTable = length(markersStart(:,1));
    for c = 1:lenTable
```

```
        currentName1 = markersStart{c,1};
        lenName = length(currentName1);
        if lenName > IDlen
            if strcmp(currentName1(1,1:IDlen),subjectID)
                mrkr_startRow = c;
            end
        end
    end

    markers = tableData{mrkr_startRow,3:59};
    justNames =
{'AA';'AA';'AA';'AA';'AA';'AA';'AA';'AA';'AA';'AA';'AA';'AA';'AA';'AA';
'AA';'AA';'AA';'AA';'AA'};
    for i = 1:19
        index = (i-1)*3+1;
        currentName = markers{index};
        justNames{i} = currentName(IDlen+2:end);
    end
    trcData.markerNames = justNames;

    RawMOCAP(b-lastGhostIndex).filename = fileName;
    RawMOCAP(b-lastGhostIndex).data = trcData;

end

%% Save 'RawMOCAP' as 'SubjectID_rawMOCAP.m'
% to (Thesis/Data Pre-Processing/Processed Data/Subject ID/MOCAP/MAT)
```

## *3.2 Export MOCAP to .trc format*

### *3.2.1 Script*

```
% A2) ExportMOCAP.m
%
% Input: Synced data structure
% Function:
%       1) Maps Optitrack marker names to OpenSim ones
%       2) Writes .trc files for import into OpenSim
% Output: .trc files with MOCAP data ready for OpenSim
%
% Megan McCabe
% Feb 2020 mod March 2020
%

%% Load 'SyncedData.m' (aka "SubjectID_SyncedData.m")
% in Processed Data/Subject Folder

subject_id = input('Please enter subject identifier: ','s');

%% Write .trc files for each file
[~,numFiles] = size(SyncedData);
for i = 1:numFiles
    currentFileName = SyncedData(i).filename;
    currentData = SyncedData(i).MOCAPdata.data;
    currentMrkrNames = SyncedData(i).MOCAPdata.markerNames;
    mrkrs = currentMrkrNames;

    % Swap Optitrack marker names for OpenSim ones
    Optitrack =
{'LAnkleIn','LAnkleOut','LHeel','LKneeIn','LKneeOut',...
        'LShin','LThigh','LToeIn','RAnkleIn','RAnkleOut','RHeel',...
        'RKneeIn','RKneeOut','RShin','RThigh','RToeIn','WaistBack',...
        'WaistLFront','WaistRFront'};
    Opti_nums = 1:19;
    Opti_map = containers.Map(Optitrack,Opti_nums);

    OpenSim =
{'L.Ankle.Med','L.Ankle.Lat','L.Heel','L.Knee.Med','L.Knee.Lat',...

'L.Shank.Rear','L.Thigh.Upper','L.Toe.Med','R.Ankle.Med','R.Ankle.Lat',
...

'R.Heel','R.Knee.Med','R.Knee.Lat','R.Shank.Rear','R.Thigh.Upper',...
        'R.Toe.Med','V.Sacral','L.ASIS','R.ASIS'};
    OpenSim_nums = 1:19;
    OS_map = containers.Map(OpenSim_nums,OpenSim);

    mrkr_num = zeros(19,1);
    for a = 1:19
        mrkr_num(a) = Opti_map(currentMrkrNames{a,1});
        mrkrs{a,1} = OS_map(mrkr_num(a));
    end

    time = SyncedData(i).time;
```

```
    trcFilename = strcat(subject_id,'_',currentFileName,'.trc');
    fileID = fopen(trcFilename,'w');

    success =
trcWriter_2_13(fileID,trcFilename,time,currentData,mrkrs);

end

%% Move .trc files that result from current code folder to:
% Processed Data/Subject Folder/MOCAP/trc
```

### *3.2.2 Function to write .trc files*

```
function [success] =
trcWriter_2_13(fileID,filename,time,MOCAPdata,mrkrNames)
%
% This function writes .trc files from synced MOCAP data for import
into
% OpenSim.
%
% Megan McCabe
% Feb 2020
%

    %% Create file header
    numFrames = length(time);
    FrameCount = zeros(numFrames,1);
    for i = 2:numFrames
        FrameCount(i) = FrameCount(i-1) + 1;
    end

    % line 1
    fprintf(fileID,'PathFileType\t4\t(X/Y/Z)\t');
    % fprintf(fileID,'/Users/megan_mccabe/Desktop/Thesis/Data Pre-
Processing/Processed Data\t');
    fprintf(fileID,'%s\n',filename);

    % line 2

fprintf(fileID,'DataRate\tCameraRate\tNumFrames\tNumMarkers\tUnits\tOri
gDataRate\t');
    fprintf(fileID,'OrigDataStartFrame\tOrigNumFrames\n');

    % line 3
    numFrames_str = int2str(numFrames);

fprintf(fileID,'128\t128\t%s\t19\tm\t128\t0\t%s\n',numFrames_str,numFra
mes_str);

    % line 4
    fprintf(fileID,'Frame#\tTime\t');
    for m = 1:19
        fprintf(fileID,'%s\t\t\t',mrkrNames{m});
    end
    fprintf(fileID,'\n\t\t');

    % line 5
    numMarkers = zeros(19,1);
    numMarkers(1) = 1;
    for i = 2:19
        numMarkers(i) = numMarkers(i-1)+1;
    end
    for j = 1:18
        A = int2str(numMarkers(j));
        fprintf(fileID,'X%s\tY%s\tZ%s\t',A,A,A);
    end
    fprintf(fileID,'X19\tY19\tZ19\t\n\n');
```

```
    %% Write data

    allData = [FrameCount,time,MOCAPdata];
    numCols = length(allData(1,:));
    for i = 1:numFrames
        for j=1:numCols-1
            if mod(j-1,59) == 0
                fprintf(fileID,'%d\t',round(allData(i,j),0));
            else
                fprintf(fileID,'%f\t',allData(i,j));
            end

        end
        fprintf(fileID,'%f\n',allData(i,59));
    end

    fclose(fileID);
    success = 'success';

end
```

## 4. Syncing

### *4.1 Main Script*

```
% A1) SyncAllSensorData.m
%
% This script syncs all sensor data.
%
% Megan McCabe
% Jan 2020
% Updated Feb 2020
%

%% Before running script, load 1) MOCAP, 2) Loadsol, 3) APDM data
mainFilepath = '/Users/megan_mccabe/Desktop/Thesis/Data Pre-
Processing/Processed Data';

h1 = msgbox('Please select the SubjectID_rawMarkerData.mat');
uiwait(h1);
[filename_mo, filepath_mo] = uigetfile(mainFilepath,'*.mat');
filepath_mo = strcat(filepath_mo,filename_mo);
load(filepath_mo);

h2 = msgbox('Please select the SubjectID_filtForceFiles.mat');
uiwait(h2);
[filename_lo, filepath_lo] = uigetfile(mainFilepath,'*.mat');
filepath_lo = strcat(filepath_lo,filename_lo);
load(filepath_lo);

h3 = msgbox('Please select the
SubjectID_splitAPDM_date_priorityFiles.mat');
uiwait(h3);
[filename_ap, filepath_ap] = uigetfile(mainFilepath,'*.mat');
filepath_ap = strcat(filepath_ap,filename_ap);
load(filepath_ap);

%% Check to make sure that all files are present

[~, numFiles_mo] = size(RawMOCAP);
[~, numFiles_lo] = size(FiltLoadsol);
[~, numFiles_ap] = size(FiltAPDM);

if (numFiles_mo == numFiles_lo && numFiles_mo  == numFiles_ap) ~= 1
    disp('Number of files is not consistent');
    return
end

%% Check if subject is right or left footed

footedness = input('Is subject right (1) or left (0) footed?: ');

%% Ask for angle of rotation of shank (alpha)

alpha = input('What is the angle of rotation (in deg)? ');
alpha = alpha/180*pi;
```

```
rotMatrix = [1,0,0;0,cos(alpha),-sin(alpha);0,sin(alpha),cos(alpha)];


