Passing the Load

An In-Situ and In-Silico Approach for Biomechanical Analysis of Industrial Scaffold Workers

Master Thesis Biomedical Engineering

D.G.D. Koole

University of Technology

TU Delft

Passing the Load

An In-Situ and In-Silico Approach for Biomechanical Analysis of Industrial Scaffold Workers

by

D.G.D. Koole

to obtain the degree of Master of Science at the Delft University of Technology
to be defended on Monday 10th of November, 2025 at 11:00

TU Delft Supervisor:	Dr. Ir. A.H.A. Stienen
TU Delft committee member:	dr. J.K. Moore
Project Duration:	December, 2024 - November, 2025
Faculty:	Faculty of Mechanical Engineering, Delft
Student number:	4838882

Cover: Picture of Bilfinger SE scaffold workers on a scaffold

TU Delft

CONTENTS

Passing the Load: An *In-Situ* and *In-Silico* Approach for Biomechanical Analysis of Industrial Scaffold Workers

D.G.D. Koole

***Abstract*—Work-related Musculoskeletal Disorders represent the largest occupational health burden in the European union, imposing significant economic and societal burdens. In this study, an ergonomic risk assessment at Bilfinger SE identified the scaffolders' passing-on task, involving repetitive vertical transfer of heavy materials, as the highest-risk activity among their services.**

Additionaly, *in-situ* markerless kinematic (OpenCap) and kinetic (acceleration and forces) captures, as well as *in-silico* biomechanical analysis (OpenSim) were combined to uncover the highest-impact areas of the passing-on task on three experienced scaffold workers.

Inverse Dynamics analysis revealed that when external forces (stander weight) were included, mean lumbar lateral bending and rotation moments increased by over 10%. Shoulder flexion moments exceeded 60 degrees regularly, with the dominant arm experiencing over 15% higher mean moments over the non-dominant arm. Smaller materials (ledger, console) resulted in average mean joint moments that are 25% higher compared to the stander: 39.8 and 40.3 Nm compared to 31.3 Nm. The mean shoulder flexion moment in the dominant arm was 55% (ledger) and 20% (console) higher compared to the non-dominant arm.

These findings highlight excessive lumbar lateral bending and rotation, and asymmetric arm usage as key risk factors for WMSDs.

This study demonstrates the feasibility of a combined *in-situ* and *in-silico* approach of uncovering high-impact areas within occupational workers, informing targeted interventions.

Future works should address OpenCap's limitations, explore the biomechanical effects of compensatory movements, and the develop interventions to mitigate the identified risks.

I. Introduction

Work-related Musculoskeletal Disorders (WMSDs) account for the biggest group of occupational health issues in the European Union [1], with 41% of all workers reporting muscular pains in the shoulders, neck, and/or upper limbs, and 43% reporting backache. These disorders are responsible for significant social and economic impact, including absenteeism, decreased productivity, long-term disability, and mental health problems such as anxiety and sleeping problems. The European Agency for Safety and Health at Work (EU-OSHA) estimates that WMSDs cost billions of euros annually due to healthcare expenses and lost workdays [1]. In physically demanding sectors such as construction and industrial services, the risk of WMSDs is particularly high due to repetitive lifting, awkward postures, and heavy material handling.

For industrial services companies, like Bilfinger SE, this is a highly relevant problem. Their tasks involve heavy material lifting and heavy tools usages. Personnel fall out within Bilfinger SE as a result of WMSDs is higher than the average in the industry. Although they provide ergonomic interventions and training programs to mitigate and/or prevent these risks, they are not always sufficient [2] [3].

The use of industrial aids, such as exoskeletons, offer a promising solution to alleviate and/or prevent these problems by reducing or negating musculoskeletal loads on joints and muscles during occupational tasks [4]. However, these conflict with mandatory safety equipment. Other mainstream solutions such as the back belt have been shown to offer little to no preventive benefits [5]. Environmental constraints often prevent the usage of external equipments such as lifts.

Standard ergonomic risk assessments (ERAs) can identify which tasks are potentially harmful [6]. These assessments rely on human observations, but cannot grasp the detailed mechanical interactions that happen in the body. Experimental methods—such as *in-vivo* or *in-vitro* studies—provide deeper insights but are often constrained to laboratory settings, partial body regions, or invasive instrumentation such as electromyography [7] [8]. Consequently, these results lack the validity of the mechanisms in real life situations.

In contrast, *in-silico* approaches, which leverage dynamic computational models or simulation software, offer valuable alternatives. These techniques enable detailed analysis of the entire musculoskeletal system without the need for physical instruments on the body. Software platforms like OpenSim enable the estimation of joint moments, muscle forces, and body loading based on captured movement data [9]. Combining these with markerless motion capture tools such as OpenCap, which use computer vision algorithms to extract the needed movement data from simple video recordings, it becomes feasible to perform *in-situ* experiments. That is, analyzing movements on the original worksite during normal work tasks.

Uncovering high-impact areas via the *in-silico* approach, subsequent future development can be done that minimize WMSDs. These can target the high-impact specifically, while complying with the environmental constraints present. The central research question is:

Can a combination of ergonomic risk assessment and an in-situ in-silico approach identify high-impact musculoskeletal areas in a heavy industrial task, thereby guiding targeted intervention strategies?

TABLE I: Risk Assessment Matrix combining the relative frequency and intensity into three categories: Green is low risk, orange is medium risk, and red is high risk of musculoskeletal injury over a prolonged time frame.

Freq / Int	1	2	3	4	5
80-100%					
60-80%					
40-60%					
20-40%					
0-20%					

II. Task Exploration

As a way of exploring harmful tasks within the industrial services, a task exploration was conducted at a worksite of Bilfinger SE located at TATA Steel IJmuiden, the Netherlands. These tasks were then subjected to an ergonomic risk assessment (ERA), to uncover which one of them has the highest impact on the body over a period of time.

A. Background

To understand the working conditions of the Bilfinger employees, and consequently its physical impact on them, a background on the worksite and the involvement of Bilfinger SE at the TATA Steel site is performed.

1) TATA Steel Site: TATA Steel is located at the coastline, near the city of IJmuiden, the Netherlands. This provides excellent access to the sea, and thus is perfect for importing and exporting its products via ships. However, this close proximity to the sea also accelerates corrosion of steel structures and machinery due to the salty seawater [10].

Additionally, the inherent nature of steel production is one that leaves residue around the entire plant. During the creation of coke fuel, byproducts such as coal tar and coal tar pitch may partially escape the coke- and gasplant and descend on other parts of the factory [11].

Therefore, many of the structures require frequent maintenance in removing and preventing the rust and residue.

2) Bilfinger: Bilfinger provides industrial services to worksites around the globe. They focus on three main tasks: insulation, scaffolding, and painting. Insulation services provides insulation setups for the tubes and factories around the industrial sites, scaffold services set up scaffolds for other services to use, and painting services removes rust and residue and coats new protective layers. This is done with around 4800 employees placed over 23 worksites globally.

All employees present at TATA Steel wear personal protective equipment, as guided by safety guidelines. The minimum gear is a coverall, helmet, safety glasses, safety shoes, safety gloves, earplugs, and a carbon monoxide meter. Depending on the tasks at hand, there may be additional safety equipment necessary, such as fall harnesses, gas masks, or additional toxicity meters. These factors combined with the constrained factory environment provide conditions in which musculoskeletal strain can more easily develop.

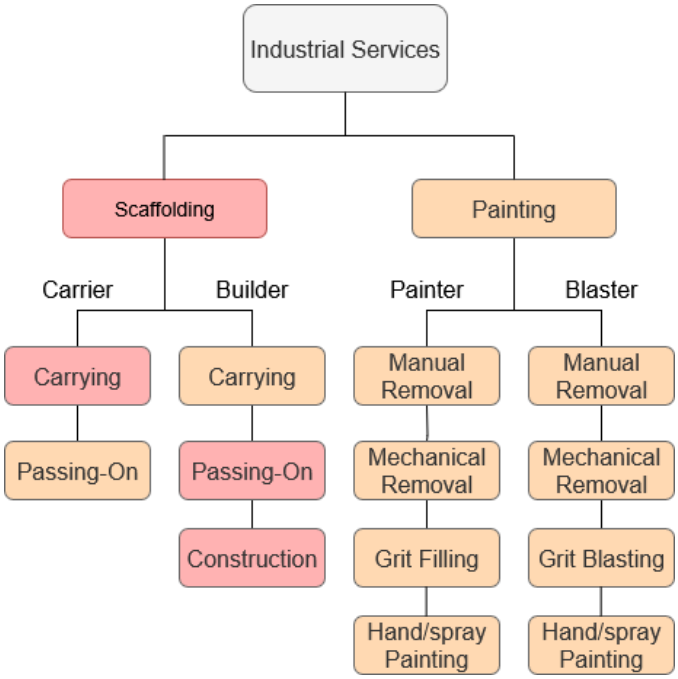


Fig. 1: Work breakdown structure of the scaffolding and painting services, with the resulting risk for each sub task as retrieved from the REBA method. Both the scaffolding and painting have different task sets for different employees. Red is a high risk and orange a medium risk of WMSDs over a prolonged period of time.

B. Ergonomic Risk Assessment

To identify tasks with the greatest long-term musculoskeletal risk, an ergonomic risk assessment (ERA) was conducted using a Work Breakdown Structure (WBS) combined with the Rapid Entire Body Assessment (REBA) scoring method [12].

The insulation group has been purposely left out of this analysis due to restricted time, as they were deemed a non-intensive task by the employees.

Each sub-task within Bilfinger's services was rated for frequency (percentage of weekly working time) and intensity (REBA score). The results were mapped in a risk matrix classifying tasks as low (green), medium (orange), or high (red) risk of WMSD over prolonged exposure as shown in Table I. Detailed scoring procedures are provided in Appendix A.

The results for the risk score combined with the WBS are shown in Figure 1.

The scaffolding group have an overall higher risk of WMSDs according to this analysis. The biggest ergonomic no-no's are present within this group: Handling heavy materials, lifting above the head, bending down, forceful exertions, repetitive motions, and twisting motions [13]. During a questionnaire performed with the employees, the passing-on task (POT) is especially noted to be the most intense task. This task involved passing on scaffolding material vertically up/down, and can be done for several hours at a time. This task is therefore chosen to be further analyzed with the *in-silico* approach in chapter III.

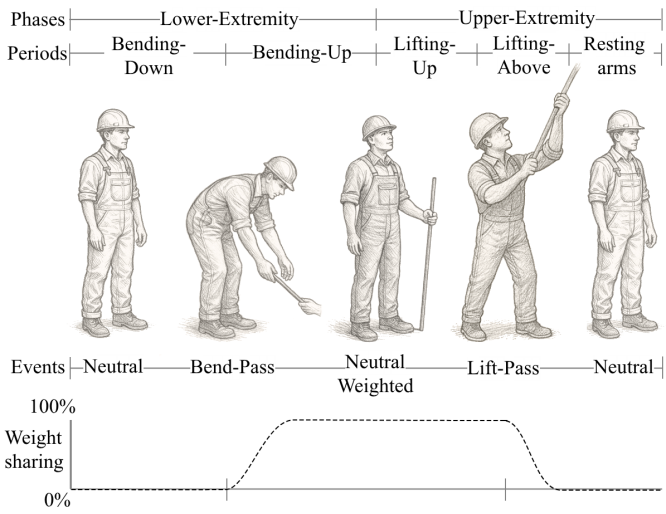


Fig. 2: Schematics of the passing-on task cycle, divided in two phases, five periods, and four events with corresponding movement involved. The weight sharing shows how much of the material weight is carried by the worker, shown in a percentage from 0% (not holding material) to 100% (fully carrying material).

C. Passing-On Task Cycle

The POT involves vertically transferring scaffolding components between two or more workers positioned on different scaffold levels, typically about 2 meters apart. To characterize its movement pattern, a POT cycle was created in Figure 2, inspired by the human gait cycle [14]. The POT cycle is characterized by the 'weight sharing': the percentage of the weight the worker themselves carries. With two workers sharing equal weight, the weight sharing is 50%. With the worker carrying all the weight, the weight sharing is 100%. The POT cycle consists of two main phases:

1) Lower-Extremity Phase
   - Bend-Down Period (BDP): worker flexes the trunk and/or knees to reach for incoming material.
   - Bend-Pass (BP) Event: weight transfer of material starts
   - Bend-Up Period (BUP): worker extends trunk to neutral-weighted event, weight sharing gradually increases to 100%
2) Upper-Extremity Phase
   - Lift-Up Period (LUP): material is lifted to shoulder height
   - Lift-Pass (LP) Event: weight transfer of material starts
   - Lift-Above Period (LAP): worker reaches above shoulder height, weight sharing gradually decreases to 0%
   - Resting Arms Period (RAP): worker returns to neutral event

1) Lower Extremity Phase: Occupational safety organizations like OSHA (USA) and ARBO (NL) advise lifting materials from a low height by bending the knees and keeping the back straight, also known as squat lift [15] [16]. This is to prevent the motion of excessive flexion of the back with keeping the legs straight, called a stoop lift, which would increase the moment on the lumbar spine by about 75%, resulting in a greater risk of injury [17]. The movement in between these is called a semi-squat. The differences in these techniques can be seen in Figure 3.

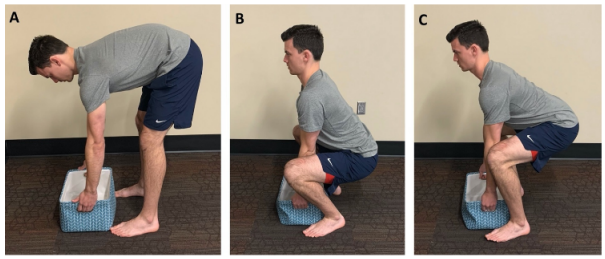


Fig. 3: Techniques of picking up material below knee-height. (A) Stoop lift characterized by straight legs and flexed back, (B) Squat lift characterized by bend legs and straight vertical back, (C) Semi-squat lift characterized by bend legs and straight non-vertical back. Image retrieved from [18].

Although the squat lift is generally recommended, and also taught to the scaffold workers, during the ERA many of them are still seen doing stoop lifts or at a maximum semi-squats in the BDP and BUP.

This is due to a number of factors, with the most likely one being that stoop lifting is more metabolically efficient and energy-sparing than (semi-)squat lifting [18] [19]. When it feels like squat lifting is more tiring than stoop lifting, and especially if the task is done repetitively for multiple hours at a time, workers tend toward the one method that has them less tired at the end of the day. Additionally, according to Washmuth et al. [18], stoop lifting is more familiar and may even be more efficient in areas with environmental constraints, which is the case in the POT .

Kingma et al. [20] also found that, by comparing even more lifting techniques and configurations of lifting up, the stoop lift results in a lower moment compared to a squat lift in some cases. As such, different situations generally call for different methods. Scaffold workers wear a fall harness, which inherently limit a worker's range of motion, and work in constrained environments which would sometimes prevent effective squat lifting to be possible.

2) Upper Extremity Phase: Occupational safety organizations recommended against lifting arms above shoulder height at all, due to the increased risk of shoulder injury [21] [22] [23]. Again, due to the aforementioned factors at the lower extremity phase, the workers are limited in alternative motions.

TABLE II: Subject characteristics for the experiment. Experience is the amount of years the subjects have had experience with the passing-on task.

Subject	Age (years))	Height (m)	Weight (kg)	Dominant Hand	Experience (years)
1	51	1.78	94	Right	14
2	46	1.74	96	Right	5
3	48	1.71	80	Right	22

III. Experimental Method

In the following, an *in-situ in-silico* experimental design to capture and analyze full-body kinematics and kinetics of scaffold workers is implemented. Markerless kinematic data was recorded using OpenCap, while external forces and accelerations were captured with a custom kinetic module using sensors. The data were then analyzed in OpenSim to retrieve relevant joint angles and moments.

A. Participants

Three middle-aged male professional scaffolders employed by Bilfinger SE volunteered for the study. Subject requirements include being able to talk Dutch or English, having no current injury, and at least 2 years of experience with the POT, which the scaffolders all met. Subject characteristics can be found in Table II.

All participants provided informed consent before participating in the experiment. The study was conducted at Bilfinger's training facility outside Shell Pernis, the Netherlands.

B. Materials and Tools

1) Physical Experimental Setup: The experimental setup is shown in Figure 4. The setup utilized a two-story scaffold to perform the POT on, with the distance between levels being about 2 meters.

Two iPhones, a model 10 and 11, were used to capture the kinematics, and were each placed at an 45 degree offset, at a height of 2.5 meters, with a distance of 3 meters to the subject. The point of view of these angles is shown in Figure 5. For calibration purposes, a raster grid is placed vertically on the scaffold, consisting of 4x5 35mm squares.

Four different scaffolding materials ranging in size and weight were used for the POT experiment, shown in Figure 6. These are the stander, used as the vertical corners in the scaffold; the ledger, a horizontal connector; the console, used mainly to build out of existing structures (as can be seen in Figure 4); and steel deck, used for the platform.

To capture the accelerations needed to calculate the forces, a custom kinetic module was created. This module is driven by an Arduino Nano, and consists of a BMI160 Intertial Measurement Unit (IMU) for all the materials. This module was then placed on the materials during the trials.

An additional force sensor was used with the stander, which was the KELI DEE Loadcell. This was placed in series with the stander, and was used to measure the weight-sharing between the subject and the below/above worker. A signal processor was added to boost the output signal of the loadcell.

Fig. 4: Experimental Setup for the POT, using a two-story scaffold setup. (A) and (B) iPhone cameras, (C) Rastergrid used for the calibration of the cameras.

Fig. 5: Point of view from iPhone cameras. (A) Subject standing in calibration pose, (B) Subject passing on steel deck.

Each subject wore their usual protective gear. In this case, this included the full-blue coverall, protective shoes, helmet, and fall harness.

2) Software: OpenCap was used as the capture software for the cameras. OpenCap is an open-source markerless motion capture software designed to make 3D movement analysis accessible, by using computer vision algorithms to capture and

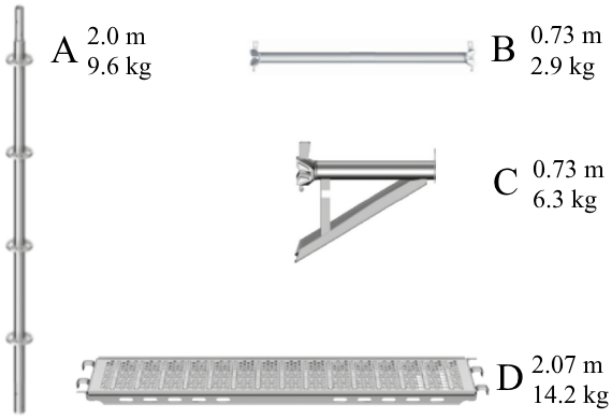


Fig. 6: Scaffolding materials used for the POT experiment with their main physical properties. (A) Stander, (B) Ledger, (C) Console (D) Steel deck.

analyze human movement without the need for lab equipment [24]. From the markerless motion capture systems available, OpenCap is the most accurate [25].

Arduino IDE was used to capture the kinetic data from the kinetic module at 125 Hz. The code for this is shown in Appendix H.

OpenSim [9] was used to analyze the dynamics of the scaffold worker. OpenSim is an open-source software platform developed primarily for biomechanics and movement science research. In order to uncover areas of high impact during the POT cycle, the movement (kinematics) along with the applied forces (kinetics) need to be analyzed. This can be done using the inverse dynamics (ID) tool within OpenSim to uncover joint dynamics, and then apply static optimization (SO) to calculate the resulting muscle forces. Details on the (mathematical) mechanics of these tools within OpenSim are shown in Appendix F. In this thesis, the SO performed gave non-viable results, likely due to a lack of supporting sensors, such as a ground reaction plate. These are therefore excluded in the results section.

As the OpenCap software is mainly used for gait analysis, the provided model, the Lai-Ulrich 2022 model [24], only has lower-extremity muscles included. However, the area of high impact of scaffold workers also include a lot of the upper body area, for which a model that includes full-body muscles is desirable. A model from the University of Maryland's Neuromechanics Research Core is chosen for this purpose, named UMocoD (version 3) [26]. This model is used to perform simulations of walking with an intact limb and with a unilateral transtibial prothesis. Similarly to the Lai-Ulrich 2022 model, it is based on the Rajagopal model, and includes 122 muscles (matching the same lower-extremity muscles as in Lai-Ulrich 2022), 22 joints, as well as actuators for most joints. The markers from the Lai-Ulrich 2022 model are exported, and copied onto the UMocoD model. This led to an almost correct 1-to-1 placement on the UMocoD model, with the exception of the arm and shoulder markers. This discrepancy is manually fixed in OpenSim.

Spyder Python (version 3.12) was used for the data acquisition and data analysis, to run the OpenSim API in (version 4.5.2).

C. Procedure

Before recording, each subject performed a brief calibration using the OpenCap neutral pose, as can be seen in Figure 5. For each material, the subject passed the component upward and downward between levels in repeated trials, mimicking actual working behavior. The standard sequence included:

1) Calibration (neutral pose for camera tracking);
2) Three repetitions for the stander and steel deck;
3) Four repetitions for the console and ledger: two passing via inside, two via outside.

Each trial consisted of one passing-up and one passing-down cycle, separated by a 3-second pause to stabilize postures.

D. Data Analysis

1) Model Validation: Because the Lai-Ulrich 2022 model was originally intended to be used for OpenCap, the new UMocoD model needed to be validated.

To ensure compatibility, comparative simulations were performed using OpenCap's sample "cut" and "squat" movements retrieved from their app.opencap.ai website. The code used for this correlation is shown in Appendix I.

The correlation between the two models for the 'cut' movement was 0.728 ± 0.283 for the IK, and 0.506 ± 0.300 for the ID.

The correlation for the 'squat' movement was 0.858 ± 0.163 for the IK, and 0.730 ± 0.297 for the ID.

These results are sufficient for further analysis, given expected inter-modal differences in joint conventions.

2) Pipeline: To automate the processing of the data, a pipeline is created. The code of this pipeline is shown in Appendix J. The pipeline takes the following steps:

0) Calibration
1) Record Kinematics and Kinetic data
2) Time-match and Trim Kinematics and Kinetic data
3) Create Hand Grab File
4) Create Scaled Model and Perform Inverse Kinematics
5) Determine Cutoff Frequency
6) Filter Data
7) Create External Force File
8) Perform Inverse Dynamics with and without External Force File
9) Plot results

0) Both the force sensor and IMU were calibrated prior to data collection to convert raw voltage and acceleration readings into physical units. The load cell was calibrated under known compressive and tensile conditions to determine calibration constants F_C and F_T, while the IMU was zeroed along each cardinal axis for 5 s per orientation.

The force conditions are needed for the weight-sharing calculation for the stander.

2) Kinematic and kinetic recordings were time-aligned using a visible tapping cue captured in both data. The synchronized data were trimmed to include only the passing-up and passing-down phases of each trial.

3) A hand grab file (HGF) was created for each trial. This HGF showed the timings of when the subject's left or right hand holds the material, and whether the above/below worker is also holding the material. This is used to accurately apply the forces on the OpenSim model. These functions are either 0 or 1, and as such are not accurate for human motion control. Therefore, a filtering function is used to smooth out the step.

A Gaussian filtering function is used. This is preferable over other types of filtering, like Butterworth or Savitzky-Golay filtering, as a Gaussian filter prevents overshooting at edge cases. The continuous Gaussian function is defined as:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \tag{1}$$

In which x is the function to filter, and σ is the standard deviation of the Gaussian. A larger σ results in a smoother

output. A -3dB factor is added, along with a factor of the time it takes for a full-grab to happen. This results in a smooth S-like curve starting when the hand touches the material, and ends when the hand fully grabs it.

During observations, it was determined that this time is about 50 ms for the POT.

4) Using the subject characteristics, the model was scaled accordingly. Subsequently, inverse kinematics was performed using the trials tracer data.

5) The captured data needed to be low-pass filtered due to high-frequency noise. For this, a suitable cutoff frequency (COF) needed to be determined, which was done with a Fourier analysis.

For the kinematic data, Uhlrich et al. [24] recommends the cutoff frequency to be one that retains 99.7% of the cumulative signal power of the Fourier-transformed kinematic motion. For the kinetic data, a similar process was done, separately for the force and accelerations.

An important consideration for the COF was using one consistently for both the kinematic and kinetic data. Applying separate cutoff frequencies to these are shown to alter the results negatively [27] [28] [29]. The literature recommends choosing a COF that is somewhere in between all of the found frequencies, but one that is closer to the lower-frequency kinematic data. Additionally, for effective comparisons between subjects and trials, this cutoff frequency also needs to be consistent over all the subjects and trials that are analyzed.

The resulting cutoff frequency was set at 2.8 Hz for all trials.

6) A Gaussian smoothing function was again used, for the aforementioned reasons, to filter the kinematic and kinetic data with the suitable COF. The reason for this repeated filter type usage was that using a single consistent filter reduced unwanted distortions in the data.

7) Using the HGF and kinetic data, the external force file (EFF) was created. Firstly, the weight-sharing function was applied to the force vector, which could only be done for the stander. This was applied in three parts depending on where in the POT cycle it occurred. It used the data from the HGF to determine whether the above/below worker was also holding the part.

During the BUP, the weight the subject was carrying was determined by how much tension occurred in the force sensor, as the subjects holds the top part of the stander above the sensor. At $F = F_T$, the subject was carrying 100% of the weight of the stander.

During the LUP, the subject was the only one holding the stander, and the full weight for the stander was therefore applied to the subject.

During the LAP, the opposite to the BUP happened. The bottom part of the stander was held by the subject. At maximum compression, $F = F_C$, the subject was carrying 100% of the weight of the stander.

In the case of the non-stander materials, this F function was a smooth gaussian function, acting on the period during which the subject and above/below worker held the material at the same time.

TABLE III: List of valid trials, and their characteristics. When the material is passed via the outside is noted with 1, via the inside noted with 2.

Subject	Trial	Material	Outside/Inside
1	1	1: Stander	1: Outside
1	2	1: Stander	1: Outside
1	3	1: Stander	1: Outside
1	4	2: Layer	1: Outside
1	5	2: Layer	1: Outside
1	7	3: Console	2: Inside
1	8	3: Console	2: Inside
2	4	3: Console	1: Outside
2	6	1: Stander	1: Outside
2	7	1: Stander	1: Outside
2	9	2: Layer	2: Inside
3	1	2: Layer	1: Outside
3	2	2: Layer	1: Outside
3	5	1: Stander	1: Outside
3	6	1: Stander	1: Outside
3	7	1: Stander	1: Outside
3	8	3: Console	1: Outside
3	9	3: Console	1: Outside

With the weight sharing applied, the force vector for each hand was created by applying the acceleration data and the HGF.

8) With the EFF and filtered kinematic data, the ID tool was used. This was both done with and without the EFF being applied, for effective comparison of the effect of the weight of the material. With the exact same movement and this discrepancy, the effects on the joints and muscles that are mainly active in the movement in the first place can be better analyzed. The resulting data was yID with the EFF, and nID without the EFF.

9) From the ID results, joints that were relevant to the difference between with/without EFF were chosen to be plotted.

IV. Results

The steel deck material was deemed inviable after a few try-outs. The plank obstructed too much of the camera views, thereby causing de-synchronization and invalid kinematics. This obstruction can be seen in Figure 5.

A total of 31 trials over the three subjects were captured.

14 trials were deemed invalid in the post-processing. The data provided for download had too much of the beginning cut off, with time synchronization not being able to be done, or the trials stopped during a passing.

An overview of the valid trials and their characteristics are shown in Table III.

11 of these invalid trials included those in which the material was passed on through the inside of the scaffold. The cause of this was likely two-fold: The monochrome overall of the subjects causes problems when body segments overlap, and the scaffold railings occluded the subject.

A. Inverse Dynamics

The possible joints for evaluation of the ID are shown in Table IV, along with them either being included/excluded, and for what reason.

TABLE IV: List of all ID joint motion forces, along with whether they are included in the results. *For both left and right extremity.

ID Joint motion	Included	Reason
Pelvis tilt moment	No	Captured with lumbar extension
Pelvis list moment	No	Captured with lumbar bending
Pelvis rotation moment	No	Captured with lumbar rotation
Pelvis tx force	No	Not relevant
Pelvis ty force	No	Not relevant
Pelvis tz force	No	Not relevant
Hip flexion moment*	No	No change with external forces
Hip adduction moment*	No	No change with external forces
Hip rotation moment*	No	No change with external forces
Lumbar extension moment	Yes	Relevant
Lumbar bending moment	Yes	Relevant and changes with external forces
Lumbar rotation moment	Yes	Relevant and changes with external forces
Knee angle beta force*	No	No change with external forces
Knee angle beta moment*	No	No change with external forces
Shoulder flexion moment*	Yes	Relevant and changes with external forces
Shoulder adduction moment*	Yes	Relevant and changes with external forces
Shoulder rotation moment*	No	Invalid results
Ankle angle moment*	No	No change with external forces
Elbow flexion moment*	No	No change with external forces
Subtalar angle moment moment*	No	No change with external forces
Pro-supination moment*	No	No change with external forces

From the 35 possible joint movements, 7 were relevant and show useful results. These were the lumbar flexion-extension, lateral bending and rotation, and shoulder flexion-extension and adduction-abduction. These were valid for both left and right arm.

An example of the resulting difference between the inclusion of EFF can be seen in Figure 7. The results for each subject and trial can be found in Appendix G.

For all trials, the mean joint moments for both nID and yID taken over the entire trial, is shown in Table V. The invalid results within this table are ones with unrealistic high force spikes, as can be seen in the corresponding graphs in Appendix G.

B. Stander

1) Lumbar: Unlike the explanation of the BDP-BUP periods in the POT cycle from Figure 2, handling the stander involved keeping the back nearly straight for the entire duration of the POT cycle, which can be seen in Figure 5 in the case of the steel deck. This resulted in a relatively stable lumbar extension, with no visible changes between the nID and yID case.

However, both the lumbar rotation and bending increased drastically with the yID case. As can also be seen in Figure 7, there was a more than 10% increase in the peak-to-peak changes in rotation moments. This mainly happened during the period during which the subject had to alternate their arms, leaving one arm to handle all the weight.

2) Shoulder: All shoulder moments increased in the yID case. And again, due to the alternating of the arms, the peak-to-peak between left and right arm also resulted in the quick changing of this moment.

Although longer materials would prevent the need for the arm to reach above shoulder height, the shoulder flexion on both arms reaches above 60 degrees regularly. This mainly occurred in the dominant arm's shoulder

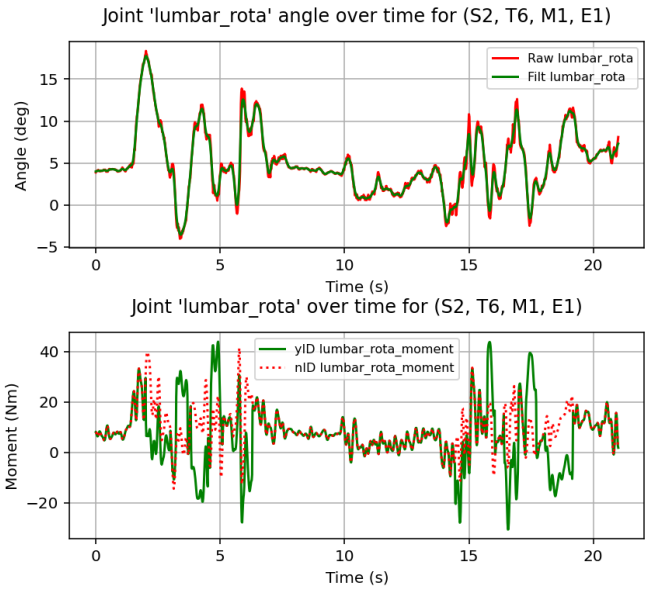


Fig. 7: Resulting joint angle and joint moment for the lumbar rotations over time, with both the passing-up (0-7s) and passing-down (13-21s) of the stander material. (Top figure) Joint angle, showing both the non-filtered (raw) and filtered (filt) angles over time. (Bottom figure) Joint moment, with yID including the EFF, and nID excluding the EFF.

C. Ledger and Console

The ledger and console both showed nearly no increase in joint moments comparing the nID and yID states, but did show a total increase in joint moments over the stander. The total average moment of each joint resulted in 31,3 Nm for the stander, and 39,8 Nm and 40,3 Nm for the ledger and console respectively.

1) Lumbar: Both the ledger and console show higher lumbar extension and bending compared to the stander, both within the subjects and also within the average means. Due to those two materials being relatively short, the subjects had to flex forward, typically to around 40 degrees, to reach over the scaffolding railing to grab the material. The increased bending moment was be attributed to the subject reaching out with one arm.

2) Shoulder: Similarly to the stander, the flexion moment in the dominant right shoulder is noticeably higher, with an about 55% and 20% increase for the ledger and console respectively.

D. Inside Passing

Due to the lack of valid results with inside passing trials, there is only one direct comparison between inside/outside. For subject 1, trial 4 and 5 involve the same material, and passing the ledger along the outside and inside respectively. Both in trial 4 and 5, the subject performs a stoop lift. From Table V, mainly the lumbar bending moment shows a difference, with about 7% increase in moment for passing it via the inside. The other moments do not show a high difference. This does

TABLE V: Resulting mean absolute joint moments (Nm) + standard deviation, over the duration of the trial. Additionally, the average mean for each subject S1, S2, and S3 is noted, as well as the average mean for each material M1, M2, and M3. For shoulder flexions, due to the flipped direction, a lower moment indicates more intense experience. inv (invalid trial results, mostly due to sudden spikes of unrealistic high forces), * (>1.05 increase), ** (>1.10 increase)

Trial	**Lumbar extension moment**		**Lumbar rotation moment**		**Lumbar bending moment**		**Shoulder flexion moment left**		**Shoulder flexion moment right**		**Shoulder adduction moment left**		**Shoulder adduction moment right**	
S-T-M-OI	nID	yID	nID	yID	nID	yID	nID	yID	nID	yID	nID	yID	nID	yID
1-1-1-1	57,50 ± 21,20	56,77 ± 21,65	8,49 ± 6,35	13,19 ± 9,82	19,61 ± 18,76	24,90 ± 22,88	18,36 ± 10,87	16,47 ± 9,57	16,08 ± 10,05	13,08 ± 8,28	42,07 ± 5,22	44,71 ± 5,52	39,89 ± 4,77	42,48 ± 4,35
1-2-1-1	53,45 ± 20,40	52,69 ± 21,74	8,10 ± 6,87	11,88 ± 10,29	21,74 ± 16,89	23,29 ± 18,41	21,02 ± 15,41	19,33 ± 14,98	18,49 ± 12,02	16,28 ± 11,35	39,91 ± 6,25	43,22 ± 8,01	39,43 ± 4,51	42,16 ± 4,75
1-3-1-1	61,88 ± 24,47	61,62 ± 25,48	8,24 ± 5,38	11,50 ± 9,66	17,48 ± 15,88	19,06 ± 19,07	19,06 ± 12,29	17,92 ± 11,88	19,46 ± 13,47	17,09 ± 12,95	40,49 ± 5,67	43,20 ± 6,44	38,49 ± 5,72	41,75 ± 5,96
1-4-2-1	97,88 ± 74,93	99,19 ± 76,91	inv	inv	23,52 ± 19,83	23,76 ± 20,67	19,35 ± 16,17	18,88 ± 16,07	23,46 ± 18,30	22,96 ± 17,91	42,76 ± 6,91	43,35 ± 6,67	40,02 ± 8,00	41,69 ± 8,15
1-5-2-2	inv	inv	15,22 ± 9,46	15,61 ± 9,15	25,37 ± 24,70	25,83 ± 25,31	18,92 ± 16,67	18,61 ± 16,48	24,29 ± 19,22	23,83 ± 18,90	40,23 ± 7,71	41,04 ± 7,20	39,45 ± 8,01	40,64 ± 8,17
1-7-3-2	inv	inv	10,83 ± 8,02	12,26 ± 10,07	22,99 ± 23,37	23,12 ± 23,90	21,20 ± 24,94	21,31 ± 24,80	20,12 ± 24,32	19,62 ± 24,34	40,82 ± 7,63	42,25 ± 8,92	40,61 ± 9,30	43,71 ± 12,35
1-8-3-2	inv	inv	9,34 ± 7,14	13,17 ± 12,84	22,35 ± 22,93	22,85 ± 24,29	19,11 ± 18,36	19,12 ± 18,21	20,43 ± 22,42	19,97 ± 22,42	40,11 ± 8,86	41,54 ± 9,95	40,48 ± 8,19	43,87 ± 12,08
2-4-3-1	79,75 ± 78,54	83,15 ± 77,16	inv	inv	23,89 ± 31,33	24,88 ± 31,86	21,32 ± 25,45	20,91 ± 25,26	34,45 ± 53,73	34,54 ± 54,98	41,41 ± 9,36	43,04 ± 8,05	39,19 ± 13,00	41,38 ± 12,02
2-6-1-1	49,17 ± 22,16	49,60 ± 22,91	10,07 ± 7,19	10,50 ± 8,70	14,45 ± 14,12	16,96 ± 15,85	15,57 ± 12,69	14,94 ± 12,81	17,40 ± 11,41	15,21 ± 10,51	39,55 ± 7,25	41,42 ± 6,37	38,95 ± 5,90	41,62 ± 5,63
2-7-1-1	45,28 ± 21,94	46,17 ± 23,27	9,63 ± 6,69	10,55 ± 9,25	16,62 ± 13,76	21,44 ± 17,33	13,51 ± 12,23	12,79 ± 11,70	16,89 ± 15,80	15,25 ± 14,47	41,19 ± 8,50	43,81 ± 6,92	39,11 ± 6,26	42,41 ± 7,03
2-9-2-2	inv	inv	inv	inv	26,58 ± 27,21	26,45 ± 26,77	15,65 ± 15,32	15,50 ± 15,41	inv	inv	37,79 ± 6,09	38,21 ± 6,14	inv	inv
3-1-2-1	118,40 ± 38,82	119,53 ± 38,88	13,34 ± 8,63	12,39 ± 8,30	42,27 ± 31,86	42,91 ± 31,82	15,32 ± 13,22	14,79 ± 12,66	34,39 ± 74,57	33,95 ± 74,86	43,12 ± 4,73	43,81 ± 3,73	38,17 ± 8,67	39,61 ± 8,42
3-2-2-1	inv	inv	16,28 ± 14,47	15,57 ± 14,46	38,66 ± 36,54	38,99 ± 36,86	14,77 ± 13,86	14,42 ± 13,84	22,43 ± 19,61	22,12 ± 19,68	43,46 ± 7,63	43,88 ± 7,33	38,42 ± 6,97	38,89 ± 7,24
3-5-1-1	111,26 ± 35,79	111,18 ± 36,97	10,77 ± 8,03	11,77 ± 9,31	25,46 ± 21,36	29,89 ± 26,48	16,15 ± 14,93	15,29 ± 14,24	25,61 ± 23,91	23,73 ± 22,87	38,93 ± 6,54	40,99 ± 6,77	36,70 ± 7,16	40,47 ± 7,39
3-6-1-1	inv	inv	18,93 ± 12,37	16,93 ± 10,57	48,98 ± 32,38	49,96 ± 31,05	15,27 ± 14,71	14,22 ± 14,06	34,08 ± 54,74	31,66 ± 55,25	42,69 ± 7,31	44,41 ± 6,73	35,56 ± 8,54	39,47 ± 8,50
3-7-1-1	inv	inv	14,82 ± 8,37	15,88 ± 10,17	34,13 ± 20,44	37,49 ± 23,45	21,26 ± 18,84	19,55 ± 18,33	19,85 ± 18,79	17,34 ± 17,98	40,01 ± 9,00	42,48 ± 8,82	35,99 ± 6,04	39,30 ± 6,91
3-8-3-1	123,11 ± 40,57	124,80 ± 40,58	8,36 ± 6,51	9,06 ± 6,85	44,05 ± 37,66	45,13 ± 37,31	21,75 ± 23,01	20,51 ± 22,95	29,57 ± 31,66	28,83 ± 31,80	40,45 ± 8,20	42,09 ± 7,49	39,48 ± 9,25	41,74 ± 8,27
3-9-3-1	126,16 ± 42,86	126,33 ± 45,37	16,27 ± 15,73	16,00 ± 15,79	46,79 ± 35,79	46,97 ± 35,16	20,38 ± 19,31	18,99 ± 19,06	19,74 ± 17,50	17,99 ± 16,40	40,91 ± 7,58	42,12 ± 7,27	39,50 ± 6,92	41,91 ± 6,81
Avg μ S1	67,7	67,6	10	12,9**	21,9	23,3*	19,6	18,8	20,3	19*	40,9	42,8	39,8	42,3*
Avg μ S2	58,1	59,6	9,7	11,4**	20,8	22,5*	17	16,7	22,3	21,2*	40	41,6	39,4	42,3*
Avg μ S3	119,7	120,5	14,1	13,9	40	41,6	17,8	16,8*	26,5	25,1*	41,4	42,8	37,7	40,2*
Avg μ M1	63,1	63	11,1	12,8**	24,8	27,9**	17,5	16,3*	21	18,7**	40,6	43*	38	41,2*
Avg μ M2	108,1	109,4	14,9	14,5	31,3	31,6	16,8	16,4	26,1	25,7	41,5	42,1	39	40,2
Avg μ M3	109,7	111,4	11,2	12,6**	32	32,6	20,8	20,2	24,9	24,2	40,7	42,2	39,9	42,5*

not take into account the compensatory actions that will be discussed in the next part.