%% Match sensor data based on filenames

NameMatchesIndices = zeros(numFiles_mo,3); % 1) MO 2) LO 3) AP
for i = 1:numFiles_mo
    currentNameToMatch = RawMOCAP(i).filename;
    NameMatchesIndices(i,1) = i;

    for j = 1:numFiles_lo
        currentName2 = FiltLoadsol(j).filename;
        if strcmp(currentName2,currentNameToMatch) == 1
            NameMatchesIndices(i,2) = j;
        end
    end

    for k = 1:numFiles_ap
        currentName3 = FiltAPDM(k).filename;
        if strcmp(currentName3,currentNameToMatch) == 1
            NameMatchesIndices(i,3) = k;
        end
    end
end

%% Plot data and identify syncing point
numSubplots = 3;

for i = 1:numFiles_mo
    currentFileName = RawMOCAP(i).filename;

    % Determine whether file is a gait file - if so plot & sync
    GaitFileIndicator =
contains(currentFileName,["walk","stair","step"]);

    if GaitFileIndicator == 1

        figure(i);
        maxTime = 5;

        % extract and plot MOCAP data
        currentMOCAP = RawMOCAP(NameMatchesIndices(i,1)).data;
        data_mo = currentMOCAP.data;
        frames_mo = data_mo(:,1);
        data_mo(:,1:2) = [];
        mrkrNames = currentMOCAP.markerNames;

        for m = 1:19
            if footedness == 1
                if strcmp(mrkrNames{m},'RHeel') == 1
                    Heel_index = m;
                    start = (Heel_index-1)*3+1;
                    stop = start+2;
                    Heel = data_mo(:,start:stop);
                else
                    if strcmp(mrkrNames{m},'RToeIn') == 1
```

```
                    Toe_index = m;
                    start = (Toe_index-1)*3+1;
                    stop = start + 2;
                    Toe = data_mo(:,start:stop);
                end
            end

        else
            if strcmp(mrkrNames{m},'LHeel') == 1
                Heel_index = m;
                start = (Heel_index-1)*3+1;
                stop = start+2;
                Heel = data_mo(:,start:stop);
            else
                if strcmp(mrkrNames{m},'LToeIn') == 1
                    Toe_index = m;
                    start = (Toe_index-1)*3+1;
                    stop = start + 2;
                    Toe = data_mo(:,start:stop);
                end
            end
        end

        if strcmp(mrkrNames{m},'WaistBack') == 1
            WaistBack_index = m;
            start = (WaistBack_index-1)*3+1;
            stop = start+2;
            WaistBack = data_mo(:,start:stop);
        end
    end

    lenData_mo = length(data_mo(:,1));
    t_mo = zeros(lenData_mo,1);
    for m = 2:lenData_mo
        t_mo(m) = t_mo(m-1) + 1/128;
    end

    % shift the data to account for shortening of the mocap file
during
    % editing:
    frames_mo = frames_mo + 1; % shift frame count to equal indices
    if frames_mo(1) > 1
        frameShift_mo = frames_mo(1);
    else
        frameShift_mo = 1;
    end

    v_heel_z = diff(Heel(:,3));
    v_toe_y = diff(Toe(:,2));
    p_heel_z = (Heel(:,3) - WaistBack(:,3)); % assumes subject is
walking in 'z' direction!

    subplot(numSubplots,1,1);
    yyaxis left
    plot(t_mo,p_heel_z);
    ylabel('Z_{heel} - Z_{waistback} (m)');
    yyaxis right
```

```
plot(t_mo(1:end-1),v_heel_z,t_mo(1:end-1),v_toe_y);
ylabel('Velocity (m/s)');
legend('Z Heel Pos','Z Heel Vel','Y Toe Vel');
title(currentFileName);
xlabel('time (s)');
if t_mo(end) > maxTime
    xlim([0,maxTime]);
end

% extract & plot Loadsol data
data_lo = FiltLoadsol(NameMatchesIndices(i,2)).data;
data_lo = data_lo(frameShift_mo:end,:);
data_lo(:,1) = []; % delete time
if footedness == 1
    force = data_lo(:,2); % Right
else
    force = data_lo(:,1); % Left
end

dataLen_lo = length(force);
t_lo = zeros(dataLen_lo,1);
for z = 2:dataLen_lo
    t_lo(z) = t_lo(z-1) + 1/128;
end
threshold = 20*ones(length(t_lo),1);
subplot(numSubplots,1,2);
plot(t_lo,force,t_lo,threshold,'--r');
xlabel('time (s)');
ylabel('force (N)');
if t_lo(end) > maxTime
    xlim([0,maxTime]);
end

% extract & plot APDM data
data_ap = FiltAPDM(NameMatchesIndices(i,3)).data;
data_ap = data_ap(frameShift_mo:end,:);
shankAV = data_ap{:,15};

dataLen_ap = length(shankAV(:,1));
t_ap = zeros(dataLen_ap,1);
t_ap(1) = 0;
for j = 2:dataLen_ap
    t_ap(j) = t_ap(j-1)+1/128;
end

 % rotate APDM data
shankAV_rotated = zeros(dataLen_ap,3);
for j = 1:dataLen_ap
    shankAV_rotated(j,:) = shankAV(j,:)*rotMatrix;
end

subplot(numSubplots,1,3);
plot(t_ap,shankAV_rotated(:,3));
xlabel('time (s)');
ylabel('shank \omega_Z (rad/s)');
if t_ap(end) > maxTime
    xlim([0,maxTime]);
```

```
end

figure(5);
plot(t_ap,shankAV_rotated(:,3));
xlabel('time (s)');
ylabel('shank \omega_Z (rad/s)');
if t_ap(end) > maxTime
    xlim([0,maxTime]);
end

% Identify syncing points
h1 = msgbox('Please click on HS for the first three plots:');
uiwait(h1);
HS = ginput(3);

% Translate time vectors to sync (convert to t_mo)
syncTime_mo = HS(1,1);
syncTime_lo = HS(2,1);
syncTime_ap = HS(3,1);
t_lo = t_lo - (syncTime_lo - syncTime_mo);
t_ap = t_ap - (syncTime_ap - syncTime_mo);

% Convert syncTime to index on each of the 3 datasets
syncIndex_mo = find(abs(t_mo - syncTime_mo) < 0.5/128);
syncIndex_lo = find(abs(t_lo - syncTime_mo) < 0.5/128);
syncIndex_ap = find(abs(t_ap - syncTime_mo) < 0.5/128);

% Determine which sensor has shortest front end tail
shortestFront = min([syncIndex_mo;syncIndex_lo;syncIndex_ap]);
cutFront = [syncIndex_mo-shortestFront;...
    syncIndex_lo-shortestFront;...
    syncIndex_ap-shortestFront];
if cutFront(1) ~= 0
    data_mo(1:cutFront(1),:) = [];
end
if cutFront(2) ~= 0
    data_lo(1:cutFront(2),:) = [];
end
if cutFront(3) ~= 0
    data_ap(1:cutFront(3),:) = [];
end

% Determine which sensor has shortest back end tail
newDataLens =
[length(data_mo(:,1));length(data_lo(:,1));length(data_ap{:,1})];

shortestLen = min(newDataLens);

if newDataLens(1) > shortestLen
    data_mo(shortestLen+1:end,:) = [];
end
if newDataLens(2) > shortestLen
    data_lo(shortestLen+1:end,:) = [];
end
if newDataLens(3) > shortestLen
    data_ap(shortestLen+1:end,:) = [];
end
```

```
        dataLen_universal = length(data_lo(:,1));
        t_universal = zeros(dataLen_universal,1);
        for m = 2:dataLen_universal
            t_universal(m) = t_universal(m-1) + 1/128;
        end

        SyncedData(i).filename = currentFileName;
        SyncedData(i).time = t_universal;
        SyncedData(i).MOCAPdata.data = data_mo;
        SyncedData(i).MOCAPdata.markerNames = mrkrNames;
        SyncedData(i).Loadsoldata = data_lo;
        SyncedData(i).APDMdata = data_ap;

    else

        % extract MOCAP data
        currentMOCAP = RawMOCAP(NameMatchesIndices(i,1)).data;
        data_mo = currentMOCAP.data;
        data_mo(:,1:2) = [];
        mrkrNames = currentMOCAP.markerNames;

        lenData_mo = length(data_mo(:,1));
        t_mo = zeros(lenData_mo,1);
        for m = 2:lenData_mo
            t_mo(m) = t_mo(m-1) + 1/128;
        end