The four inside trials also show fairly similar techniques being done, as is shown in Figure 8. The subjects reported that they perceive the full-squat method to be more tiring than the semi-squat, with the outside passing being the least tiring.

E. Compensatory Movements

During the experiments, the subjects performed compensatory movements that are not included in the biomechanical analyses.

Within Figure 8, the subjects used the non grabbing arm to either lean on their own body, or to hold the scaffolding railing

More compensatory actions were found during the pass via the outside. In addition to the left arm leaning, Figure 9 shows the subject performing a golfer's lift, which was done with swinging one leg backward. Between the results of trial 9 and 8 of subject 3, the lumbar extension and bending moments do not change, but the lumbar rotation moment is almost half in trial 8.

Another compensatory action in Figure 9 was the leaning of the trunk against the railing.

A combination of all these compensatory actions should lower the resulting moments in the lumbar regions in Table V, but there would be no change in the shoulder region.

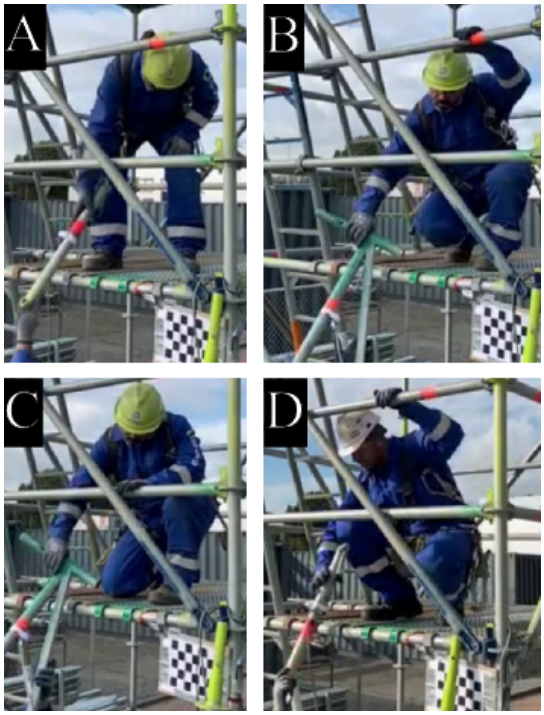


Fig. 8: Techniques applied when first grabbing the materials through the inside of the scaffold. (A) S1-T5 Stoop with hand on thigh, (B) S1-T7 Asymmetric semi-squat with hand on upper railing, (C) S1-T8 Asymmetric semi-squat with hand on bottom railing, (D) S2-T9 Semi-squat with hand on upper railing.

V. Discussion

A. Experimental Method

1) In-situ: The setup of an *in-situ* experimental setup with the combination of OpenCap and OpenSim is largely successful.

With the experiment as close to real-world conditions as was possible, the results showed an accurate representation of the scaffolding work. Overall, the movements of the POT of the scaffolders were consistent with each trial, and as such created a viable way of directly comparing trials.

However, the setup of this scaffold is of an ideal situation, with little additional environmental constraints that my occur elsewhere in the factory site. The results therefore lack the

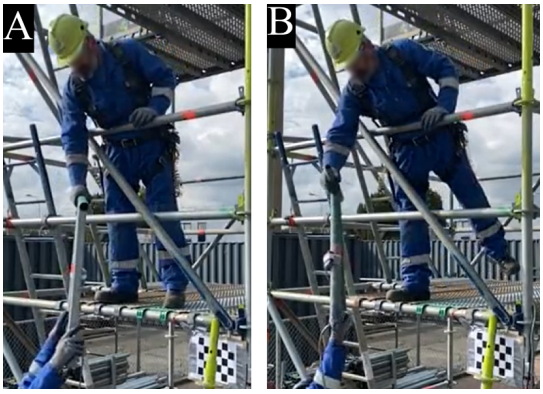


Fig. 9: Example of compensatory movements while passing-up. (A) S3-T9 Standing with both legs while leaning arm and trunk against railing. (B) S3-T8 Performing golfer's lift also while leaning arm and trunk against railing.

additional bodily impacts that can occur with these constraints.

Another limitation in the *in-situ* is the limited amount of materials being passed-up/down at one time. Typically the top worker constructing the scaffold takes some time to take over material, and as such a queue can form in between. For smaller materials, the workers mostly lean the material on the scaffold, but this is challenging with the larger materials. This would likely create additional impact on the body, which is not shown in this experiment.

2) OpenCap: OpenCap's kinematic capturing showed useful results, considering this software was not originally created to be used for these types of movements and environments.

The loss of 16 of the 33 trials does show the fragility of the software, with the majority of them being of inside passings. The captured videos for download would start too late, which resulted in the time not being able to be synchronized, or stopped before a full pass-up was even done. Why these mishaps happened in this specific way is not clear.

One likely explanation of why these trials were cut short was segment occlusion, where one of the two cameras does not see a segment, and thus cuts the trial short. When the subject passes material through the inside, most of their body were blocked by the scaffolding railing. This problem could be mitigated with the inclusion of more cameras at different angles, both horizontally and vertically.

Another more likely cause of these losses were the poor contrast between pieces of clothing. As the Bilfinger coverall is mainly monochrome, separations between segments was more difficult for OpenCap. This was especially true when body segments overlap a lot and if shadows were created, as seen in Figure 9. This problem is inherent to the *in-situ* experiment, in which the coverall is a vital piece of protective clothing. Having an experiment with a special coverall, one which has different colors for different body segments, may improve the trial validity.

The use of an coverall, compared to everyday clothing, may have hampered the joint angle accuracies. Within the literature, there are no tests done with OpenCap with this type of coverall. A comparisons of clothing conditions in OpenCap was one by Horsak et al. [30], in which the subjects perform repeatable movements undressed and with everyday clothing, wearing mainly jeans and semi-loose t-shirt. Keller et al. [31] compared bsubjects wearing 'casual' everyday clothing between those wearing sportswear, resulting in similar OpenCap joint angles. The subjects of these studies who wore loose jeans and jumpers seem to not influence the results as well. These clothing types have similar loosely-ness to an coverall.

The resulting problems in OpenCap may therefore not be because of the loosely coverall, but only due to the monochrome color.

The steel deck is also a clear indicator of the limitations of OpenCap. Again, having more cameras, placed on the side angles of the subject, are likely to prevent obstruction by the material similar to those shown in Figure 5.

B. Results

1) Validation Models: The use of the UMocoD model was to acquire muscle data from the upper extremities. Unfortunately due to the problems with SO, the inclusion of the UMocoD model was redundant.

These problems with SO are attributed to the lack of supporting data, such as a ground plate to measure ground reaction forces, or electromyography (EMG) data for the activation levels of the muscles. These likely would have helped produce relevant SO results.

For future usage of full-body muscles in the UMocoD model, the model should more closely be matched to the Lai-Uhlrich model. Although the correlations are acceptable, considering the UMocoD model was minimally altered, improvements in the accuracy can be made.

2) Subjects: Although only three subjects have performed the experiment, the similar body structures (height, weight, and dominant hand) resulted in the data being viable to compare between the subjects.

Future research should broaden the demographic of the subjects to strengthen the validity of statistical significance of this experiment.

This experimental tool seems to have potential for evaluating movements of new scaffold workers, to correct harmful movements in an early stage.

3) Valid Trials: The trials that were valid for use have 8 stander, 5 ledger, and 4 consoles for the material. As the ledger and console are fairly similar in size and handling methods, a total of 8 'large' and 9 'small' materials allowed for good comparisons between the materials.

The comparison between inside and outside passing is more lackluster, with all inside passings having invalid results for the lumbar extension moment. Again, this can be attributed to the OpenCap body segment overlap problem.

4) Inverse Dynamics: The choice for the lumbar and shoulder joints being included is mainly due to them having changes with external forces, and are, not very coincidentally, the two musculoskeletal areas that have the highest reports of musculoskeletal disorders [1]. This does not include the shoulder rotation, due to the aforementioned OpenCap issues.

There is an interesting note on the discrepancies in the mean joint moment between subjects 1 and 2, and subject 3. The mean lumbar extension and bending moments are about twice as large in subject 3 compared to subject 1 and 2. This is most likely attributed to the subject performing exaggerated movements compared to the other two subjects. Subject 3 had the tendency to lean forward more during the BP event, as is especially noticeable in Figure 9, which would increase the extension and bending moments.

5) Stander: The increased lateral shear loading on the lumbar regions may be the most critical high impact part of the POT. The passing of larger materials (10 kg), such as the stander, steel deck, and also the larger format ledgers, is most commonly done in the POT.

Although no literature was found which investigates the passing of an object by repeatedly alternating arms like in the POT, it can be partially compared to literature which compares the lifting of materials with one arm compared to using two arms. Weston et al. [32], and Marras and Davis [33] both showed increasing lateral shear forces when performing one arm lifts, resulting from increased bending and rotation forces due to the asymmetric loading.

This increased lateral shear loading on the spine causes an increased degradation in intervertebral disc, and results in a smaller intervertebral foramen, the occlusion in a spinal disc in which the nerves pass through [34] [35]. This in turn increases risks for general lower back pain and spinal stenosis [36] [37].

This asymmetric loading due to alternating arms may therefore be a probable cause for the lower back problems within scaffolders, and is the most important factor found in this thesis.

Additionally, the excessive shoulder flexions occurring are also a critical point. This is shown to increase the risk for shoulder tendinopathies [22] [23]. With the subjects using the right arm for both the BP and LP, this angle increased, and also resulted in the 15% higher average shoulder flexion moment compared to the left arm.

Although this excessive flexion problem is less prevalent materials larger than the 2 meter stander, being able to find a solution that is effective for both large and small materials may be helpful.

6) Ledger and Console: Although the ledger and console show a higher total bending moment compared to the larger material, the frequency of these small materials being passed up is smaller. A key note that these results do not take into account is the use of the compensatory actions involved that potentially alleviate the lumbar region.

As the subjects consistently use their dominant arm for the BP and LP events, this asymmetric effect may have a higher impact on the lumbar region.

The discrepancy between the left and right arm are even more noticeable compared to the larger materials in the shoulders.

A 'quick fix' for the scaffolders would be to involve their non-dominant arm in the BP and LP events for about 50% of the time to balance out these lumbar and shoulder moments.

7) Inside Passing: Due to the lack of direct comparisons between inside and outside passings, drawing a conclusion is difficult. The direct comparison between subject 1, trial 4 and 5 performing a stoop lift via the outside and inside respectively only showed a 7% increase in the lumbar bending in the inside passing. However, this does not take into account the compensatory action of leaning the hand on the thigh.

The use of semi-squats and it's linking to fatigue supports the findings in literature [18] [19].

Future research is necessary to compare the differences between inside stoop-lift and outside stoop-lift passings, both with and without hand leanings against the thigh/railing. If the complications with OpenCap can be resolved, the addition of squats and semi-squats should also be done. Within these experiments, the measuring of the subject's metabolism can be included to validate the reporting of the subjects with these different techniques.

With the relative low frequency of small materials needing to be passed and little differences in moments, there is likely little problems with the subjects performing the passing via the outside. This would prevent the increased tiring as a result of performing (semi-)squats [18] [19].

8) Compensatory Movements: The usage of different compensatory movements, although not directly measured, does show to have a positive impact as was shown in the literature. Kingma et al. [38] showed that using the hand as a support on the thigh reduces the forces in the lumbar region by 13-19% compared to just one-handed lifting. The leaning of the trunk against the railing also was shown to alleviate lumbar loading [39] [40].

Although the improvements in lumbar region might be positive, these techniques may result in excessive loading in for example the shoulder and wrist areas in the case of arm leaning, or with pressure loading in the trunk area.

The golfer's lift technique is more controversial [41]. Wilson et al. [41] showed a similar technique as used by the subjects. Their results showed decreased lumbar flexion loading when using this technique, but it did cause a higher muscle compressive loading of the spine.

The subjects in Wilson et al. [41] did mention less back-pain following the experiment while performing the golfer's lift compared to the stoop lift. However, this movement could lead to counter-productive asymmetric loading, similar to the alternating arms problem, if not handled properly [42].

Future research could involve these compensatory actions into the external forces within OpenSim, by for example placing force sensors on the railing to measure the reaction force when the subject leans against it with their trunk or hand. With these, the potential positive and negative effects on the lumbar region and shoulder/wrist region can be investigated.

VI. Future works

Several recommendations on future works, that includes but is not limited to the scaffolding industry, based on this thesis are made:

A. Future research

1) In-situ improvements: Due to the idealized POT situation lacking additional environmental constraints that otherwise

occurs, such as tight spaces and the queue of materials, the full extent of the POT is not yet known. By performing the same experiment during normal working conditions, the potential extra impact can be uncovered.

Challenges in these include the placement of cameras in these constrained environments, and the difficulty with the kinetic module being on only one material. Alternatives using video tracking of the material can perhaps be used.

2) Coverall vs Normal Clothing: The use of a monochrome coverall showed potential problems. By conducting controlled experiments comparing a coverall to the recommended clothing types can validate joint accuracies in this thesis.

3) Monochrome vs Multicolor: By directly comparing monochrome against a multi color clothing, the limits on OpenCap's tracking during dynamic movements can be assessed. Movements which result in body segments overlapping, such as squats shown in Figure IV-D, should be the focus.

4) Additional Capturing Devices: What effect does the addition of extra cameras have on experiments with drastic occlusions, such as the steel deck in Figure 5.

5) Compensatory Actions: Integrate additional force sensors that measure reaction forces due to the compensatory actions of the scaffold workers. With these, investigations on different methods, such as the golfer's lift or trunk leaning, can more effectively be compared against recommended movements such as the full-squat. With these, better ergonomic alternatives can be found for other industries as well.

6) Static Optimization: Integrate additional sensors, such as IMUs, EMGs, and force plates, as a support for the SO tool within OpenSim.

B. Product Development

1) Ergonomic Intervention: With the high potential of the *in-situ* setup, tracking and evaluating industrial tasks can become an useful tool for ergonomic interventions. Employees can be filmed during their respective tasks, and subsequently be corrected if deemed necessary.

2) Lumbar Solutions: With the results showing on the lumbar bending and lumbar rotations during the moving of large materials, alleviating these can prevent WMSDs. A solution can be figured out that tackles these two movements, but still allow enough flexibility to grab smaller materials and do not intervene with the fall harness.

3) Shoulder Solutions: The excessive shoulder flexion angles/moments and adduction moments show improvements to be had. Creating a device which either prevents this movement entirely or aids in the movement, but do not intervene in other ways can be tried.

VII. Conclusion

This thesis aimed to answer the research question:

Can a combination of ergonomic risk assessment and an in-situ in-silico approach identify high-impact musculoskeletal areas in a heavy industrial task, thereby guiding targeted intervention strategies?

Via an ergonomic risk assessment, the passing-on task was selected. OpenCap was used to capture the kinematics, a kinetic module to capture acceleration and forces, after which OpenSim was utilized to perform the biomechanical analysis. This combination has shown to create viable results in identifying high-impact musculoskeletal areas, and enabling the informing for future development. Therefore, the research question can be considered successfully answered.

The results showed that the lumbar and shoulder regions were the most affected when handling material, consistent with prevalence reports of lower back and shoulder pain.

With the inclusion of the external forces for the stander material, the analyses over the entire POT showed an over 10% increase in mean lumbar bending and rotation moments. These result in an increased risk of lumbar disorders, and is therefore a key takeaway from this thesis. Additionally, the dominant arm's shoulder flexion and adduction moments increasing over 5% with the stander. Shoulder flexion angles exceeded 60 degrees consistently, which has increased risks of its own.

For the layer and console materials, no changes with the inclusion of the external forces were found. The total joint moments did increase over the stander, mainly in the lumbar flexion, showing that the movement involved in having to pass on these materials have a higher impact, albeit the frequency that these materials are passed on are lower.

Fragility in OpenCap led to the data loss of 14 out of 31 trials, likely due to obstruction of body segments or low-contrast conditions caused by the monochrome Bilfinger coverall. Eleven of these lost trials were from inside passings, causing a lack of effective analysis for these.

Compensatory movements were found to be used by the subjects, such as leaning on the railings with the arms or trunks, and performing a golfer's lift. The former two are shown to have positive effects on the lumbar region, but it is unclear what the effect on the shoulder/wrist region might be. The latter shows an overall negative impact on the lumbar. Although lumbar moments decrease, the asymmetric movement might result in more harm.

Future works can focus on additional research, such as improving OpenCap's functionality in the industrial field and studying the effect of the compensatory actions, or on product development, with targeted interventions on the critical areas.

References

[1] J. de Kok, P. Vroonhof, J. Snijders, G. Roullis, M. Clarke, K. Peereboom, P. van Dorst, and I. Isusi, "Work-related MSDs: prevalence, costs and demographics in the EU [Statistics amount of WMSD]," *European Agency for Safety and Health at W ork*. [Online]. Available: https://osha.europa.eu/sites/default/files/Work_related_MSDs_prevalence_costs_and_demographics_in_EU_summary.pdf

[2] A. Bassey Etuknwa and S. Humpheries, "A Systematic Review on the Effectiveness of Ergonomic Training Intervention in Reducing the Risk of Musculoskeletal Disorder," *Journal of Nursing and Health Studies*, vol. 03, no. 02, 2018. [Online]. Available: http://www.imedpub.com/articles/a-systematic-review-on-the-effectiveness-of-ergonomic-training-intervention-in-reducing-the-risk-of-musculoskeletal-disorder.php?aid=23102

[3] V. C. Hoe, D. M. Urquhart, H. L. Kelsall, E. N. Zamri, and M. R. Sim, "Ergonomic interventions for preventing work-related musculoskeletal disorders of the upper limb and neck among office workers," *Cochrane Database of Systematic Reviews*, vol. 2018, no. 10, Oct. 2018. [Online]. Available: http://doi.wiley.com/10.1002/14651858.CD008570.pub3

[4] A. Cardoso, A. Ribeiro, P. Carneiro, and A. Colim, "Evaluating Exoskeletons for WMSD Prevention: A Systematic Review of Applications and Ergonomic Approach in Occupational Settings," *International Journal of Environmental Research and Public Health*, vol. 21, no. 12, p. 1695, Dec. 2024. [Online]. Available: https://www.mdpi.com/1660-4601/21/12/1695

[5] C. Ammendolia, M. S. Kerr, and C. Bombardier, "Back Belt Use for Prevention of Occupational Low Back Pain: A Systematic Review," *Journal of Manipulative and Physiological Therapeutics*, vol. 28, no. 2, pp. 128–134, Feb. 2005. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0161475405000102

[6] R. Vijayakumar and J.-h. Choi, "Emerging Trends of Ergonomic Risk Assessment in Construction Safety Management: A Scientometric Visualization Analysis," *International Journal of Environmental Research and Public Health*, vol. 19, no. 23, p. 16120, Dec. 2022. [Online]. Available: https://www.mdpi.com/1660-4601/19/23/16120

[7] L. Zheng, B. Lowe, A. L. Hawke, and J. Z. Wu, "Evaluation and Test Methods of Industrial Exoskeletons In Vitro, In Vivo, and In Silico: A Critical Review," *Critical reviews in biomedical engineering*, vol. 49, no. 4, pp. 1–13, 2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9199587/

[8] I. Campanini, C. Disselhorst-Klug, W. Z. Rymer, and R. Merletti, "Surface EMG in Clinical Assessment and Neurorehabilitation: Barriers Limiting Its Use," *Frontiers in Neurology*, vol. 11, p. 934, Sep. 2020. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7492208/

[9] A. Seth, M. Sherman, J. A. Reinbolt, and S. L. Delp, "OpenSim: a musculoskeletal modeling and simulation framework for in silico investigations and exchange," *Procedia IUTAM*, vol. 2, pp. 212–232, 2011. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2210983811000228

[10] F. Smith, F. Brownlie, T. Hodgkiess, A. Toumpis, A. Pearson, and A. Galloway, "Effect of salinity on the corrosive wear behaviour of engineering steels in aqueous solutions," *Wear*, vol. 462-463, p. 203515, Dec. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0043164820309741

[11] H. P. Tiwari, R. Sharma, R. Kumar, P. Mishra, A. Roy, and S. K. Haldar, "A review of coke making by-products," *Coke and Chemistry*, vol. 57, no. 12, pp. 477–484, Dec. 2014. [Online]. Available: http://link.springer.com/10.3103/S1068364X14120072

[12] S. Hignett and L. McAtamney, "Rapid entire body assessment (REBA)," *Applied Ergonomics*, vol. 31, no. 2, pp. 201–205, Apr. 2000.

[13] *Musculoskeletal Disorders and the Workplace: Low Back and Upper Extremities*. Washington, D.C.: National Academies Press, May 2001, pages: 10032. [Online]. Available: http://www.nap.edu/catalog/10032

[14] A. Kharb, V. Saini, Y. K. Jain, and S. Dhiman, "A REVIEW OF GAIT CYCLE AND ITS PARAMETERS," vol. 13, 2011.

[15] "OSHA 10 Hour Construction & General Industry Training by OSHA-Pros." [Online]. Available: https://www.osha-pros.com/

[16] M. v. S. Z. e. Werkgelegenheid, "Wat zegt de wet over tillen en dragen? - Tillen en dragen - Arboportaal," Jul. 2020, last Modified: 2024-03-20T17:00 Publisher: Ministerie van Sociale Zaken en Werkgelegenheid. [Online]. Available: https://www.arboportaal.nl/onderwerpen/tillen-en-dragen/wat-zegt-de-wet-over-tillen-en-dragen

[17] P. Dolan, M. Earley, and M. A. Adams, "Bending and compressive stresses acting on the lumbar spine during lifting activities," *Journal of Biomechanics*, vol. 27, no. 10, pp. 1237–1248, Oct. 1994. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0021929094902771

[18] N. B. Washmuth, A. D. McAfee, and C. S. Bickel, "Lifting Techniques: Why Are We Not Using Evidence To Optimize Movement?" *International Journal of Sports Physical Therapy*, vol. 17, no. 1, Jan. 2022. [Online]. Available: https://ijspt.scholasticahq.com/article/30023-lifting-techniques-why-are-we-not-using-evidence-to-optimize-movement

[19] L. Straker, "Evidence to support using squat, semi-squat and stoop techniques to lift low-lying objects," *International Journal of Industrial Ergonomics*, vol. 31, no. 3, pp. 149–160, Mar. 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169814102001919

[20] I. Kingma, G. S. Faber, A. J. Bakker, and J. H. Van Dieën, "Can Low Back Loading During Lifting Be Reduced by Placing One Leg Beside the Object to Be Lifted?" *Physical Therapy*, vol. 86, no. 8, pp. 1091–1105, Aug. 2006. [Online]. Available: https://academic.oup.com/ptj/article/86/8/1091/2857438

[21] M. Wærsted, M. Koch, and K. B. Veiersted, "Work above shoulder level and shoulder complaints: a systematic review," *International Archives of Occupational and Environmental Health*, vol. 93, no. 8, pp. 925–954, Nov. 2020. [Online]. Available: https://link.springer.com/10.1007/s00420-020-01551-4

[22] B. A Rhode, "Occupational Risk Factors for Shoulder Tendon Disorders 2015 Update," *MOJ Orthopedics & Rheumatology*, vol. 3, no. 4, Nov. 2015. [Online]. Available: https://medcraveonline.com/MOJOR/occupational-risk-factors-for-shoulder-tendon-disorders-2015-update.html

[23] A. J. McClellan, W. J. Albert, S. L. Fischer, F. A. Seaman, and J. P. Callaghan, "Shoulder loading while performing automotive parts assembly tasks: A field study," *Occupational Ergonomics*, vol. 8, no. 2-3, pp. 81–90, Mar. 2009. [Online]. Available: https://journals.sagepub.com/doi/full/10.3233/OER-2009-0162

[24] S. D. Uhlrich, A. Falisse, Kidziński, J. Muccini, M. Ko, A. S. Chaudhari, J. L. Hicks, and S. L. Delp, "OpenCap: Human movement dynamics from smartphone videos," *PLOS Computational Biology*, vol. 19, no. 10, p. e1011462, Oct. 2023. [Online]. Available: https://dx.plos.org/10.1371/journal.pcbi.1011462

[25] X. Cheng, Y. Jiao, R. M. Meiring, B. Sheng, and Y. Zhang, "Reliability and validity of current computer vision based motion capture systems in gait analysis: A systematic review," *Gait & Posture*, vol. 120, pp. 150–160, Jul. 2025. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0966636225001791

[26] "SimTK: Moco simulations at University of Maryland: Project Home." [Online]. Available: https://simtk.org/projects/umocod

[27] P. Mai and S. Willwacher, "Effects of low-pass filter combinations on lower extremity joint moments in distance running," *Journal of Biomechanics*, vol. 95, p. 109311, Oct. 2019.

[28] E. Kristianslund, T. Krosshaug, and A. J. van den Bogert, "Effect of low pass filtering on joint moments from inverse dynamics: implications for injury prevention," *Journal of Biomechanics*, vol. 45, no. 4, pp. 666–671, Feb. 2012.

[29] T. E. Hewett, G. D. Myer, B. D. Roewer, and K. R. Ford, "Letter to the editor regarding "Effect of low pass filtering on joint moments from inverse dynamics: implications for injury prevention"," *Journal of biomechanics*, vol. 45, no. 11, pp. 2058–2060, Jul. 2012. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4943337/

[30] B. Horsak, H. Kainz, and B. Dumphart, "Repeatability and minimal detectable change including clothing effects for smartphone-based 3D markerless motion capture," *Journal of Biomechanics*, vol. 175, p. 112281, Oct. 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021929024003592

[31] V. T. Keller, J. B. Outerleys, R. M. Kanko, E. K. Laende, and K. J. Deluzio, "Clothing condition does not affect meaningful clinical interpretation in markerless motion capture," *Journal of Biomechanics*, vol. 141, p. 111182, Aug. 2022.

[32] E. B. Weston, A. Aurand, J. S. Dufour, G. G. Knapik, and W. S. Marras, "Spinal Loading During One and Two-Handed Lifting," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 63, no. 1, pp. 1126–1127, Nov. 2019, publisher: SAGE Publications Inc. [Online]. Available: https://doi.org/10.1177/1071181319631067

[33] W. S. Marras and K. G. Davis, "Spine loading during asymmetric lifting using one versus two hands," *Ergonomics*, vol. 41, no. 6, pp. 817–834, Jun. 1998.

[34] A. Fujiwara, H. S. An, T.-H. Lim, and V. M. Haughton, "Morphologic Changes in the Lumbar Intervertebral Foramen Due to Flexion-Extension, Lateral Bending, and Axial Rotation: An In Vitro Anatomic and Biomechanical Study," *Spine*, vol. 26, no. 8, pp. 876–882, Apr. 2001. [Online]. Available: http://journals.lww.com/00007632-200104150-00010

[35] B. H. Nowicki, V. M. Haughton, T. A. Schmidt, T.-H. Lim, H. S. An, L. H. R. Iii, L. Yu, and J.-W. Hong, "Occult Lumbar Lateral Spinal Stenosis in Neural Foramina Subjected to Physiologic Loading."

[36] B.-G. Peng, "Fundamentals of intervertebral disc degeneration and related discogenic pain," *World Journal of Orthopedics*, vol. 16, no. 1, Jan. 2025. [Online]. Available: https://www.wjgnet.com/2218-5836/full/v16/i1/102119.htm

[37] Yusof, H. Mn, A. Ms, and Department of Radiology, Universiti Sains Malaysia, Kubang Kerian, Malaysia, "The Relationship amongst Intervertebral Disc Vertical Diameter, Lateral Foramen Diameter and Nerve Root Impingement in Lumbar Vertebra," *Malaysian Orthopaedic Journal*, vol. 12, no. 1, pp. 21–25, Mar. 2018. [Online]. Available: http://morthoj.org/2018/v12n1/interverterbal-disc-vertical-diameter.pdf

[38] I. Kingma, G. S. Faber, and J. H. Van Dieën, "Supporting the upper body with the hand on the thigh reduces back loading during lifting," *Journal of Biomechanics*, vol. 49, no. 6, pp. 881–889, Apr. 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0021929015005308

[39] M. Abdoli-Eramaki, C. Damecour, and A. Ghasempoor, "Biomechanics of leaning forward against a support with an extreme standing reach," *Ergonomics*, vol. 58, no. 2, pp. 208–219, Feb. 2015. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/00140139.2014.945489

[40] S. Jin and G. A. Mirka, "The effect of a lower extremity kinematic constraint on lifting biomechanics," *Applied Ergonomics*, vol. 42, no. 6, pp. 867–872, Nov. 2011. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0003687011000214

[41] D. J. Wilson, J. L. Gorham, and K. M. Hickey, "FORCES AND MOMENTS AT THE L41L5 VERTEBRAL LEVEL WHILE FORWARD BENDING IN A SUPPORTED POSTURE."

[42] D. M. Hooper, V. K. Goel, A. Aleksiev, K. Spratt, K. M. Bolte, and M. Pope, "Three dimensional moments in the lumbar spine during asymmetric lifting," *Clinical Biomechanics*, vol. 13, no. 6, pp. 386–393, Sep. 1998. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0268003398000084

[43] L. McAtamney and E. Nigel Corlett, "RULA: a survey method for the investigation of work-related upper limb disorders," *Applied Ergonomics*, vol. 24, no. 2, pp. 91–99, Apr. 1993. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/000368709390080S

[44] B. Buchholz, V. Paquet, L. Punnett, D. Lee, and S. Moir, "PATH: a work sampling-based approach to ergonomic job analysis for construction and other non-repetitive work," *Applied Ergonomics*, vol. 27, no. 3, pp. 177–187, Jun. 1996.

[45] E. Occhipinti, "OCRA: a concise index for the assessment of exposure to repetitive movements of the upper limbs," *Ergonomics*, vol. 41, no. 9, pp. 1290–1311, Sep. 1998.

[46] "OpenSim Documentation." [Online]. Available: https://opensimconfluence.atlassian.net/wiki/spaces/OpenSim/overview?homepageId=53084162

APPENDIX

A. Ergonomic Risk Assessment

1) Work Breakdown Structure: To identify all the different tasks and sub-tasks that an worker can encounter during their work, a WBS can be setup. Within this WBS each identified sub-task can then be evaluated for their respective intensity. As such, a clear and concise diagram can be constructed in which the resulting physical intensities can be presented. Heavily impacting sub-tasks can therefore be identified quickly.

2) Questionnaire: The questionnaire serves as a qualitative quick-assessment on the physical intensity of the sub-tasks as experienced by the workers, as well as resulting pain/discomfort as a result of the respective sub-tasks. This questionnaire can be used as a verification of the full ergonomic risk assessment.

The questionnaire that is used is shown in figure 10. Apart from the quantitative questions, a non-structured conversation is held to discuss the sub-task and its challenges at hand.

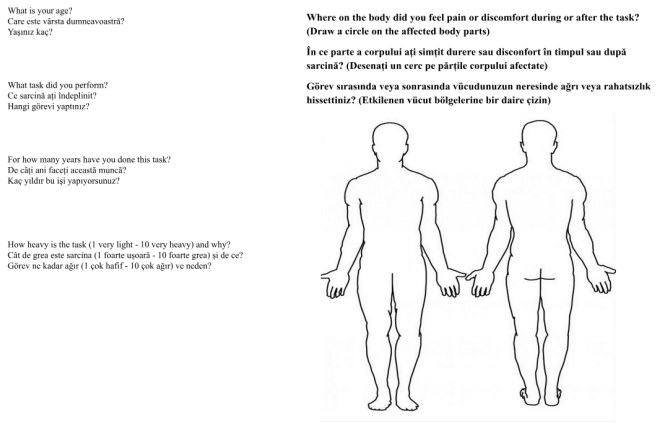


Fig. 10: Questionnaire provided to employees in multiple languages.

3) Method for Risk Assessment: To determine the potential physical risk of a sub-task, a combination of the intensity and the relative frequency of the task is plotted in a risk matrix. The combinations shows low, medium, and high risk of injury if the task is done over a prolonged time. The risk matrix can be seen in table I.

Intensity To assess the physical intensity of the sub-tasks, a choice between methods of postural risk assessments have to be made. In a review by Vijayakumar and Choi [6], several of these assessment are found: Rapid Upper Limb Assessment (RULA) [43], Rapid Entire Body Assessment (REBA) [12], Ovako Working Posture Analysis System (OWAS), and Posture, Activity, Tools, Handling (PATH) [44] are some to name.

RULA mainly evaluates the upper body. A coding system is used to generate a score, which results in an level of intervention required to reduce the risk of physical injury [43]. RULA is mainly useful if focusing on individual postures.

REBA is based on RULA, and includes lower extremity into the evaluation [12].

OWAS, unlike RULA and REBA, assesses all postures adopted throughout a task, at the cost of less precise eval-

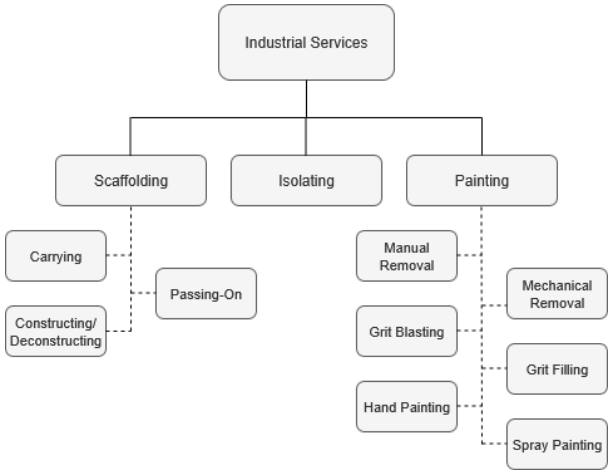


Fig. 11: The WBS of the overall industrial services tasks.

uations. This is done by the addition of a relative frequency within this method [6].

PATH was developed to characterize the ergonomic hazards of construction and other non-repetitive work [44]. By also including tools and the handling thereof, it encompasses most industrial work. However, the variety in different postures is lacking compared to the other methods. Additionally, it is a very thorough and therefore a time-consuming method.

Due to the high variety in relatively static and repetitive positions the workers have to work in, the REBA is chosen to be the assessment tool used to evaluate the intensity of a sub-task. REBA evaluates into 5 intensity levels. Additionally, a tool penalty has been added to the REBA scores. This is done in inspiration of PATH's method. This would result in tasks using heavy equipment leading to additional intensity compared to the same task using non-heavy equipment.

Relative Frequency The relative frequency is the amount of time a certain task is done in a time frame. Some tasks may be rare, and as such have an overall lower impact on the potential WMSD.

4) On-Site Visits: The WBS is constructed during on-site visits. In a two week period, the tasks and their respective sub-tasks are observed and assessed for the frequency and intensity, and quick interviews are held by using the questionnaires.

B. Results Work Breakdown Structure

After the two weeks on-site visiting period, three main tasks have been discovered: Scaffolding, isolating, and painting. The general consensus at Bilfinger is that isolating is not a physically intensive task, and as such this task was chosen to be left outside of the ERA. A diagram of the WBS is shown in figure 11, and pictures of the sub-tasks are shown in figure **??**.

1) Scaffolding: Scaffolding is the constructing of scaffolds. At the worksite, scaffolds are used by workers to complete a variety of tasks for a specific project. As such, scaffolds need to be constructed and deconstructed on a daily basis. At TATA Steel (and also other worksites of Bilfinger, including Shell), scaffolds have to be made of steel instead of aluminum, due to the environmental corrosive impact on the structures. This provides an additional weight challenge compared to typical scaffolds.