        % extract Loadsol data
        data_lo = FiltLoadsol(NameMatchesIndices(i,2)).data;
        t_lo = data_lo(:,1);
        data_lo(:,1) = [];
        if footedness == 1
            force = data_lo(:,2); % Right
        else
            force = data_lo(:,1); % Left
        end

        % extract APDM data
        data_ap = FiltAPDM(NameMatchesIndices(i,3)).data;
        shankAV = data_ap{:,9};

        dataLen_ap = length(shankAV(:,1));
        t_ap = zeros(dataLen_ap,1);
        t_ap(1) = 0;
        for j = 2:dataLen_ap
            t_ap(j) = t_ap(j-1)+1/128;
        end

        % Trim files on back end
        if (length(t_mo) < length(t_lo) && length(t_mo) < length(t_ap))
            data_lo(length(t_mo)+1:end,:) = [];
            data_ap(length(t_mo)+1:end,:) = [];
            t_universal = t_mo;
        else
```

```
            if (length(t_lo) < length(t_mo) && length(t_lo) <
length(t_ap))
                data_mo(length(t_lo)+1:end,:) = [];
                data_ap(length(t_lo)+1:end,:) = [];
                t_universal = t_lo;
            else
                data_lo(length(t_ap)+1:end,:) = [];
                data_mo(length(t_ap)+1:end,:) = [];
                t_universal = t_ap;
            end
        end

        SyncedData(i).filename = currentFileName;
        SyncedData(i).time = t_universal;
        SyncedData(i).MOCAPdata.data = data_mo;
        SyncedData(i).MOCAPdata.markerNames = mrkrNames;
        SyncedData(i).Loadsoldata = data_lo;
        SyncedData(i).APDMdata = data_ap;

    end

end

%% Attach gait % vector

for j = 1:numFiles_mo
    currentGRFs = SyncedData(j).Loadsoldata;
    currentTime = SyncedData(j).time;
    currentFileName = SyncedData(j).filename;

    % Determine whether file is a gait file - if so attach gait %
vector
    GaitFileIndicator = contains(currentFileName,["walk","stair"]);

    if GaitFileIndicator == 1
        L = currentGRFs(:,1);
        R = currentGRFs(:,2);
        time = currentTime;

        gaitCycleR = gaitCycleSplitter(time,R);
        gaitCycleL = gaitCycleSplitter(time,L);

        SyncedData(j).percentGaitR = gaitCycleR;
        SyncedData(j).percentGaitL = gaitCycleL;

    else
       SyncedData(j).percentGaitR = 1;
       SyncedData(j).percentGaitL = 1;
    end
end

%% Save 'SyncedData.m' as "SubjectID_syncedData_date.m"
% in Processed Data/Subject Folder
```

*4.2 Function to create gait percentage vector*

```
function [gaitCycle] = gaitCycleSplitter(time,force)

% This function identifies the swing stage (unloaded) portion of the
gait
% cycle and averages that force value

%% Modify this number if not all of the HS/TO points are being
identified!!
precision = 1.5;

% Find where function hits threshold 20N (min force picked up by
loadsol)
threshold = 20;
timeLen = length(time);
newTimeLen = round(timeLen * 1000/128,0); % resampling to 1000 Hz
tt = transpose(linspace(time(1),time(end),newTimeLen));
ff = spline(time,force,tt);
thresLocs = find(abs(ff-threshold)<precision);

% Determine whether the threshold is HS (+ slope) or TO (- slope)
numThres = length(thresLocs);
slopeAtThreshold = zeros(numThres,1);

halfSec = 50; % 0.05 sec = 50 indices
for i = 1:numThres
    currentThres = thresLocs(i);
    start = currentThres - halfSec;
    if start < 1
        start = 1;
    end
    stop = currentThres + halfSec;
    if stop > length(ff)
        stop = length(ff);
    end
    slopeAtThreshold(i) = (ff(stop)-ff(start))/(tt(stop)-tt(start));
end

HS_locs = thresLocs(slopeAtThreshold>0);
TO_locs = thresLocs(slopeAtThreshold<0);

% Find the original time points that best correspond with the HS time
% points identified in the resampled (1000 Hz) data
tt_HS = tt(HS_locs);
tt_TO = tt(TO_locs);

trep_matrix_HS = repmat(time,[1 length(tt_HS)]);
trep_matrix_TO = repmat(time,[1 length(tt_TO)]);
[~,hs128_indices] = min(abs(trep_matrix_HS-transpose(tt_HS)));
[~,to128_indices] = min(abs(trep_matrix_TO-transpose(tt_TO)));
hs128_indices = unique(transpose(hs128_indices));
to128_indices = unique(transpose(to128_indices));
numHS = length(hs128_indices);
numTO = length(to128_indices);
```

```
% Plot data & HS, TO locations to check code
figure;
yvalHS = 20*ones(numHS,1);
yvalTO = 20*ones(numTO,1);
P = plot(time,force,'b-
',time(hs128_indices),yvalHS,'om',time(to128_indices),yvalTO,'ok');
P(1).LineWidth = 2;
P(2).MarkerSize = 8;
P(2).MarkerFaceColor = 'm';
P(3).MarkerSize = 8;
P(3).MarkerFaceColor = 'k';
xlabel('time (s)');
ylabel('force (N)');
legend('force','HS','TO');
title('Check Code!');

% Construct gaitCycle variable that has the following columns:
    % 1: % of gait cycle
    % 2: 1 indicates stance stage
    % 3: 1 indicates swing stage

gaitCycle = [999*ones(length(force),1),zeros(length(force),2)];
for i = 2:numHS
    start = hs128_indices(i-1);
    stop = hs128_indices(i)-1;
    for j = 1:numTO
        if (to128_indices(j) < stop && to128_indices(j) > start)
            stanceEnd = to128_indices(j);
        end
    end

    cycleDuration_indices = stop - start + 1;
    cycleSteps = zeros(cycleDuration_indices,1);
    for k = 2:cycleDuration_indices
        cycleSteps(k) = cycleSteps(k-1) + 1/cycleDuration_indices *
100;
    end
    gaitCycle(start:stop,1) = cycleSteps;

    for l = start:stanceEnd-1
        gaitCycle(l,2) = 1;
    end

    for m = stanceEnd:stop
        gaitCycle(m,3) = 1;
    end

end

end
```

## 5. ANN Design

*5.1 Create and train ANN (10 rounds)*

```
% ANN_28_var_10_4.m
%
% Megan McCabe
% May 2020

%% Load ML data

if exist('allData','var') == 0
    filepath = '/Users/meganmccabe/Desktop/Thesis/Data Analysis/ML
Approach/Data Ready for ML/allData_5_6_no13.mat';
    load(filepath)
end

% Note about allData:
% row 1 - stair, row 2 - walk
% col 1 = header data, col 2 - inputs, col 3 - outputs

% Choose an activity
activity = input('Stair Ascent (1) or Walking (2)? ');

% collect activity data
header = allData{activity,1};
inputs = allData{activity,2};
outputs = allData{activity,3};
dataLen = length(inputs{:,1});

% remove waist IMU data for now - S011 doesn't have waist IMU data
(filled
% with 0s)
inputs.AC_w = [];
inputs.AV_w = [];
inputs.MF_w = [];
inputs.QUAT_w = [];

% Loop through rounds of training the ANN
subjectID = header{:,1};
numRounds = 10;
for i = 1:numRounds
    disp(['Iteration ',num2str(i)]);

    % 4 columns = t_flex, t_add, m_flex, m_add
    % 19 rows = 17 subjects + 2 rows for avg and std
    RMSE = zeros(18,4);
    rRMSE = zeros(18,4);
    Rsq = zeros(18,4);
    time = zeros(18,1);

    net = feedforwardnet([5,5]);
    in = transpose(inputs{:,:});
    out = transpose(outputs{:,[1,2,4,5]}); % remove transverse plane
```