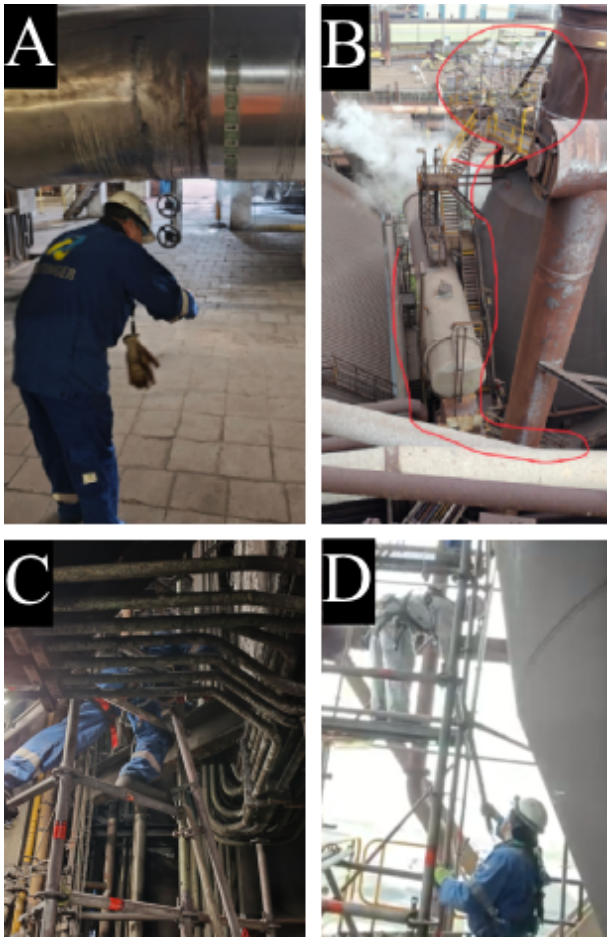


Fig. 12: Sub-tasks of scaffolders shown with their challenges. (A) and (B) Carrying material through constrictive environments, (C) Passing-on and construction in constrictive environments, (D) Passing-on task.

Scaffold workers have three main sub-tasks: Carrying the material to the construction site, passing-on materials, and constructing the scaffold itself. Pictures of the workers performing these tasks can be found in appendix **??**.

A diagram showing the chronological order of these sub-tasks is shown in figure 13.

Carrying The carrying task is moving the scaffold materials from a drop-site, where a cart with steel scaffold materials is provided, to the construction site. Due to the complex constructions in industry, the drop-place and constructing place can often be far apart, involve obstacles such as stairs, rails, and low passages, and involve a lot of picking-up and laying down materials.

Passing-On The passing-on task is the near-vertical movement of scaffold materials, in which workers hand over the scaffolding materials upward or downward for the construction/deconstruction of the scaffold. Typically workers are about 2 meters vertically displaced. They would get handed the material at knee height, and pass them on above the shoulders.

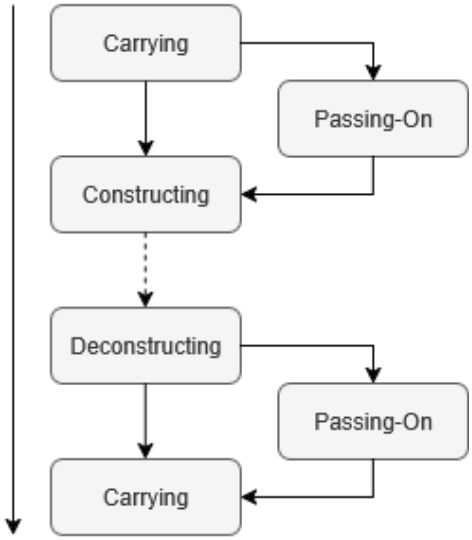


Fig. 13: Chronological sub-task order of the scaffolding task. Time between constructing and deconstructing one scaffold is typically a few days until a few weeks.

Constructing/Deconstructing The constructing and deconstructing of the scaffold itself is typically straight forward, but may also occur in difficult places, such as in a factory or around complex tubing. Within these places, workers have to move in awkward positions to assemble the scaffold.

Relative Frequency The personnel of the scaffolders are split up into two groups: Carriers and builders. The carriers typically bring all the scaffolding material from a drop-site to the construction site, and are not certified to climb scaffolds. As such, they also have minimal involvement in passing on materials, as they would only do that from the ground. The builders are more involved in passing on materials, and are the ones that actually construct the scaffold. The relative frequency of these groups for each sub-task is shown in table VI.

TABLE VI: Relative frequency of scaffolders during their respective responsibilities

Task / Employee	Carrier	Builder
Carrying	80-100%	20-40%
Passing-On	20-40%	40-60%
Constructing/ Deconstructing	0%	40-60%

2) Painting: Painting involves both the removal of build-up, such as stains, rust and residue, and the application of protective paint layer. Due to the heavy industry and seasalty conditions, factories get covered in hardy rust and residue. The painters maintain these by using different methods.

The removal process involves either manual or mechanical removal, or build-ups are removed using grit blasting. While painting, either hand painting or spray painting is done, depending on the geometry of the surface.

The workers have to work in a lot of awkward postures, due to the complexity of the factory.

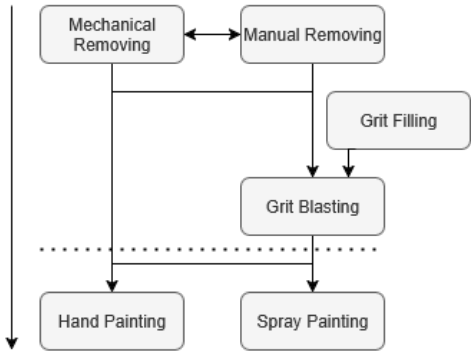


Fig. 14: Chronological sub-task order of the painting task. The hand and spray painting is done after all the removal tasks are finished.

Fig. 15: Mechanical tools used in the mechanical removing task. (Top-left) Small handheld mechanical chisels that vibrate. (Bottom-left) Handheld Mechanical sanding tool that rotates. (Right) Big handheld mechanical chisel.

Pictures of the workers performing these tasks can be found in appendix **??**, and a diagram showing these tasks in chronological order is shown in figure 14.

Manual Removing Manual removal involves the use of non-mechanical tools to remove build-up. This can range from using a hammer and chisel to shoveling.

Mechanical Removing Mechanical removal involves the use of mechanical tools to remove build-up. These can be mechanical chisels or mechanical sanding tool, as shown in figure 15. These tools are used for 'stubborn' build-ups. During the mechanical removal, the workers wear special breathers against the resulting debris clouds.

Grit Blasting Grit blasting, also known as abrasive blasting, involves tiny ceramic particles known as grits. By propelling these at high speeds through a hose, a surface can be sanded down. During this task, a special 'grit suit' is worn, which includes a breathing apparatus. Although this task is highly intensive, it is a relatively rare occurrence.

Hand/spray Painting Painting the surfaces involves simple paintbrushes and/or a small spray painters. Similarly to the removal, these tasks can result in awkward positions in order to paint hard-to-reach spots. However, there is no use of heavy mechanical tools.

Relative Frequency The personnel of the painters are split up into two groups: Painters and blasters. Painters and blasters

have very similar task distribution, with only the grit blasting portion being different. If grit blasting is necessary, the blasters will grit blast, and painters will act as the grit fillers. The relative frequency of these groups for each task is shown in table VII.

TABLE VII: Relative frequency of painters during their respective responsibilities

Task / Employee	Painter	Blaster
Manual removal	0-20%	0-20%
Mechanical removal	20-40%	20-40%
Grit blasting	0%	0-20%
Grit filling	0-20%	0%
Hand/spray painting	40-60%	40-60%

C. Results Questionnaire

A total of 10 workers, 5 scaffold builders and 5 painters, have been interviewed about their respective tasks.

4 out of the 5 builders mention that they have (had) lower back pain, and 3 out of the 5 have (had) knee pain (in the past).

The passing-on task is said to be the most intensive out of all the sub-tasks, and is a highly repetitive task that is sometimes done for hours on end.

During the carrying task, tight rooms and stairs are mentioned to be the most intensive parts.

As for the painters, during the painting tasks no complains are to be had. They mention that painting is the 'relaxing' part after the removal process is done.

Specifically the mechanical chisel task is said to be the most intensive long-term task. All 3 painters being interviewed for the mechanical removing process mentions a history of muscle problems in the upper body/limb parts as a result of this mechanical chisel.

Grit blasting is also intensive. The one person that was interviewed for the grit blasting mentioned having full-arm and shoulder pain after a session of grit blasting. Also grit filling is mentioned to be heavy, with suffering of the lower back. However, both of these are done about once or twice a month

D. Risk Assessment

The resulting intensity scores from the REBA in figure 16. These scores are then combined with the relative frequency of that task to find the final score. The final scores are then colored based on table I, and are shown in figure 1.

E. Discussion

Although the ERA gives a quick and fairly good overview of what tasks are intensive, there are still some aspects of the analysis that are lacking. First off, within REBA, dynamic elements are not accounted for. This skews the carrying task to be of a lower score in the REBA, 6, compared to the other two tasks, 14 and 13 (appendix **??**). Especially the mentioning of tight rooms and stairs, and the resulting knee problems as

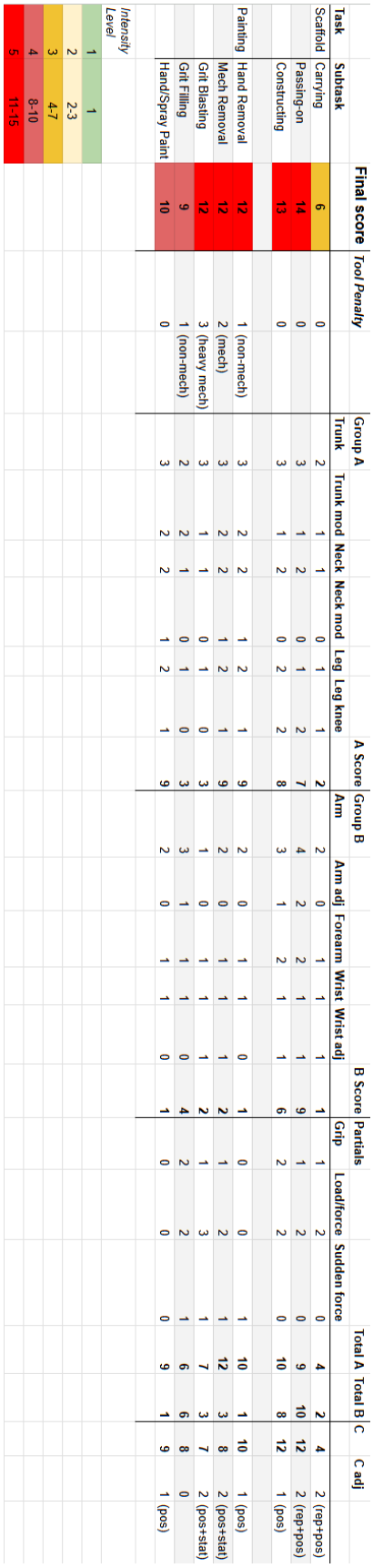


Fig. 16: Resulting intensity scores (range 1-15) using the REBA method [[12]]. The resulting scores translate into an intensity level (range 1-5)

a result, in the interviews shows that the most intensive part of the task is not included within the REBA.

This would only count for the carrying though, and using other methods which focuses on dynamics, such as Occupational Repetitive Actions [45], can hamper the effectiveness of assessing the other tasks.

Secondly, tasks that are similar in body postures, but differ in intensity of tools also are seemingly similar. Mechanical removal, score 12, is seen as the most intensive, while hand/spray painting, score 10, is seen as a leisure activity. Of course, the postures they would need to adopt in hand/spray painting may still be harmful, and the feeling of it being a 'leisure' activity is likely a matter of relative experience .

And finally, the use of relative frequency may skew results based on the amount of tasks one has. Both painting groups each have 4 sub-tasks, while the scaffolding groups have 2 and 3 sub-tasks. Therefore, the relative frequency of the scaffolders tasks are on average higher, and as such resulted in 3 of the 5 being red. This may underestimate the impact of, for example, the combination of removal and painting with their respective bad postures.

F. OpenSim Tools

1) Inverse Dynamics: Inverse Dynamics is the process of taking kinematics and kinetics, and from those calculating net joint moments and joint reaction forces. This is done using the Newton-Euler equations of motion, which calculate linear forces (Newton) and moments (Euler), but applies it recursive inversely. The ID tool within OpenSim uses the following steps to achieve the required generalized forces [46]:

$$\tau = M(q)\ddot{q} + C(q, \dot{q}) + G(q) \tag{2}$$

In which the $q, \dot{q}, \ddot{q}$ are the vectors of generalized positions, velocities, and accelerations, respectively. These are applied to the system mass matrix $M(q)$, which is the Newton part ($F = ma$). The vector of Coriolis and centrifugal forces $C(q, \dot{q})$ is the Euler part, and then the vector of gravitational forces $G(q)$ is finally applied. The ID tool uses these known variables to solve equations of motion for the unknown generalized forces τ.

To use the ID tool, it requires a kinematic file of the subjects captured motion, and an external force file (EFF). The former will be provided by OpenCap, and the latter by the dynamics module.

2) Static Optimization: The SO tool serves as an extension to the ID tool, that uses the resulting accelerations from the ID solution as a constraint [46]. Similarly to the ID tool, it calculates the unknown generalized forces τ, but in addition it solves muscles forces by calculating muscle activations. This is done due to the redundant nature of muscles, in which an analytical solution is not possible. This is either done by subjecting τ to one of the following muscle activation-to-force conditions: Ideal force generators:

$$\tau_j = \sum_{m=1}^{n} (a_m F_m^0) r_{m,j} \tag{3}$$

Where τ_j is the generalized force acting about the j^{th} joint axis, n is the number of muscles in the model, a_m is the activation level of muscle m at a discrete time step, F_m^0 is its maximum isometric force, and $r_{m,j}$ is its moment arm about the j^{th} joint axis

The second condition is constrained by the force-length-velocity properties of the muscles:

$$\tau_j = \sum_{m=1}^{n} [a_m f(F_m^0, l_m, v_m)] r_{m,j} \tag{4}$$

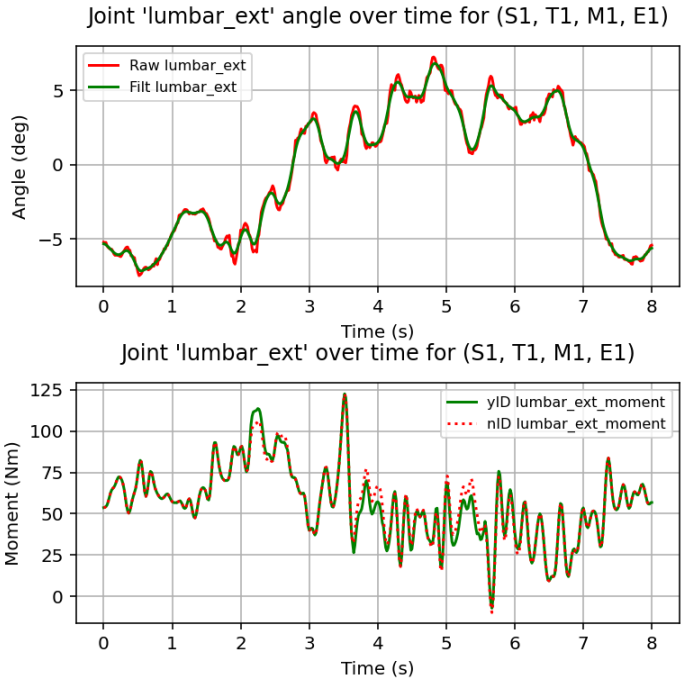
Where the force-length-velocity surface f is applied by F_m^0, the muscles length l_m, and the muscles shortening velocity v_m while minimizing the objective function:

$$J = \sum_{m=1}^{n} (a_m)^p \tag{5}$$

Where p is a user defined constant.

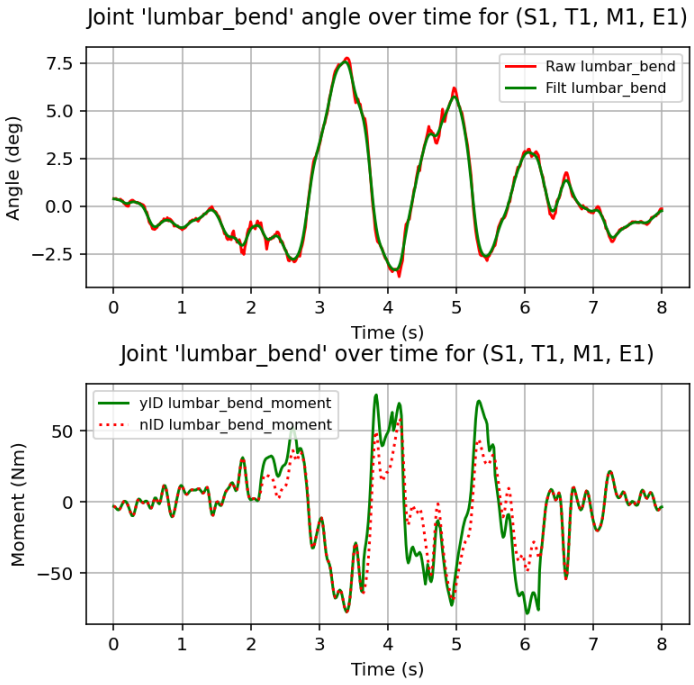
G. Resulting joint angles and moments from IK and ID

All results for the relevant joints are shown in Figures 17-34

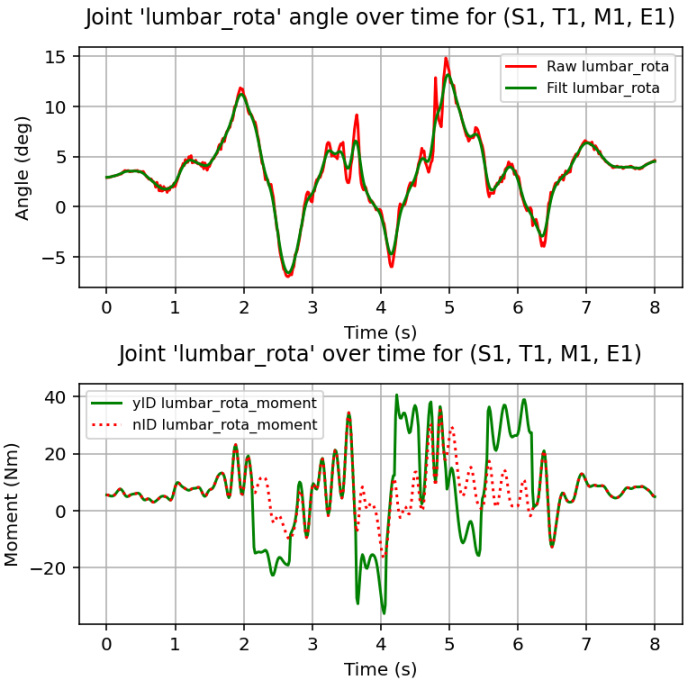


Joint Angle and Moments results for Subject 1, Trial 1

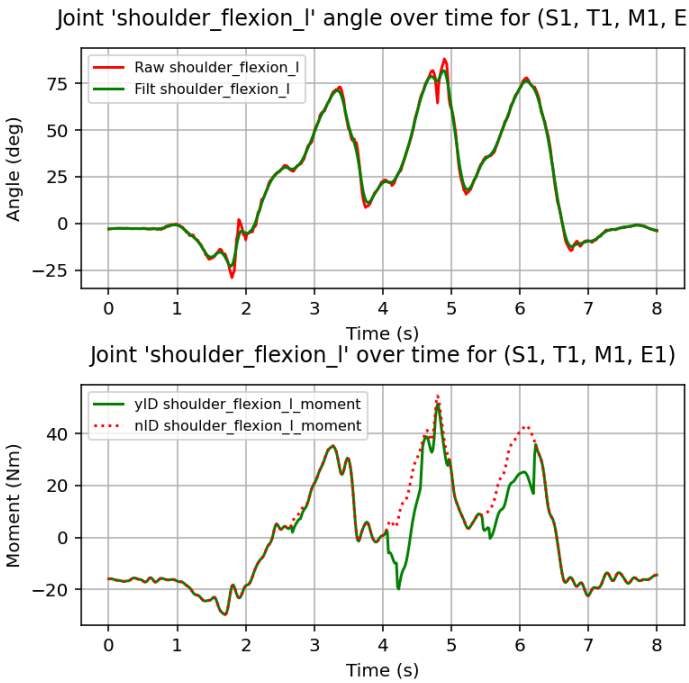
17.A: Lumbar extension.