```
    % Pre-process inputs/outputs to transform them to a [-1,1] range
    [in_norm,is] = mapminmax(in,-1,1);
    [on_norm,os] = mapminmax(out,-1,1);

     % Loop through subjects for LOO-CV
    for j = 1:16
       disp(['Subject ',num2str(j)]);
       % Determine validation subject given that S013 is excluded
       if j>=13
           validationSubject = j+1;
       else
           validationSubject = j;
       end
        validationSubject_indices =
transpose(find(subjectID==validationSubject));
        trainingSubject_indices =
transpose(find(subjectID~=validationSubject));
        [trainInd,valInd,testInd] =
divideind(dataLen,trainingSubject_indices,...
            validationSubject_indices,[]);
        net.divideFcn = 'divideind';
        net.divideParam.trainInd = trainInd;
        net.divideParam.valInd = valInd;

        % Train neural network
        [trainedNet,tr] = train(net,in_norm,on_norm);

        % Test trained model on validation data
        ival = in_norm(:,validationSubject_indices);
        oval = on_norm(:,validationSubject_indices);
        pval = trainedNet(ival);

        oval = mapminmax('reverse',oval,os);
        pval = mapminmax('reverse',pval,os);

        for k = 1:4
            is_c = oval(k,:);
            wa_c = pval(k,:);
            error = wa_c - is_c;
            rmse = sqrt(sum(error.^2)/length(error));
            RMSE(j,k) = rmse;
            rRMSE(j,k) = rmse/(0.5*sum([max(wa_c)-min(wa_c),max(is_c)-
min(is_c)]))*100;
            r = corrcoef(wa_c,is_c);
            Rsq(j,k) = (r(2,1))^2;
        end
        subjectResults{j,1} = [pval;oval];
        time(j) = tr.time(end);
    end

    % Save data from each round
    time(17:18) = [mean(time(1:16),1);std(time(1:16),0,1)];
    RMSE(17:18,:) = [mean(RMSE(1:16,:),1);std(RMSE(1:16,:),0,1)];
    rRMSE(17:18,:) = [mean(rRMSE(1:16,:),1);std(rRMSE(1:16,:),0,1)];
    Rsq(17:18,:) = [mean(Rsq(1:16,:),1);std(Rsq(1:16,:),0,1)];
    finalResults{i,1} = time;
    finalResults{i,2} = rRMSE;
```

```
    finalResults{i,3} = Rsq;
    finalResults{i,4} = RMSE;
    finalResults{i,5} = subjectResults;

    disp([rRMSE(17,:);Rsq(17,:)]);
end
```

*5.2 Average performance metrics across rounds*

```
% A2_ANN_averageRounds.m
%
% Megan McCabe
% May 2020

%% Load 'fullResults.mat'
load('SA_fullResults.mat');

nRounds = 10;

% Column 1: Time (t)
t = zeros(nRounds,2);
for i = 1:nRounds
    t(i,:) = [finalResults{i,1}(17),finalResults{i,1}(18)];
end

t_avg_acrossRounds = mean(t(:,1));
t_sd_acrossRounds = std(t(:,1));
t_avg_sd_withinRounds = mean(t(:,2));
excelReady.time{1} = [num2str(round(t_avg_acrossRounds,2)),'
',char(177),' ',...
    num2str(round(t_sd_acrossRounds,2))];
excelReady.time{2} = t_avg_sd_withinRounds;

% Column 2: rRMSE (rRMSE)
rRMSE_avg = zeros(10,4);
rRMSE_sd = zeros(10,4);
for i = 1:nRounds
    temp = finalResults{i,2};
    rRMSE_avg(i,:) = temp(17,:);
    rRMSE_sd(i,:) = temp(18,:);
end
rRMSE_avg_acrossRounds = mean(rRMSE_avg);
rRMSE_sd_acrossRounds = std(rRMSE_avg);
rRMSE_avg_sd_withinRounds = mean(rRMSE_sd);

for i = 1:4
    tempExcelReady{i,1} =
[num2str(round(rRMSE_avg_acrossRounds(1,i),2)),...
        ' ',char(177),'
',num2str(round(rRMSE_sd_acrossRounds(1,i),2))];
end
excelReady.rRMSE{1} = tempExcelReady;
excelReady.rRMSE{2} = rRMSE_avg_sd_withinRounds;

% Column 3: R^2 (Rsq)
Rsq_avg = zeros(10,4);
Rsq_sd = zeros(10,4);
for i = 1:nRounds
    temp = finalResults{i,3};
    Rsq_avg(i,:) = temp(17,:);
    Rsq_sd(i,:) = temp(18,:);
end
Rsq_avg_acrossRounds = mean(Rsq_avg);
Rsq_sd_acrossRounds = std(Rsq_avg);
```

```
Rsq_avg_sd_withinRounds = mean(Rsq_sd);

for i = 1:4
    tempExcelReady{i,1} =
[num2str(round(Rsq_avg_acrossRounds(1,i),2)),...
        ' ',char(177),' ',num2str(round(Rsq_sd_acrossRounds(1,i),2))];
end
excelReady.Rsq{1} = tempExcelReady;
excelReady.Rsq{2} = Rsq_avg_sd_withinRounds;

% Column 4: RMSE (RMSE)
RMSE_avg = zeros(10,4);
RMSE_sd = zeros(10,4);
for i = 1:nRounds
    temp = finalResults{i,4};
    RMSE_avg(i,:) = temp(17,:);
    RMSE_sd(i,:) = temp(18,:);
end
RMSE_avg_acrossRounds = mean(RMSE_avg);
RMSE_sd_acrossRounds = std(RMSE_avg);
RMSE_avg_sd_withinRounds = mean(RMSE_sd);

for i = 1:4
    tempExcelReady{i,1} =
[num2str(round(RMSE_avg_acrossRounds(1,i),2)),...
        ' ',char(177),' ',num2str(round(RMSE_sd_acrossRounds(1,i),2))];
end
excelReady.RMSE{1} = tempExcelReady;
excelReady.RMSE{2} = RMSE_avg_sd_withinRounds;

% Column 5: plottingResults
dataLen = zeros(16,1);
temp = finalResults{1,5};
for j = 1:16
    plottingResults(j).oval = temp{j,1}(5:8,:);
    temp_oval = temp{j,1}(5:8,:);
    dataLen(j) = length(temp_oval(1,:));
end

for i = 1:16
    a_f = zeros(10,dataLen(i));
    a_a = zeros(10,dataLen(i));
    m_f = zeros(10,dataLen(i));
    m_a = zeros(10,dataLen(i));

    for j = 1:10
        temp = finalResults{j,5};
        a_f(j,:) = temp{i,1}(1,:);
        a_a(j,:) = temp{i,1}(2,:);
        m_f(j,:) = temp{i,1}(3,:);
        m_a(j,:) = temp{i,1}(4,:);
    end

    a_f_avg = mean(a_f);
    a_a_avg = mean(a_a);
    m_f_avg = mean(m_f);
    m_a_avg = mean(m_a);
```

```
    newpval = [a_f_avg; a_a_avg; m_f_avg; m_a_avg];
    plottingResults(i).pval = newpval;
end

% Save 'excelReady' and 'plottingResults'
```

*5.3 Ensemble average W-A results*

```
% A3_EnsembleAVG.m
%
% Megan McCabe
% April 2020
%

%% Load gait % information as well as performance metrics and results
load('SA_plottingResults_5_8.mat');

if exist('allData','var') == 0
    filepath = '/Users/meganmccabe/Desktop/Thesis/Data Analysis/ML
Approach/Data Ready for ML/allData_4_19.mat';
    load(filepath)
end

%% Ensemble Average Results

% Use the subjectID (first column in header data - column 1 of allData)
to
% divide data by subject
activity = input('Which activity (stair - 1, walk - 2)?: ');
header = allData{activity,1};
subjectID = header.SubjectID;
newGaitPercent = transpose(linspace(0,100,1000));
numSplits = zeros(16,1);
for i = 1:16
    pval = transpose(plottingResults(i).pval);
    oval = transpose(plottingResults(i).oval);
    results = [pval,oval];
    gaitCycle = [header.GaitPercent,header.StanceTF,header.SwingTF];
    if i<13
        gaitCycle = gaitCycle(subjectID==i,:);
    else
        gaitCycle = gaitCycle(subjectID==i+1,:);
    end

    % Split data into cycles
    [results_split{i,1},gait_split{i,1}] =
splitCycles_ML(results,gaitCycle);

    % Spline data so it can be combined into matrices
    rSplit = results_split{i,1};
    gSplit = gait_split{i,1};
    [~,numSplits(i)] = size(rSplit);
    for j = 1:numSplits(i)
        gaitPerc_current = gSplit{1,j}(:,1);
        if gaitPerc_current(end) == 0
            gaitPerc_current(end) = 100;
        end
        results_current = rSplit{1,j};
        for k = 1:8
            newresult = spline(gaitPerc_current,...
                results_current(:,k),newGaitPercent);
```

```
            splitBySubject{i,k}(:,j) = newresult;
        end
    end
end

for i = 1:16
    if i == 1
        start = 1;
    else
        start = sum(numSplits(1:i-1))+1;
    end
    stop = start+numSplits(i)-1;

    for j = 1:8
        currentData = splitBySubject{i,j};
        finalResults{1,j}(:,start:stop) = currentData;
    end
end

tempAVG = zeros(1000,8);
tempSTD = zeros(1000,8);
for i = 1:8
    currentData = finalResults{1,i};
    avgData = mean(currentData,2);
    stdData = std(currentData,0,2);
    tempAVG(:,i) = avgData;
    tempSTD(:,i) = stdData;
end

x = tempAVG;
finalTable_avg = table(newGaitPercent,x(:,1),x(:,2),x(:,3),x(:,4),...
    x(:,5),x(:,6),x(:,7),x(:,8),...
    'VariableNames',{'GaitPercent','t_flex_wa','t_add_wa',...
    'm_flex_wa','m_add_wa','t_flex_is',...
    't_add_is','m_flex_is','m_add_is'});

x = tempSTD;
finalTable_std = table(newGaitPercent,x(:,1),x(:,2),x(:,3),x(:,4),...
    x(:,5),x(:,6),x(:,7),x(:,8),...
    'VariableNames',{'GaitPercent','t_flex_wa','t_add_wa',...
    'm_flex_wa','m_add_wa','t_flex_is',...
    't_add_is','m_flex_is','m_add_is'});
```

*5.4 Plot W-A and I-S ensemble averaged results*

```
% MLplotResults.m
%
% Megan McCabe
% April 2020
%

%% Load results and lit comp .mat struct
load('SA_ensembledResults_avg.mat');
load('SA_ensembledResults_sd.mat');
load('SA_litComp_Costigan_Protpapdaki.mat');


%% Plot all results (I-S, W-A, Lit Comp)

x = finalTable_avg{:,1};
waAVG = finalTable_avg{:,2:5};
isAVG = finalTable_avg{:,6:9};
waSTD = finalTable_std{:,2:5};
isSTD = finalTable_std{:,6:9};
figure(1);
for i = 1:4
    y_wa = waAVG(:,i);
    y_is = isAVG(:,i);
    y_lit = finalLitComp{:,i+1};
    if i == 2
        xlit = transpose(linspace(0,60,101));
    else
        xlit = finalLitComp{:,1};
    end

    if i == 3 % Flip m_flex data so that it is actually flexion
positive
        y_wa = y_wa*-1;
        y_is = y_is*-1;
    end
    d_wa = waSTD(:,i);
    m_wa = [y_wa - d_wa, y_wa + d_wa];
    d_is = isSTD(:,i);
    m_is = [y_is - d_is, y_is + d_is];

    subplot(2,2,i);
    plot(x,y_wa,'b',x,m_wa(:,1),':b',x,m_wa(:,2),':b',...
        x,y_is,'r',x,m_is(:,1),':r',x,m_is(:,2),':r',xlit,...
        y_lit,'-k','LineWidth',4);
    xlabel('gait cycle (%)');

    switch i
        case 1
            title('Flexion');
            ylabel('joint angle (\circ)');
        case 2
            title('Adduction');
            ylabel('joint angle (\circ)');
        case 3
```

```
            ylabel('moment (Nm/kg)');
        case 4
            ylabel('moment (Nm/kg)');
    end
    set(gca,'FontSize',14);
    set(gca,'color','w');
    xline(60,'-');

end

figure(2);
a = 1:10;
b = 11:20;
c = 21:30;
d = 31:40;
plot(a,b,'r',a,c,'b',a,d,'k','LineWidth',4);
legend('I-S','W-
A','Literature','FontSize',14,'Location','northeastoutside');
set(gca,'Color','w');
```

*5.5 Create violin plots*

```
% Function for producing violin plots was created by bastibe,
seamusholden, patrickfletcher, and sg-s and found on Github.com
(https://github.com/bastibe/Violinplot-Matlab)

% violinPlotter.m
%
% Megan McCabe
% May 2020
%

%% Load data to be plotted

load('/Users/megan_mccabe/Desktop/Thesis/Data Analysis/ML Approach/5_8
Final SA Iteration/SA_fullResults.mat');

rRMSE_struct = finalResults(:,2);
Rsq_struct = finalResults(:,3);

a_f = zeros(16,10);
a_a = zeros(16,10);
m_f = zeros(16,10);
m_a = zeros(16,10);

for i = 1:10
    current = rRMSE_struct{i,1};
    a_f(:,i) = current(1:16,1);
    a_a(:,i) = current(1:16,2);
    m_f(:,i) = current(1:16,3);
    m_a(:,i) = current(1:16,4);
end

a_f_avg = mean(a_f,2);
a_a_avg = mean(a_a,2);
m_f_avg = mean(m_f,2);
m_a_avg = mean(m_a,2);

rRMSE = [a_f_avg, a_a_avg, m_f_avg, m_a_avg];


for i = 1:10
    current = Rsq_struct{i,1};
    a_f(:,i) = current(1:16,1);
    a_a(:,i) = current(1:16,2);
    m_f(:,i) = current(1:16,3);
    m_a(:,i) = current(1:16,4);
end

a_f_avg = mean(a_f,2);
a_a_avg = mean(a_a,2);
m_f_avg = mean(m_f,2);
m_a_avg = mean(m_a,2);

Rsq = [a_f_avg, a_a_avg, m_f_avg, m_a_avg];
```

```
figure(1);
subplot(2,2,1);
violin1 = violinplot(rRMSE(:,1:2),{'flexion','adduction'});
title('Hip Joint Angle');
ylabel('rRMSE (%)');
set(gca,'FontSize',14);
set(gca,'color','w');

subplot(2,2,2);
violin2 = violinplot(rRMSE(:,3:4),{'flexion','adduction'});
title('Hip Joint Moment');
ylabel('rRMSE (%)');
set(gca,'FontSize',14);
set(gca,'color','w');

subplot(2,2,3);
violin3 = violinplot(Rsq(:,1:2),{'flexion','adduction'});
title('Hip Joint Angle');
ylabel('R^2');
set(gca,'FontSize',14);
set(gca,'color','w');

subplot(2,2,4);
violin4 = violinplot(Rsq(:,3:4),{'flexion','adduction'});
title('Hip Joint Moment');
ylabel('R^2');
set(gca,'FontSize',14);
set(gca,'color','w');
```

## 6. OpenSim Results – Data Analysis

*6.1 Import ID results (.sto files)*

### *6.1.1 Main script*

```
% A1) IDresults_sto2mat.m
%
% This script pulls OpenSim ID results from .sto file into matlab
% structure.
%
% Megan McCabe
% Dec 6, 2019


% Ask user to find folder with .sto files
h1 = msgbox('Please select the OpenSim ID results file.');
uiwait(h1);
[fileName,filePath] = uigetfile('*.sto');

% Grab data and store as Matlab table
filePath = strcat(filePath,'/',fileName);
fileID = fopen(filePath);
stoData = sto2mat_fxn(fileID);