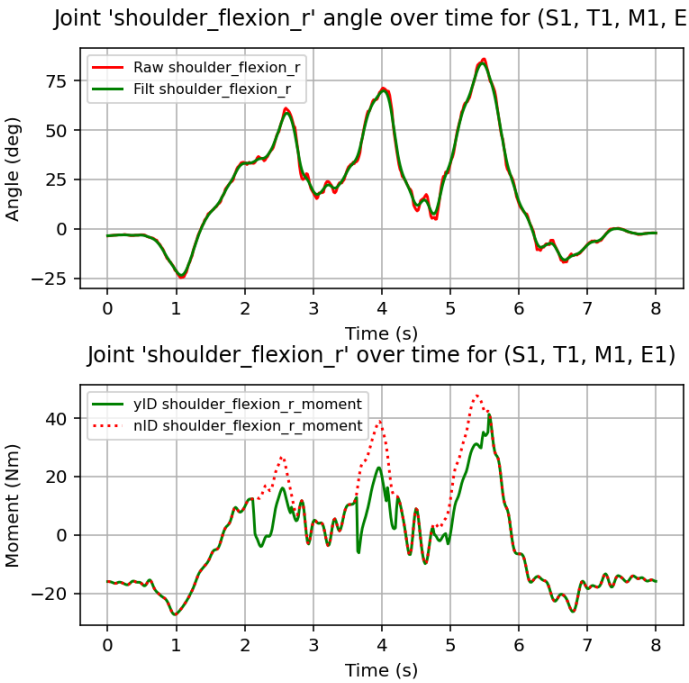
17.B: Lumbar bending.



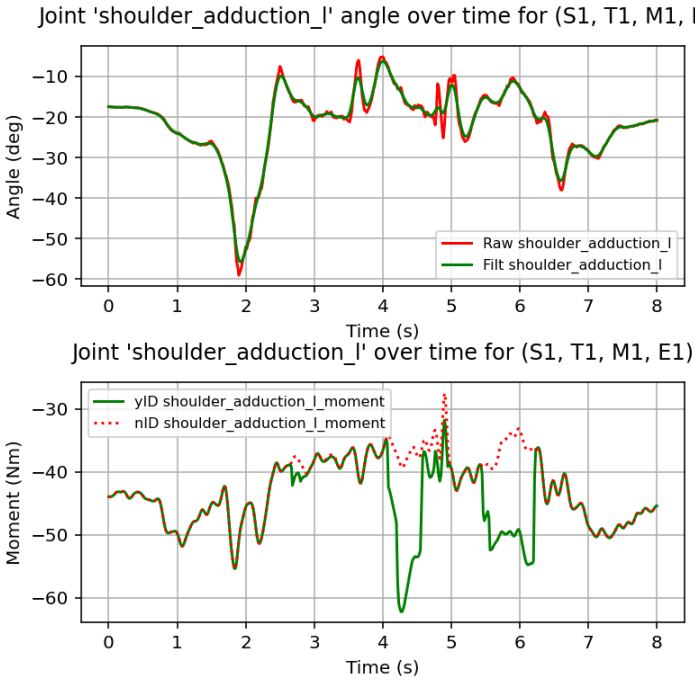
17.C: Lumbar rotation.



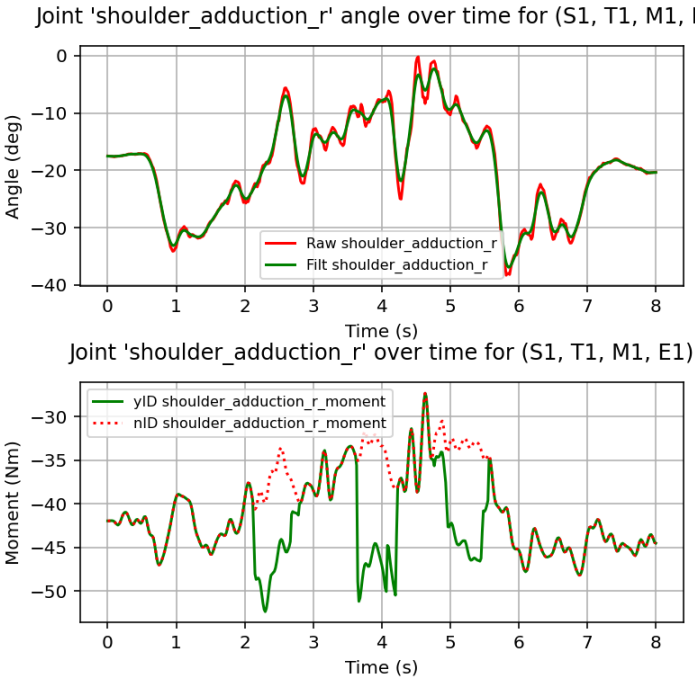
17.D: Shoulder flexion-extension of left arm.



17.E: Shoulder flexion-extension of right arm.

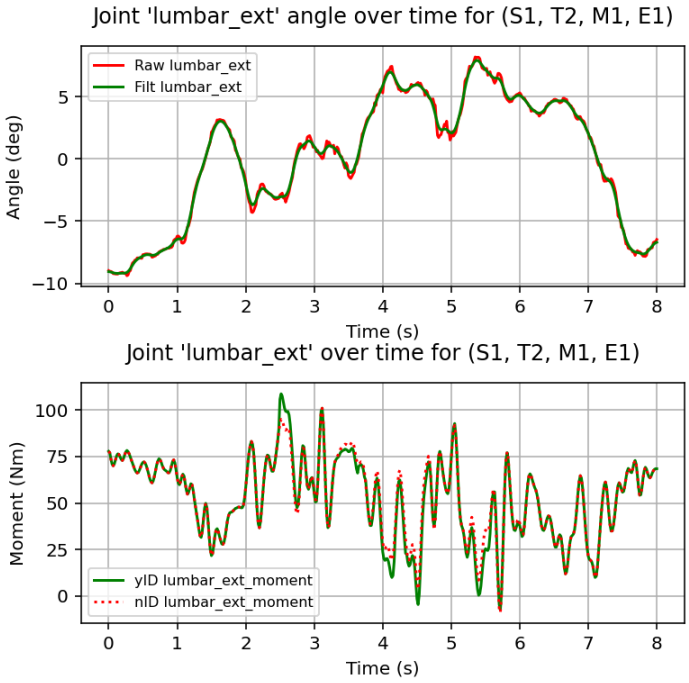


17.F: Shoulder adduction-abduction of left arm.



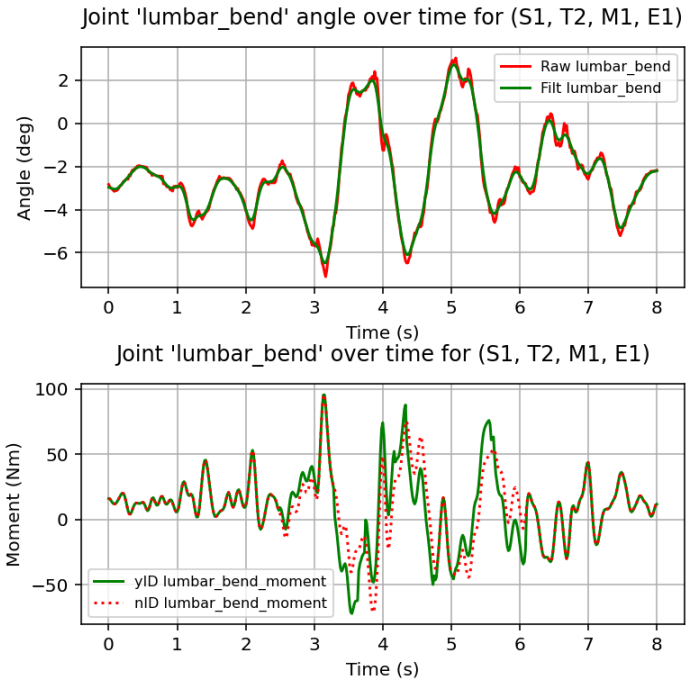
17.G: Shoulder adduction-abduction of right arm.

Fig. 17: Results from Inverse Dynamics for Subject 1, Trial 1. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

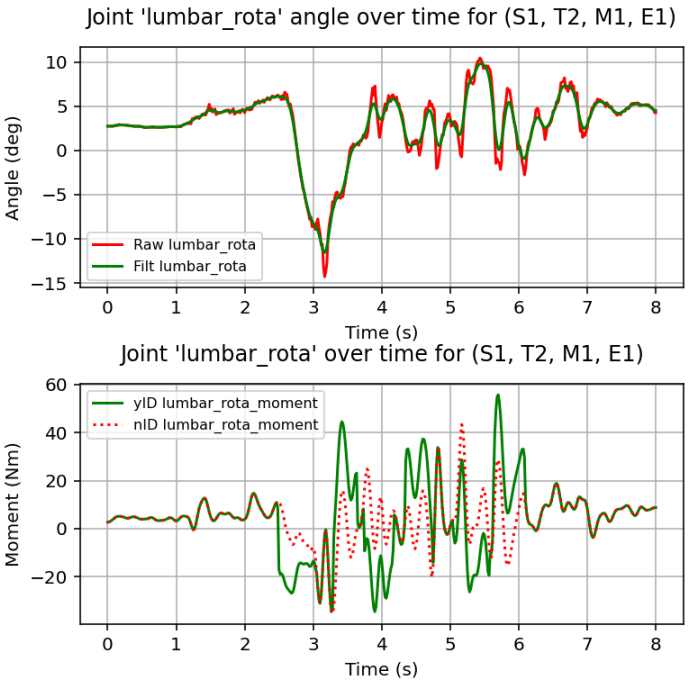


Joint Angle and Moments results for Subject 1, Trial 2

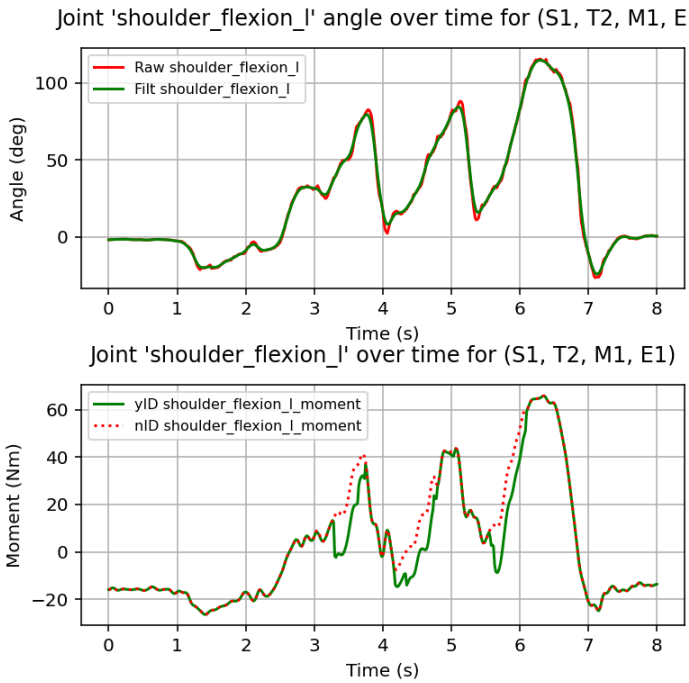
18.A: Lumbar extension.



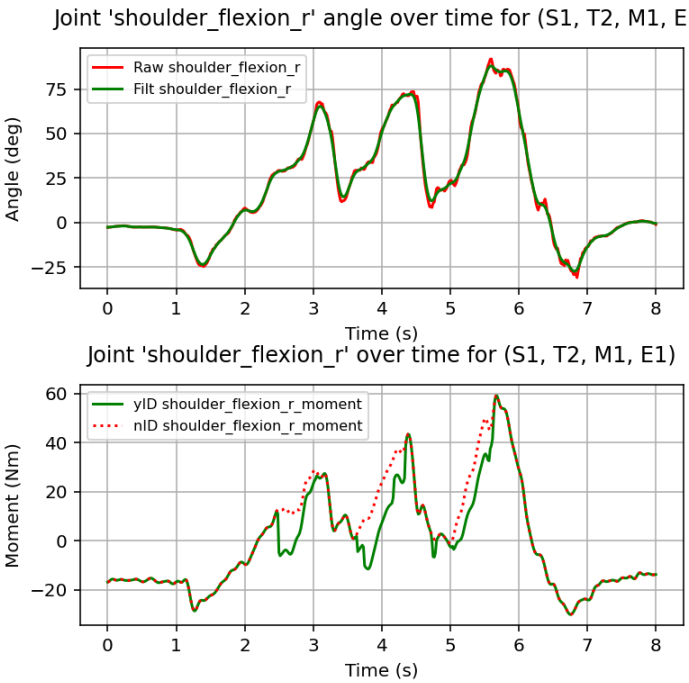
18.B: Lumbar bending.



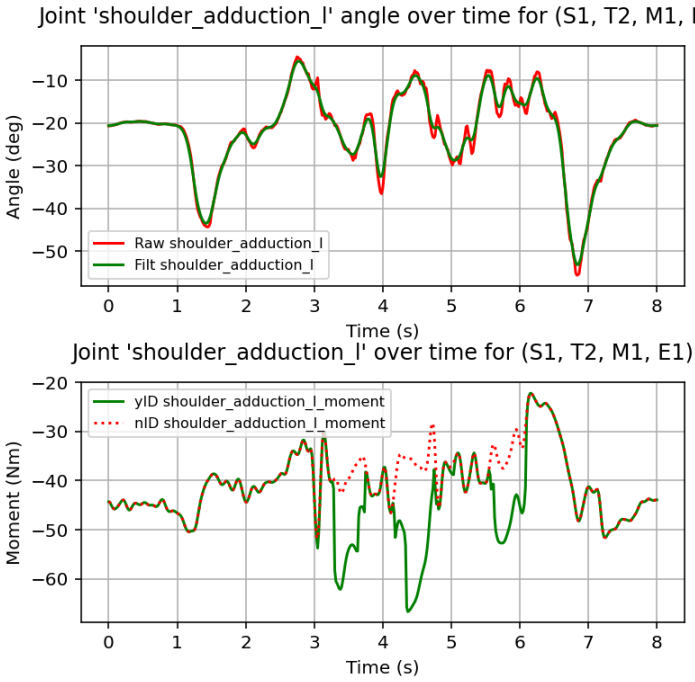
18.C: Lumbar rotation.



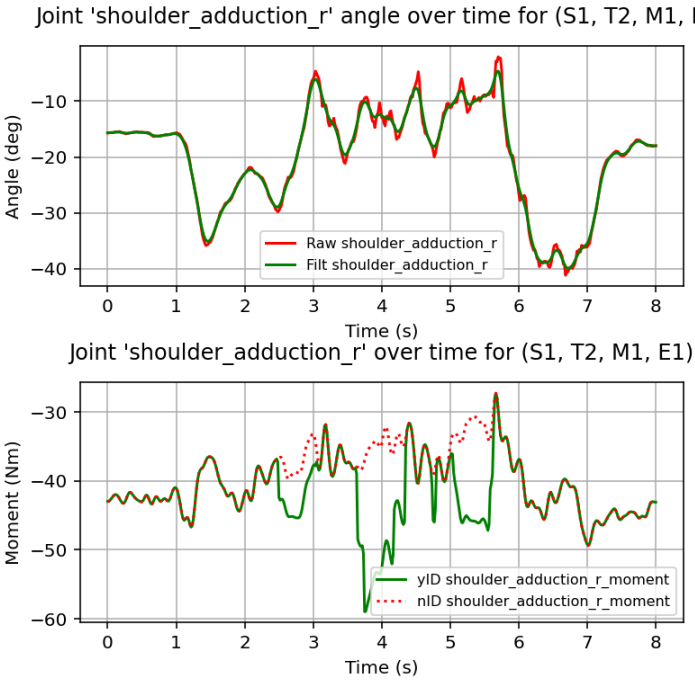
18.D: Shoulder flexion-extension of left arm.



18.E: Shoulder flexion-extension of right arm.

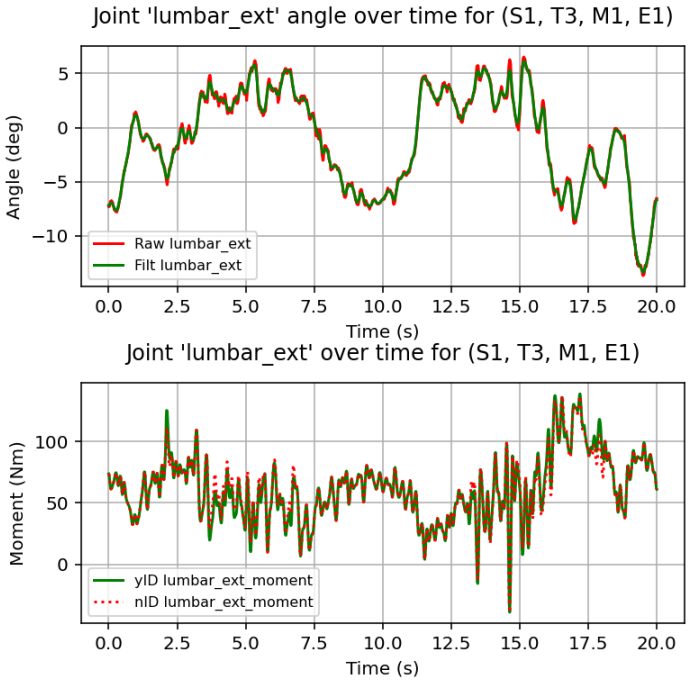


18.F: Shoulder adduction-abduction of left arm.



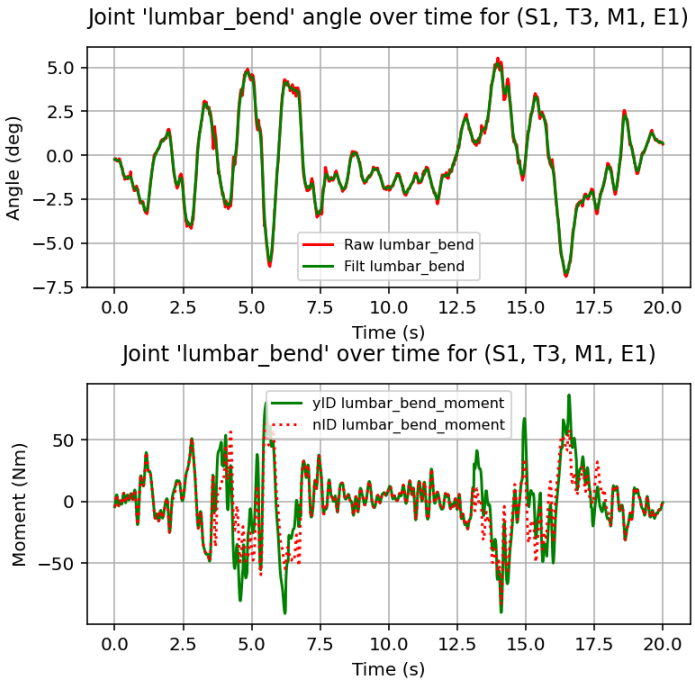
18.G: Shoulder adduction-abduction of right arm.