% Plot hip Data
hipData = stoData{:,[1,8:13]};
time = hipData(:,1);

% Left hip
figure(1);
subplot(1,2,1);
p = plot(time,hipData(:,5),time,hipData(:,6),time,hipData(:,7));
p(1).LineWidth = 3;
p(1).Color = [0.8500 0.3250 0.0980];
p(2).LineWidth = 3;
p(2).Color = [0 0.4470 0.7410];
p(3).LineWidth = 3;
p(3).Color = [0.4660 0.6740 0.1880];
set(gca,'FontSize',16);
xlim([min(time),inf]);
title('Left Hip');
legend('Flexion','Adduction','Rotation');
xlabel('time (s)');
ylabel('moment (Nm)');

% Right hip
subplot(1,2,2);
p = plot(time,hipData(:,2),time,hipData(:,3),time,hipData(:,4));
p(1).LineWidth = 3;
p(1).Color = [0.8500 0.3250 0.0980];
p(2).LineWidth = 3;
p(2).Color = [0 0.4470 0.7410];
p(3).LineWidth = 3;
p(3).Color = [0.4660 0.6740 0.1880];
set(gca,'FontSize',16);
```

```
xlim([min(time),inf]);
title('Right Hip');
legend('Flexion','Adduction','Rotation');
xlabel('time (s)');
ylabel('moment (Nm)');
```

### *6.1.2 Function to convert .sto files to .mat*

```
function [dataTable] = sto2mat_fxn(trcFileID)
    % Import data
    formatSpec = '%s';
    stoData = textscan(trcFileID,formatSpec,'delimiter','<\t>');
    stoData = stoData{1,1};

    % Organize header data
    nHeaderRows = 6;
    nRows = stoData{3,1};
    nRows(1:6) = [];
    nCols = stoData{4,1};
    nCols(1:9) = [];
    ColNames = stoData(7:30,1);


    % Organize data
    [lenFile,junk] = size(stoData);
    for c = 1:lenFile
        if strcmp(stoData{c,1},'mtp_angle_l_moment') == 1
            dataBegins = c+1;
        end
    end

    Data = zeros(lenFile-dataBegins+1,1);
    b = 0;
    for a = dataBegins:lenFile
        b = b+1;
        Data(b) = str2double(stoData{a,1});
    end

    numDataCols = str2double(nCols);
    numDataRows = str2double(nRows);
    lenData = length(Data);

    Data_reformatted = zeros(numDataRows, numDataCols);
    Data_reformatted(1,:) = transpose(Data(1:numDataCols));

    count = 0;
    for a = 1:numDataCols:lenData
        count = count + 1;
        start = a;
        stop = a+numDataCols-1;
        Data_reformatted(count,:) = transpose(Data(start:stop));
    end

   % Save data as a table
   dataTable = array2table(Data_reformatted,'VariableNames',ColNames);


end
```

## *6.2 Import IK results (.mot files)*

### *6.2.1 Main script*

```
% IKresults_mot2mat.m
%
% This script pulls OpenSim IK results from .mot file into matlab
% table.
%
% Megan McCabe
% Feb 2020


% Ask user to pick the IKresults .mot file
parentfolder = '/Users/megan_mccabe/Desktop/Thesis/Data
Analysis/Results';
h1 = msgbox('Please select the OpenSim IK results file.');
uiwait(h1);
[fileName,filePath] = uigetfile('*.mot');

% Grab data and store as Matlab table
filePath = strcat(filePath,'/',fileName);
fileID = fopen(filePath);
IKresults = mot2mat_fxn(fileID);

% Plot hip Data
hipData = [IKresults{:,1},IKresults{:,8:10},IKresults{:,15:17}];
time = hipData(:,1);

% Left hip
figure(1);
subplot(1,2,1);
p = plot(time,hipData(:,2),time,hipData(:,3),time,hipData(:,4));
p(1).LineWidth = 3;
p(1).Color = [0.8500 0.3250 0.0980];
p(2).LineWidth = 3;
p(2).Color = [0 0.4470 0.7410];
p(3).LineWidth = 3;
p(3).Color = [0.4660 0.6740 0.1880];
set(gca,'FontSize',16);
xlim([min(time),inf]);
title('Left Hip');
legend('Flexion','Adduction','Rotation');
xlabel('time (s)');
ylabel('angle (deg)');

% Right hip
subplot(1,2,2);
p = plot(time,hipData(:,5),time,hipData(:,6),time,hipData(:,7));
p(1).LineWidth = 3;
p(1).Color = [0.8500 0.3250 0.0980];
p(2).LineWidth = 3;
p(2).Color = [0 0.4470 0.7410];
p(3).LineWidth = 3;
p(3).Color = [0.4660 0.6740 0.1880];
set(gca,'FontSize',16);
```

```
xlim([min(time),inf]);
title('Right Hip');
legend('Flexion','Adduction','Rotation');
xlabel('time (s)');
ylabel('angle (deg)');
```

### *6.2.2 Function to convert .mot files to .mat*

```
function [dataTable] = mot2mat_fxn(FileID)
    % Import data
    formatSpec = '%s';
    motData = textscan(FileID,formatSpec,'delimiter','<\t>');
    motData = motData{1,1};

    % Organize header data
    nHeaderRows = 10;
    nRows = motData{3,1};
    nRows(1:6) = [];
    nRows = str2num(nRows);
    nCols = motData{4,1};
    nCols(1:9) = [];
    nCols = str2num(nCols);
    ColNames = motData(9:32,1);

    % Organize data
    motData(1:32,:) = [];
    mot_reformatted = zeros(nRows,nCols);

    tempData_int = zeros(nCols,1);
    for i = 1:nRows
        start = (i-1)*nCols+1;
        stop = start+nCols-1;

        tempData_str = motData(start:stop,1);
        for j = 1:nCols
            tempData_int(j) = str2double(tempData_str{j,1});
        end
        mot_reformatted(i,:) = transpose(tempData_int);
    end

    % Save data as a table
    dataTable = array2table(mot_reformatted,'VariableNames',ColNames);

    fclose(FileID);
end
```

*6.3 Consolidate hip ID and IK results*

```
% A2) collectHipData.m
%
% Megan McCabe
% March 2020

% Ask user to pick the synced data file
h1 = msgbox('Please select the synced data file.');
uiwait(h1);
[filename_sync,filepath_sync] =
uigetfile('/Users/megan_mccabe/Desktop/Thesis/Data Analysis/Results');
load([filepath_sync,filename_sync]);

% Ask user to pick folder with ik data
h1 = msgbox('Please select the OpenSim IK Results Folder.');
uiwait(h1);
FolderPath = uigetdir('/Users/megan_mccabe/Desktop/Thesis/Data
Analysis/Results');
FolderContents = dir(FolderPath);
[numFiles,~] = size(FolderContents);

count = 0;
for i = 1:numFiles
    filename = FolderContents(i).name;
    if contains(filename,'.mat')
        count = count+1;
        Results(count).filename = filename;
        load([FolderPath,'/',filename]);
        Results(count).time = IKresults(:,1);
        Results(count).IKresultsR = IKresults(:,[8,9,10]);
        Results(count).IKresultsL = IKresults(:,[15,16,17]);
    end
end

% Ask user to pick folder with id data
w = input('Please enter subject weight (kg): ');

h1 = msgbox('Please select the OpenSim ID Results Folder.');
uiwait(h1);
FolderPath = uigetdir('/Users/megan_mccabe/Desktop/Thesis/Data
Analysis/Results');
FolderContents = dir(FolderPath);
[numFiles,~] = size(FolderContents);

count = 0;
for i = 1:numFiles
    filename = FolderContents(i).name;
    if contains(filename,'.mat')
        count = count+1;
        Results(count).filename = filename;
        load([FolderPath,'/',filename]);

        % normalize ID results to subject weight
        temp = stoData{:,8:10};
        temp = temp./w;
```

```
        stoData{:,8:10} = temp;

        temp = stoData{:,11:13};
        temp = temp./w;
        stoData{:,11:13} = temp;

        Results(count).IDresultsR = stoData(:,8:10);
        Results(count).IDresultsL = stoData(:,11:13);
    end
end