Fig. 18: Results from Inverse Dynamics for Subject 1, Trial 2. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

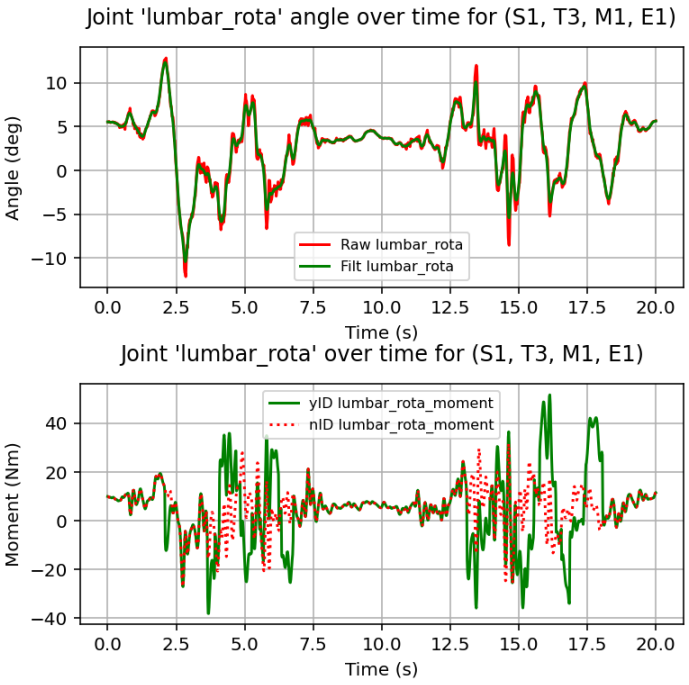


Joint Angle and Moments results for Subject 1, Trial 3

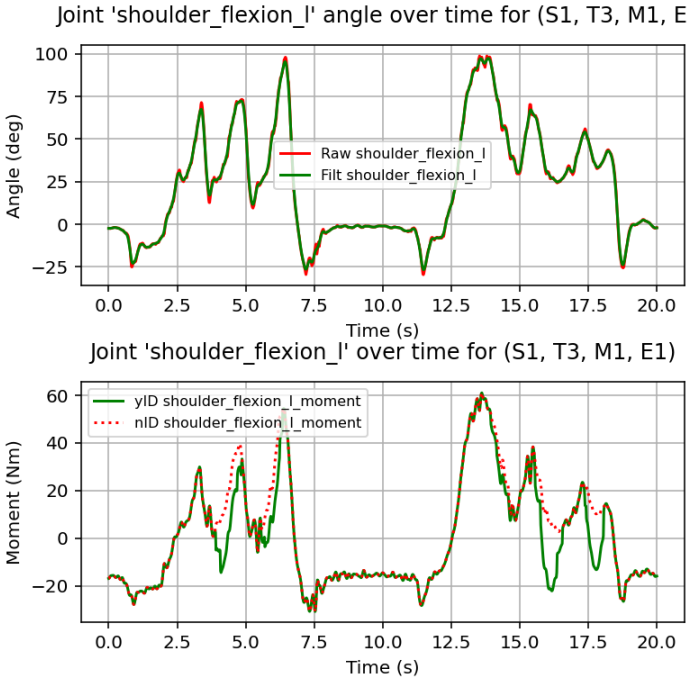
19.A: Lumbar extension.



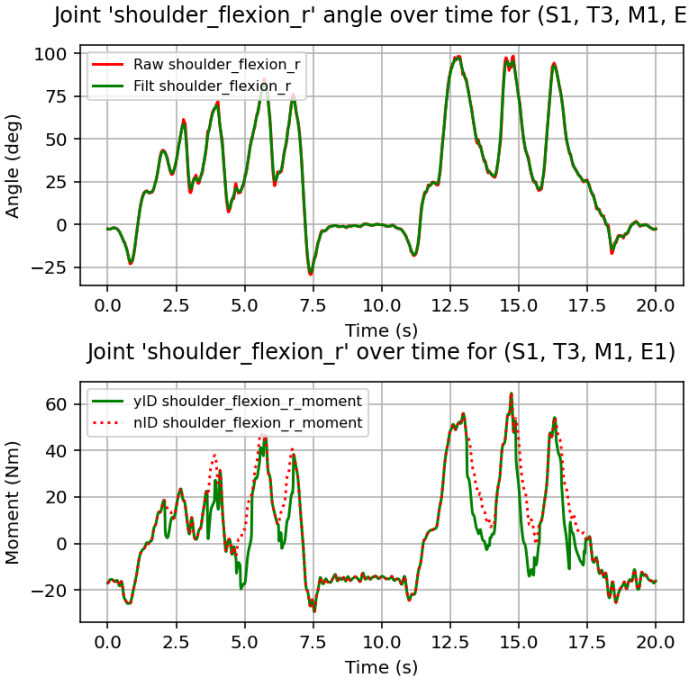
19.B: Lumbar bending.



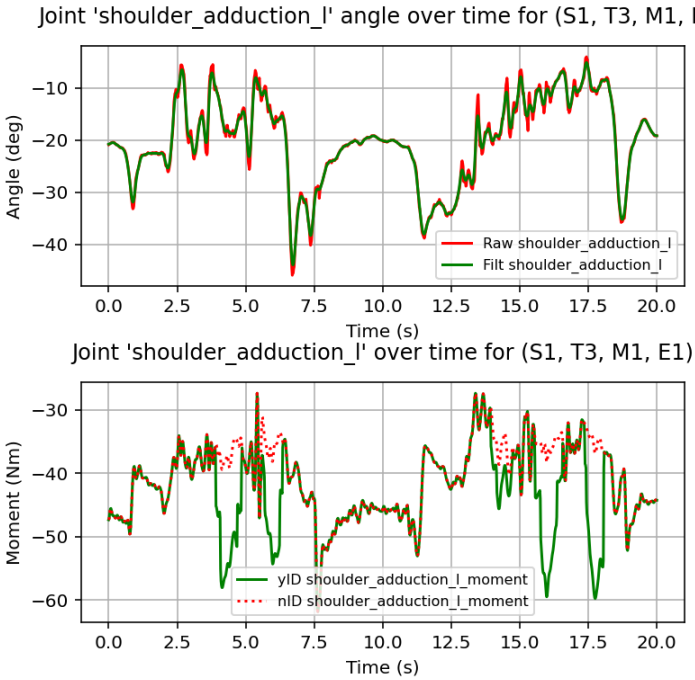
19.C: Lumbar rotation.



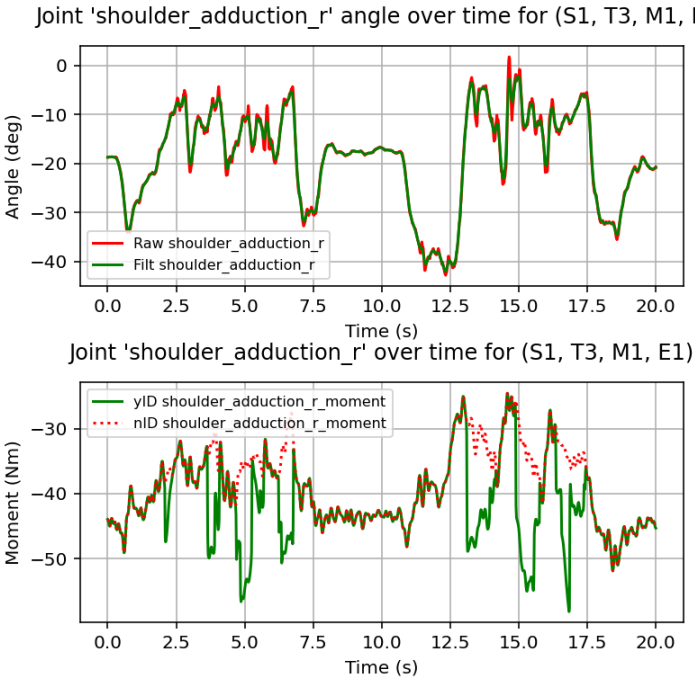
19.D: Shoulder flexion-extension of left arm.



19.E: Shoulder flexion-extension of right arm.

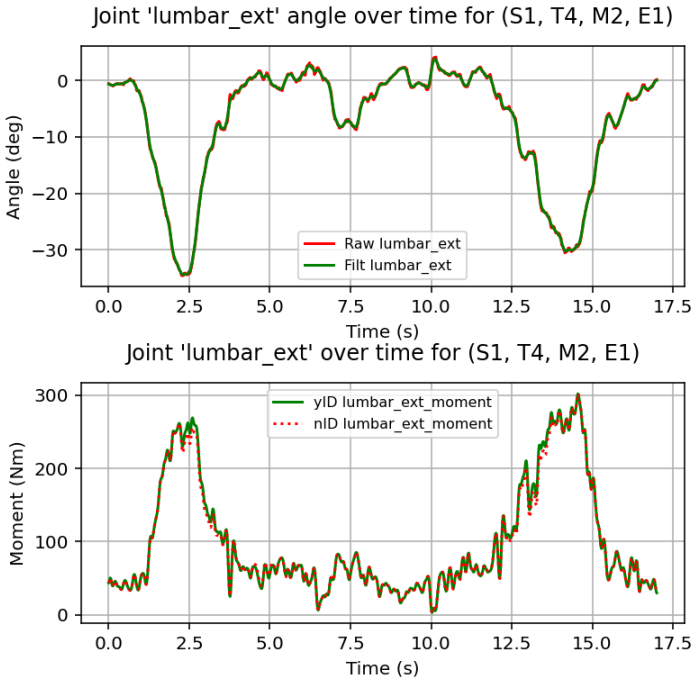


19.F: Shoulder adduction-abduction of left arm.



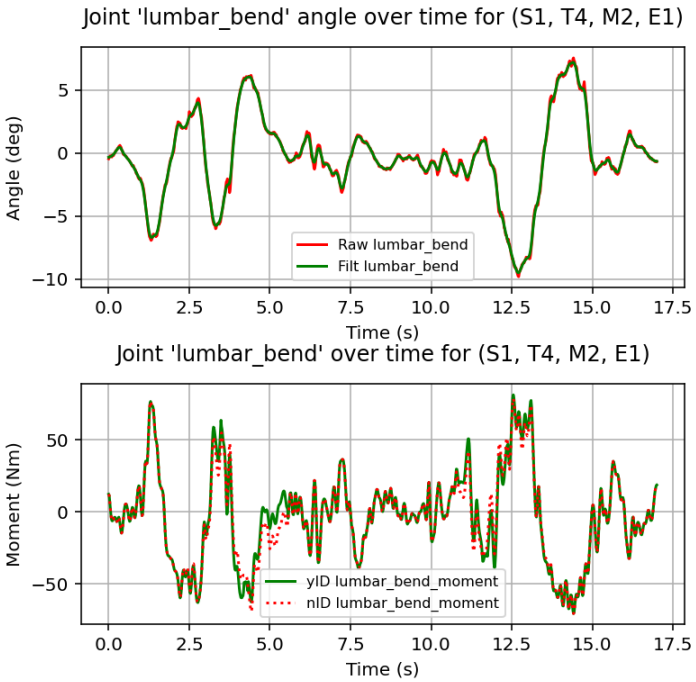
19.G: Shoulder adduction-abduction of right arm.

Fig. 19: Results from Inverse Dynamics for Subject 1, Trial 3. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

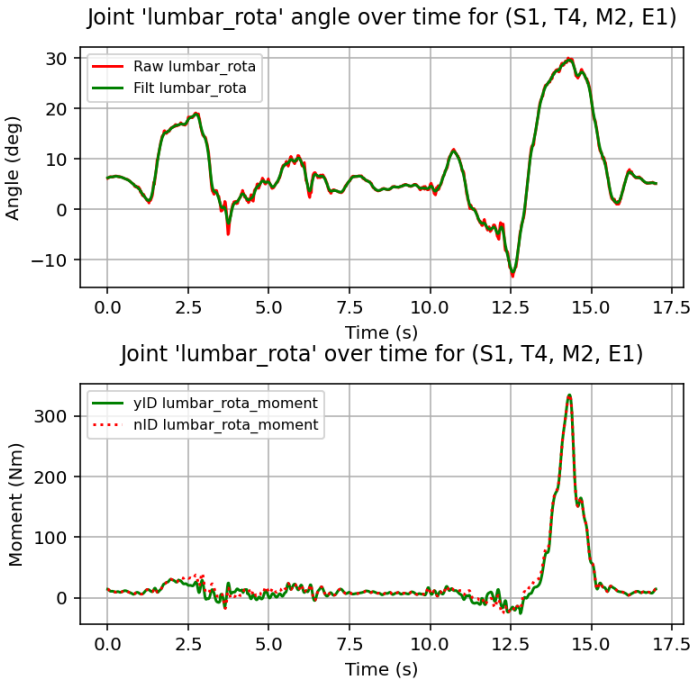


Joint Angle and Moments results for Subject 1, Trial 4

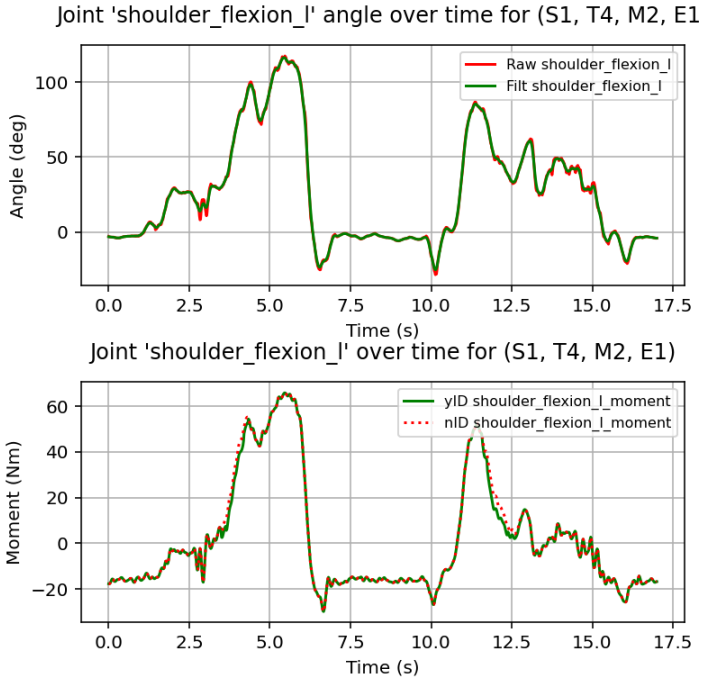
20.A: Lumbar extension.



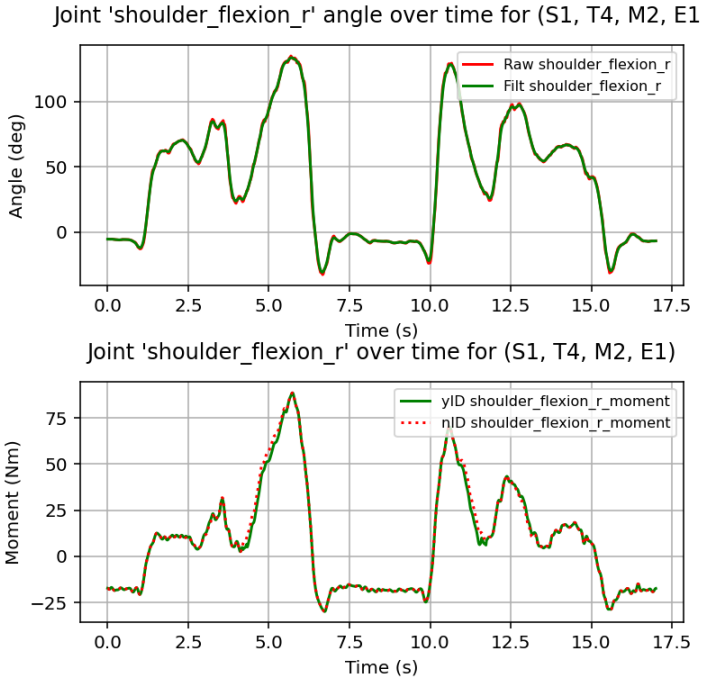
20.B: Lumbar bending.



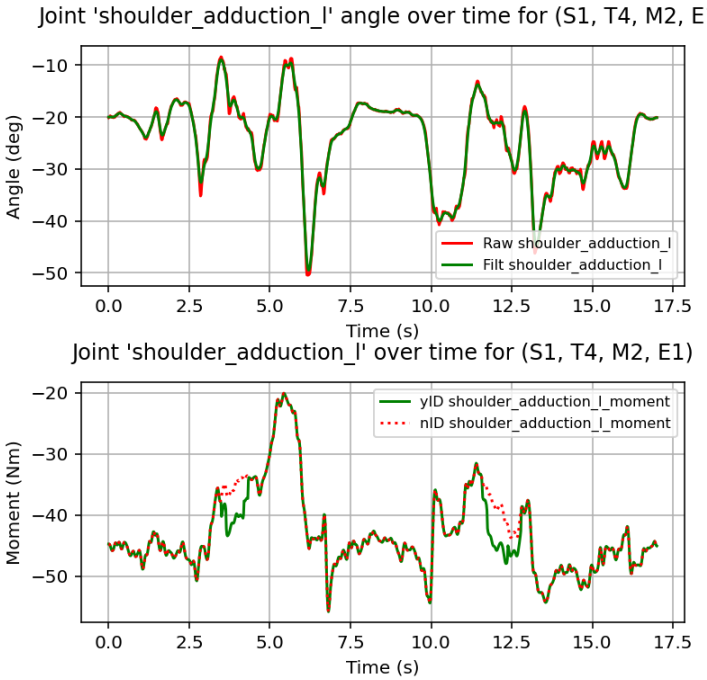
20.C: Lumbar rotation.



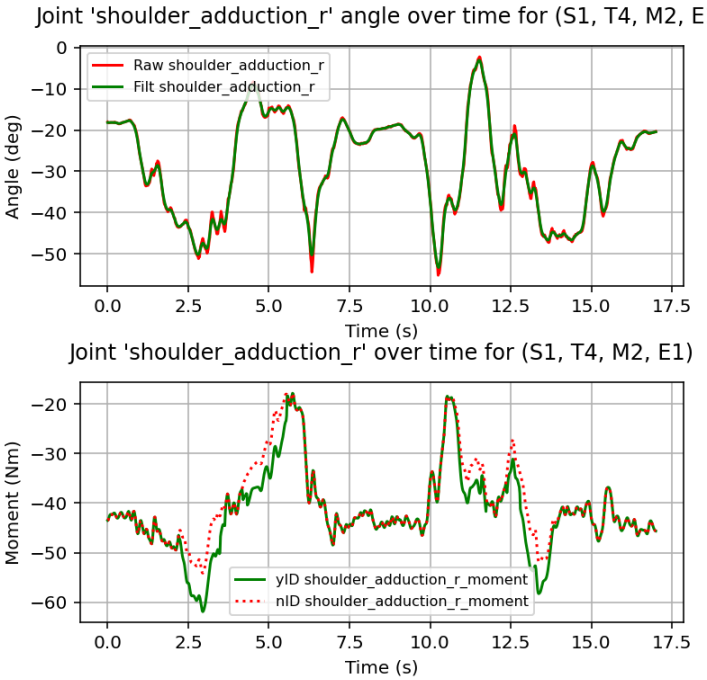
20.D: Shoulder flexion-extension of left arm.