% Delete the step up file for now
% Results(2) = [];

% Add gait % vector to data
Results(1).gaitCycleR = SyncedData(1).percentGaitR;
Results(1).gaitCycleL = SyncedData(1).percentGaitL;
Results(2).gaitCycleR = SyncedData(4).percentGaitR;
Results(2).gaitCycleL = SyncedData(4).percentGaitL;

%% Save Results struct as "SubjectID_consolidatedResults_date"!
```

## 6.4 Split data into gait cycles

### 6.4.1 Main script

```
% A4) splitIntoGaitCycles
%
% Megan McCabe
% March 2020

%% Load consolidated results

rightFooted_TF = input('Is the subject right (1) or left (0) foot
dominant? ');

for i = 1:2
    t = Results(i).time;
    filename = Results(i).filename;
    if rightFooted_TF == 1
        ik = Results(i).IKresultsR;
        id = Results(i).IDresultsR;
        gaitCycle = Results(i).gaitCycleR;
    else
        ik = Results(i).IKresultsL;
        id = Results(i).IDresultsL;
        gaitCycle = Results(i).gaitCycleL;
    end

    Results_split = splitCycles(filename,ik,id,gaitCycle);
    [~,numSplits] = size(Results_split);

    newGaitPercent = transpose(linspace(0,100,1000));
    Results_matrix(i).filename = filename;
    Results_matrix(i).gaitCyclePercent = newGaitPercent;
    theta_flex = zeros(1000,numSplits);
    theta_add = zeros(1000,numSplits);
    theta_rot = zeros(1000,numSplits);
    m_flex = zeros(1000,numSplits);
    m_add = zeros(1000,numSplits);
    m_rot = zeros(1000,numSplits);
    for j = 1:numSplits
        ik = Results_split(j).ik;
        id = Results_split(j).id;
        gaitCycle = Results_split(j).gaitCycle;

        if gaitCycle(end,1) == 0
            gaitCycle(end,:) = [];
            ik(length(ik{:,1}),:) = [];
            id(length(id{:,1}),:) = [];
        end

        for k = 1:3
            iknew(:,k) = spline(gaitCycle(:,1),ik{:,k},newGaitPercent);
            idnew(:,k) = spline(gaitCycle(:,1),id{:,k},newGaitPercent);
        end

        theta_flex(:,j) = iknew(:,1);
```

```
        theta_add(:,j) = iknew(:,2);
        theta_rot(:,j) = iknew(:,3);
        m_flex(:,j) = idnew(:,1);
        m_add(:,j) = idnew(:,2);
        m_rot(:,j) = idnew(:,3);
    end

    Results_matrix(i).theta_flex = theta_flex;
    Results_matrix(i).theta_add = theta_add;
    Results_matrix(i).theta_rot = theta_rot;
    Results_matrix(i).m_flex = m_flex;
    Results_matrix(i).m_add = m_add;
    Results_matrix(i).m_rot = m_rot;

    theta_flex_avg = mean(theta_flex,2);
    theta_add_avg = mean(theta_add,2);
    theta_rot_avg = mean(theta_rot,2);
    m_flex_avg = mean(m_flex,2);
    m_add_avg = mean(m_add,2);
    m_rot_avg = mean(m_rot,2);
    Results_matrix(i).avgTable = table(theta_flex_avg,theta_add_avg,...
        theta_rot_avg, m_flex_avg,m_add_avg,m_rot_avg);
end

%% Save Results_matrix as SubjectID_splitGaitCycles_date
```

### *6.4.2 Function to split gait cycles*

```
function Results_split = splitCycles(filename,ik,id,gaitCycle)

Results_split.filename = filename;

% first determine indices that the data needs to be split by
% gaitCycle - where are 0s, check that stance TF is '1'

hasGaitPerc = gaitCycle(:,1)~=999;
ik = ik(hasGaitPerc,:);
id = id(hasGaitPerc,:);
gaitCycle = gaitCycle(hasGaitPerc,:);

hsindices = find(gaitCycle(:,1)==0);
stanceTF = gaitCycle(hsindices,2);
hsindices = hsindices(logical(stanceTF));

numSplits = length(hsindices);
for i = 1:numSplits-1
   start = hsindices(i);
   stop = hsindices(i+1)-1;
   Results_split(i).ik = ik(start:stop,:);
   Results_split(i).id = id(start:stop,:);
   Results_split(i).gaitCycle = gaitCycle(start:stop,:);
end

Results_split(numSplits).ik = ik(hsindices(end):end,:);
Results_split(numSplits).id = id(hsindices(end):end,:);
Results_split(numSplits).gaitCycle = gaitCycle(hsindices(end):end,:);

end
```

## *6.5 Plot ensemble averaged results for just the I-S approach*

### *6.5.1 Main script*

```
% A4) plotAcrossSubjects.m
%
% Megan McCabe
% March 2020

% Ask user to pick folder with consolidated results
h1 = msgbox('Please select the Consolidated Results Folder.');
uiwait(h1);
% laptop
FolderPath = uigetdir('/Users/megan_mccabe/Desktop/Thesis/Data
Analysis/Standard Approach/Results/Consolidated Results');
FolderContents = dir(FolderPath);
[numFiles,~] = size(FolderContents);

% Get rid of ghost files
for i = numFiles:-1:1
    bytes = FolderContents(i).bytes;
    if bytes < 10000
        FolderContents(i) = [];
    end
end

[numFiles,~] = size(FolderContents);
numSplits = zeros(2,1);
numSplits = zeros(numFiles,2);
% load data from each subject file and store in struct
for i = 1:numFiles
    currentData = load([FolderPath,'/',FolderContents(i).name]);
    currentData = currentData.Results_matrix;
    stair = currentData(1).theta_add;
    walk = currentData(2).theta_add;
    numSplits(i,1) = length(stair(1,:));
    numSplits(i,2) = length(walk(1,:));
    subjectData(i).data = currentData;
end

% need to create matrix for each angle and moment with columns
% corresponding with gait cycles from all subjects
theta_flex_st = zeros(1000,sum(numSplits(:,1)));
theta_add_st = zeros(1000,sum(numSplits(:,1)));
theta_rot_st = zeros(1000,sum(numSplits(:,1)));
m_flex_st = zeros(1000,sum(numSplits(:,1)));
m_add_st = zeros(1000,sum(numSplits(:,1)));
m_rot_st = zeros(1000,sum(numSplits(:,1)));

theta_flex_wk = zeros(1000,sum(numSplits(:,2)));
theta_add_wk = zeros(1000,sum(numSplits(:,2)));
theta_rot_wk = zeros(1000,sum(numSplits(:,2)));
m_flex_wk = zeros(1000,sum(numSplits(:,2)));
m_add_wk = zeros(1000,sum(numSplits(:,2)));
m_rot_wk = zeros(1000,sum(numSplits(:,2)));
```

```
for i = 1:numFiles
    if i == 1
        st_start = 1;
        st_stop = st_start+numSplits(1,1)-1;
        wk_start = 1;
        wk_stop = wk_start+numSplits(1,2)-1;
    else
        st_start = sum(numSplits(1:i-1,1))+1;
        st_stop = st_start+numSplits(i,1)-1;
        wk_start = sum(numSplits(1:i-1,2))+1;
        wk_stop = wk_start+numSplits(i,2)-1;
    end

    currentData = subjectData(i).data;

    theta_flex_st(:,st_start:st_stop) = currentData(1).theta_flex;
    theta_add_st(:,st_start:st_stop) = currentData(1).theta_add;
    theta_rot_st(:,st_start:st_stop) = currentData(1).theta_rot;
    m_flex_st(:,st_start:st_stop) = currentData(1).m_flex;
    m_add_st(:,st_start:st_stop) = currentData(1).m_add;
    m_rot_st(:,st_start:st_stop) = currentData(1).m_rot;

    theta_flex_wk(:,wk_start:wk_stop) = currentData(2).theta_flex;
    theta_add_wk(:,wk_start:wk_stop) = currentData(2).theta_add;
    theta_rot_wk(:,wk_start:wk_stop) = currentData(2).theta_rot;
    m_flex_wk(:,wk_start:wk_stop) = currentData(2).m_flex;
    m_add_wk(:,wk_start:wk_stop) = currentData(2).m_add;
    m_rot_wk(:,wk_start:wk_stop) = currentData(2).m_rot;

end

% Average across gait cycles for all subjects to obtain final results
for
% both activities (stair ascent and walking)
gaitCycle = transpose(linspace(0,100,1000));
theta_flex = mean(theta_flex_st,2);
theta_add = mean(theta_add_st,2);
theta_rot = mean(theta_rot_st,2);
m_flex = mean(m_flex_st,2);
m_add = mean(m_add_st,2);
m_rot = mean(m_rot_st,2);
stairResults_avg =
table(gaitCycle,theta_flex,theta_add,theta_rot,m_flex,m_add,m_rot);

theta_flex = std(theta_flex_st,0,2);
theta_add = std(theta_add_st,0,2);
theta_rot = std(theta_rot_st,0,2);
m_flex = std(m_flex_st,0,2);
m_add = std(m_add_st,0,2);
m_rot = std(m_rot_st,0,2);
stairResults_std =
table(gaitCycle,theta_flex,theta_add,theta_rot,m_flex,m_add,m_rot);

theta_flex = mean(theta_flex_wk,2);
theta_add = mean(theta_add_wk,2);
theta_rot = mean(theta_rot_wk,2);
m_flex = mean(m_flex_wk,2);
```

```
m_add = mean(m_add_wk,2);
m_rot = mean(m_rot_wk,2);
walkResults_avg =
table(gaitCycle,theta_flex,theta_add,theta_rot,m_flex,m_add,m_rot);

theta_flex = std(theta_flex_wk,0,2);
theta_add = std(theta_add_wk,0,2);
theta_rot = std(theta_rot_wk,0,2);
m_flex = std(m_flex_wk,0,2);
m_add = std(m_add_wk,0,2);
m_rot = std(m_rot_wk,0,2);
walkResults_std =
table(gaitCycle,theta_flex,theta_add,theta_rot,m_flex,m_add,m_rot);

%% Plot results

% Stairs
plotResults(1,stairResults_avg,stairResults_std);
% Walking
plotResults(2,walkResults_avg,walkResults_std);
```

### *6.5.2 Function to plot*

```
function plotResults(figNum,avgData,stdData)

% 1 = hip flexion angle
% 2 = hip adduction angle
% 3 = hip internal rotation angle
% 4 = hip flexion moment (needs to be flipped from what is outputed
from
% openSim)
% 5 = hip adduction moment
% 6 = hip internal rotation moment

x = avgData{:,1};
y1 = avgData{:,2};
y2 = avgData{:,3};
y3 = avgData{:,4};
y4 = -1*avgData{:,5}; % -1 to flip data across x-axis
y5 = avgData{:,6};
y6 = avgData{:,7};

d1 = stdData{:,2};
m1 = [y1-d1,y1+d1];
d2 = stdData{:,3};
m2 = [y2-d2,y2+d2];
d3 = stdData{:,4};
m3 = [y3-d3,y3+d3];
d4 = stdData{:,5};
m4 = [y4-d4,y4+d4];
d5 = stdData{:,6};
m5 = [y5-d5,y5+d5];
d6 = stdData{:,7};
m6 = [y6-d6,y6+d6];

figure(figNum);
subplot(2,3,1);
plot(x,y1,'b',x,m1(:,1),':b',x,m1(:,2),':b','LineWidth',4);
title('Hip Flexion');
xlabel('gait cycle (%)');
ylabel('hip joint angle (deg)');
set(gca,'FontSize',14);
set(gca,'color','w');
xline(60,'-');

subplot(2,3,2);
plot(x,y2,'b',x,m2(:,1),':b',x,m2(:,2),':b','LineWidth',4);
title('Hip Adduction');
xlabel('gait cycle (%)');
ylabel('hip joint angle (deg)');
set(gca,'FontSize',14);
set(gca,'color','w');
xline(60,'-');

subplot(2,3,3);
plot(x,y3,'b',x,m3(:,1),':b',x,m3(:,2),':b','LineWidth',4);
```

```
title('Hip Internal Rotation');
xlabel('gait cycle (%)');
ylabel('hip joint angle (deg)');
set(gca,'FontSize',14);
set(gca,'color','w');
xline(60,'-');

subplot(2,3,4);
plot(x,y4,'b',x,m4(:,1),':b',x,m4(:,2),':b','LineWidth',4);
xlabel('gait cycle (%)');
ylabel('hip moment (N-m/kg)');
set(gca,'FontSize',14);
set(gca,'color','w');
xline(60,'-');

subplot(2,3,5);
plot(x,y5,'b',x,m5(:,1),':b',x,m5(:,2),':b','LineWidth',4);
xlabel('gait cycle (%)');
ylabel('hip moment (N-m/kg)');
set(gca,'FontSize',14);
set(gca,'color','w');
xline(60,'-');

subplot(2,3,6);
plot(x,y6,'b',x,m6(:,1),':b',x,m6(:,2),':b','LineWidth',4);
xlabel('gait cycle (%)');
ylabel('hip moment (N-m/kg)');
set(gca,'FontSize',14);
set(gca,'color','w');
xline(60,'-');
end
```

*6.6 Calculate gait parameters*

```
% GaitMetricCalculator.m
%
% Megan McCabe
% April 2020
%

%% Load gait % information as well as performance metrics and results

if exist('allData','var') == 0
    filepath = '/Users/megan_mccabe/Desktop/Thesis/Data Analysis/ML
Approach/Data Ready for ML/allData_5_21.mat';
    load(filepath)
end

% stride time (s)
% stride cadence (cycles/min)
% percent stance
% percent swing

for i = 1:2 % cycle through two activities
    currentData = allData(i,:);
    header = currentData{1,1};
    subjectID = header{:,1};
    strideMetrics_SubjectAVG = zeros(17,4);
    for j = 1:17 % cycle through subjects
        currentHeader = header{subjectID==j,:};
        gaitPerc = currentHeader(:,2);
        currentHeader(gaitPerc==999,:) = [];
        gaitPerc = currentHeader(:,2);
        stanceTF = currentHeader(:,3);
        swingTF = currentHeader(:,4);

        hsindices = find(gaitPerc==0);
        stanceTF_hs = stanceTF(hsindices);
        hsindices = hsindices(logical(stanceTF_hs));

        numSplits = length(hsindices);
        strideMetrics = zeros(numSplits,4);
        for k = 1:numSplits
            start = hsindices(k);
            if k == numSplits
                stop = length(gaitPerc);
            else
                stop = hsindices(k+1)-1;
            end
            strideMetrics(k,1) = (stop - start)*1/128; % stride
duration
            strideMetrics(k,2) = 1/strideMetrics(k,1); % stance cadence
(cycles/sec)
            stanceLen = length(nonzeros(stanceTF(start:stop)));
            swingLen = length(nonzeros(swingTF(start:stop)));
            durationLen = length(gaitPerc(start:stop));
            strideMetrics(k,3) = stanceLen/durationLen*100; % Percent
stance
```

```
            strideMetrics(k,4) = swingLen/durationLen*100; % Percent
swing

        end

        strideMetrics_SubjectAVG(j,:) = mean(strideMetrics,1);
    end

    acrossSubject_avgstd = [mean(strideMetrics_SubjectAVG,1);...
        std(strideMetrics_SubjectAVG,0,1)];
    finalResults{i,1} = acrossSubject_avgstd;
    finalResults{i,2} = strideMetrics_SubjectAVG;
end

stairResults = finalResults{1,1};
s_a = round(stairResults(1,:),2);
s_s = round(stairResults(2,:),2);
walkResults = finalResults{2,1};
w_a = round(walkResults(1,:),2);
w_s = round(walkResults(2,:),2);
for m = 1:4
    excelText{m,1} = [num2str(s_a(m)),' ',char(177),'
',num2str(s_s(m))];
    excelText{m,2} = [num2str(w_a(m)),' ',char(177),'
',num2str(w_s(m))];
end
```