20.E: Shoulder flexion-extension of right arm.

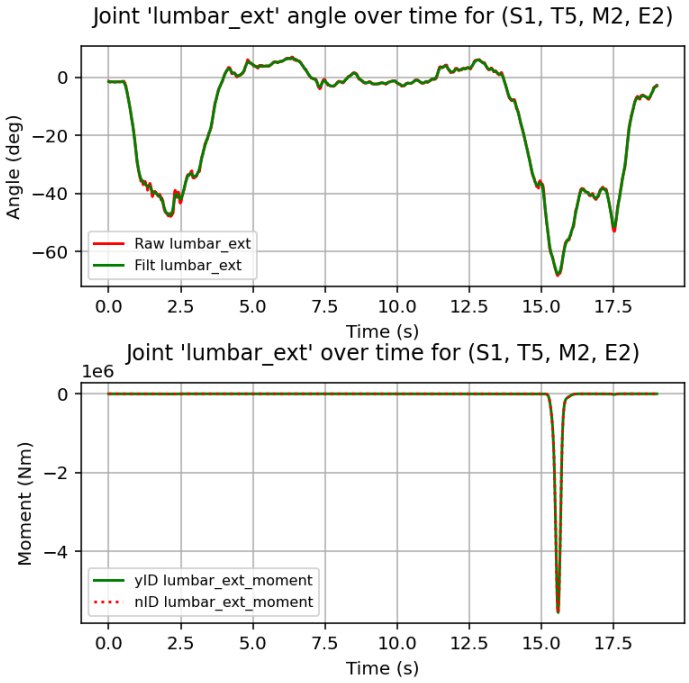


20.F: Shoulder adduction-abduction of left arm.



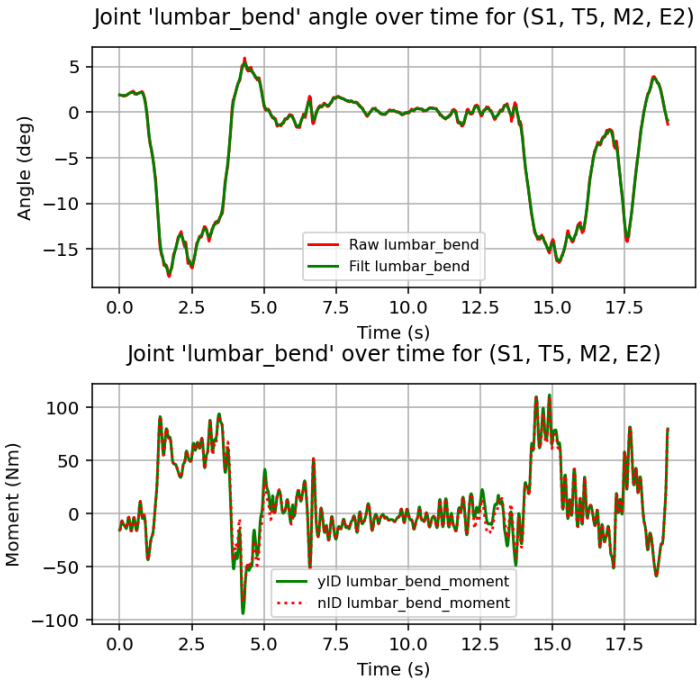
20.G: Shoulder adduction-abduction of right arm.

Fig. 20: Results from Inverse Dynamics for Subject 1, Trial 4. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

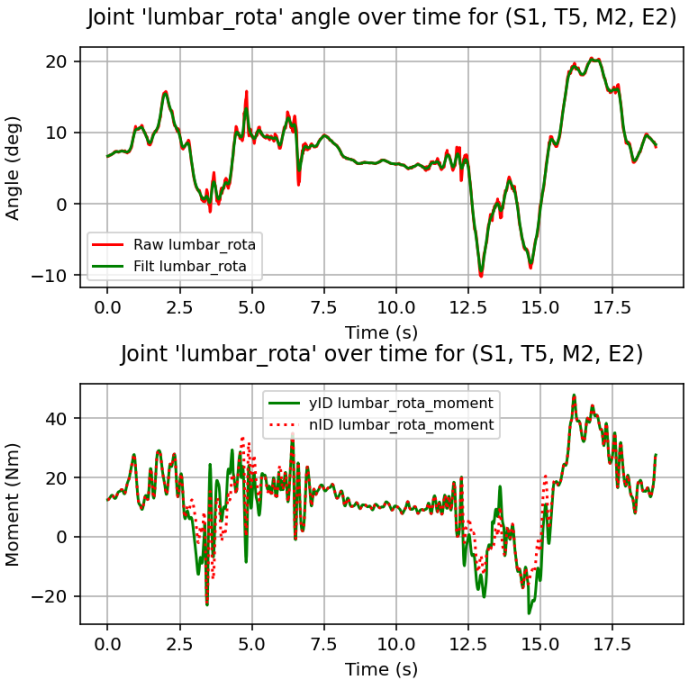


Joint Angle and Moments results for Subject 1, Trial 5

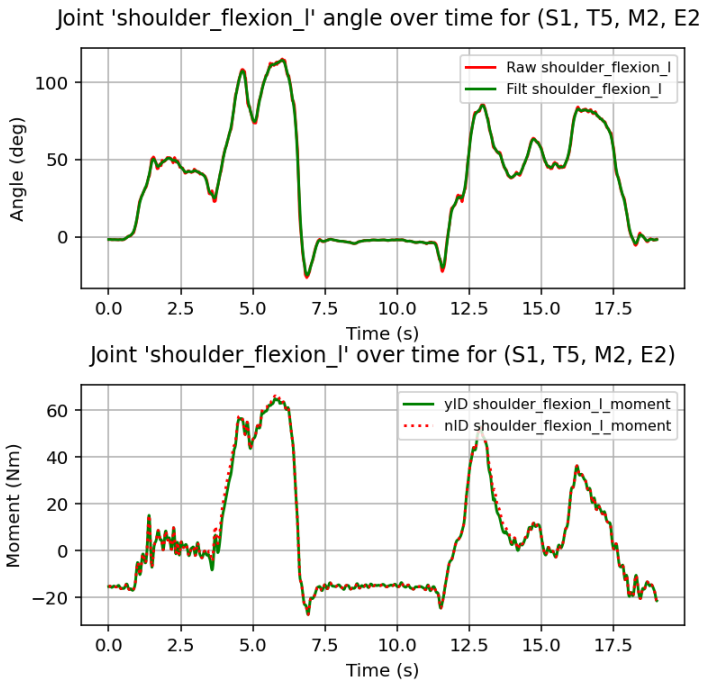
21.A: Lumbar extension.



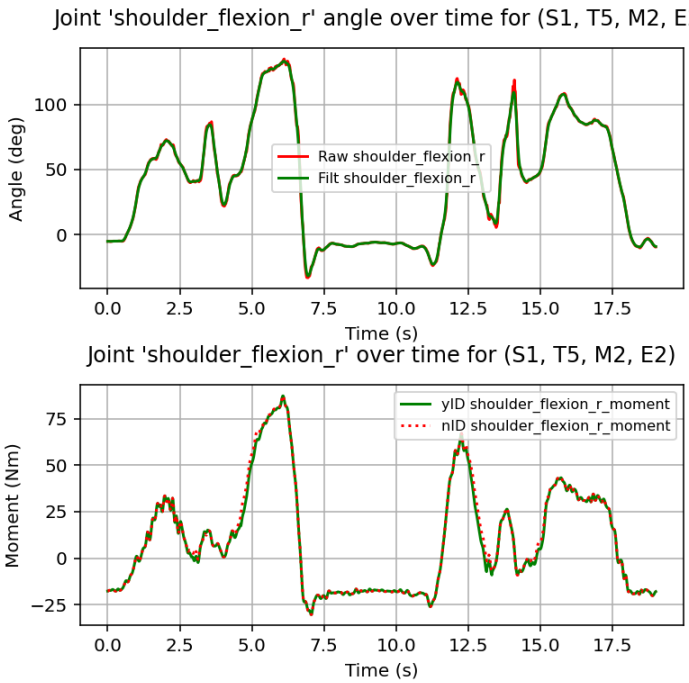
21.B: Lumbar bending.



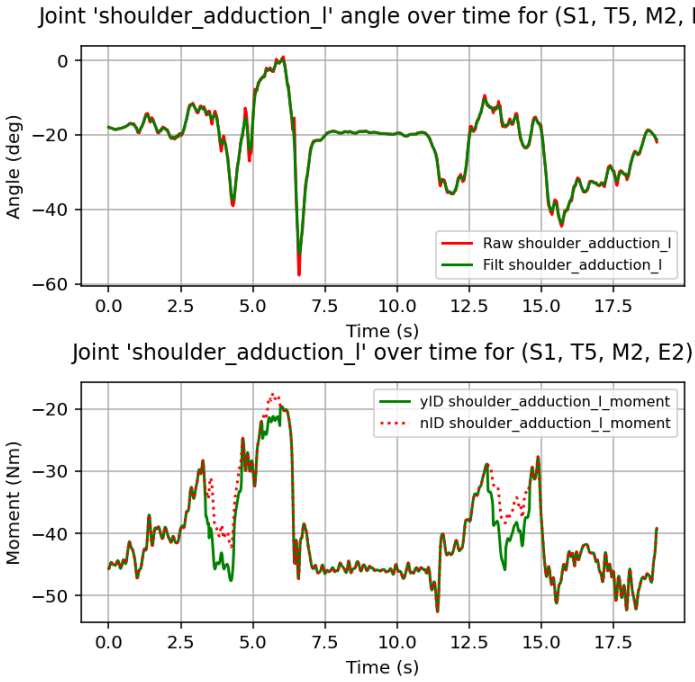
21.C: Lumbar rotation.



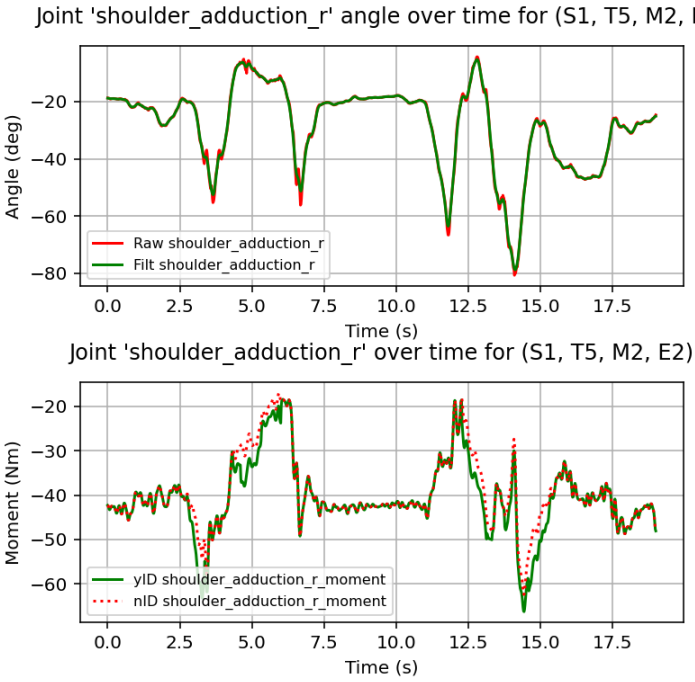
21.D: Shoulder flexion-extension of left arm.



21.E: Shoulder flexion-extension of right arm.

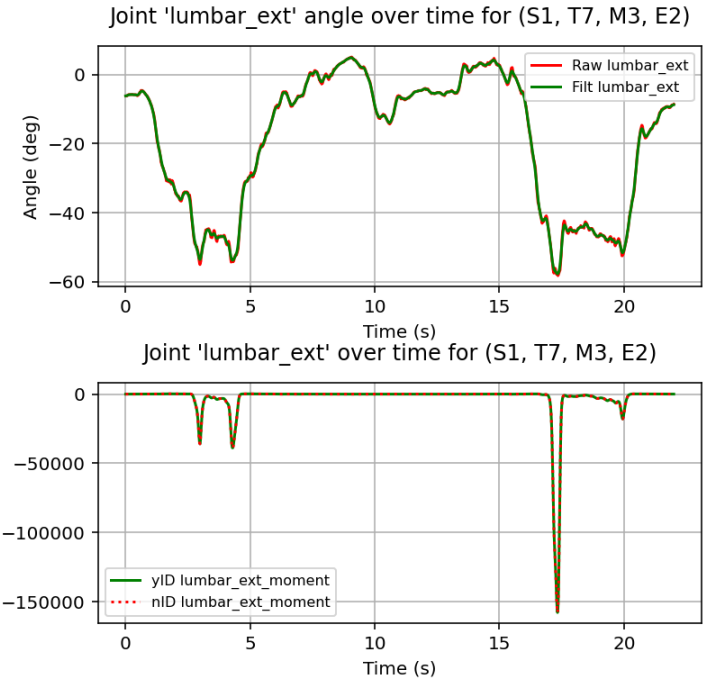


21.F: Shoulder adduction-abduction of left arm.



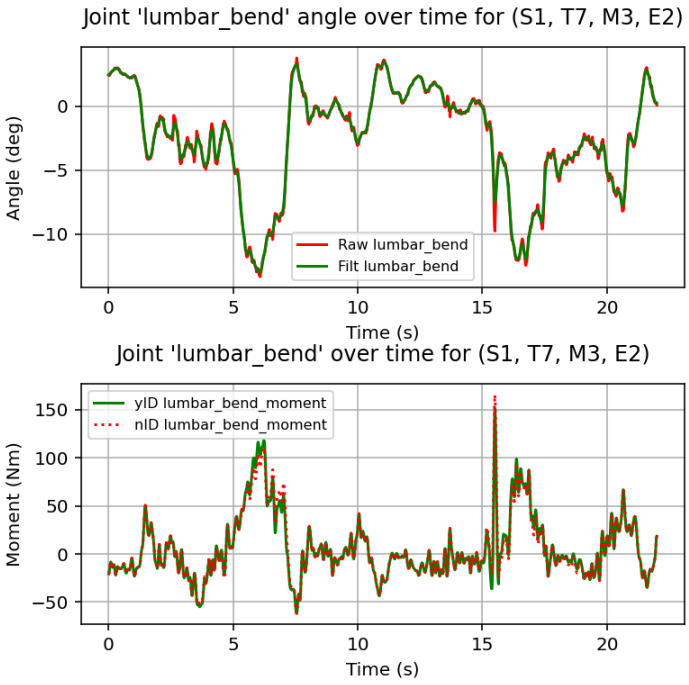
21.G: Shoulder adduction-abduction of right arm.

Fig. 21: Results from Inverse Dynamics for Subject 1, Trial 5. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

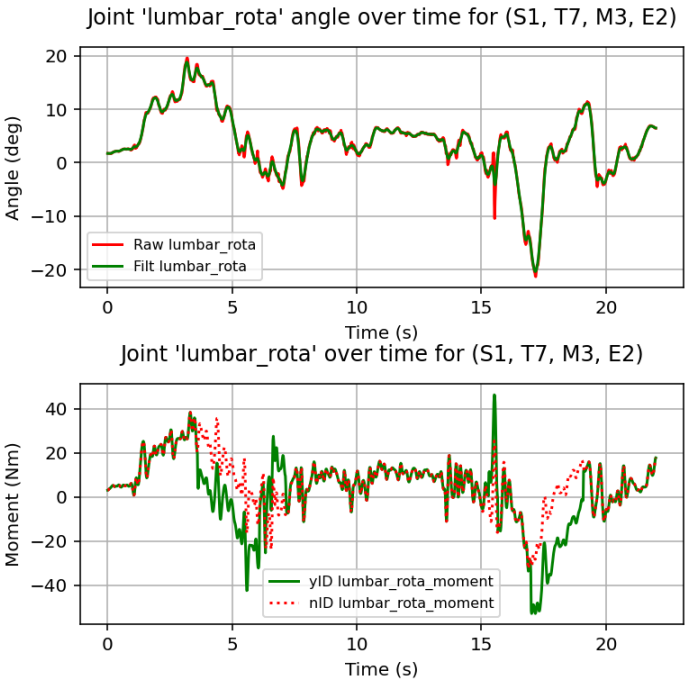


Joint Angle and Moments results for Subject 1, Trial 7

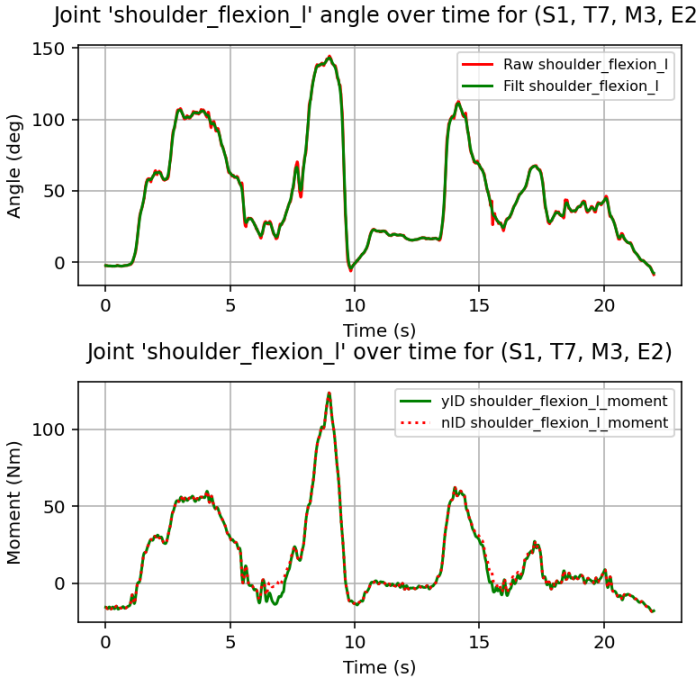
22.A: Lumbar extension.



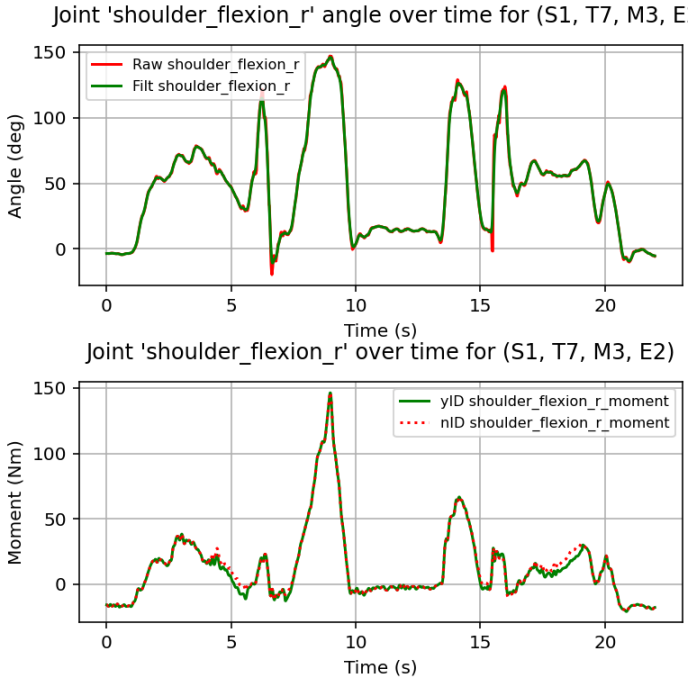
22.B: Lumbar bending.



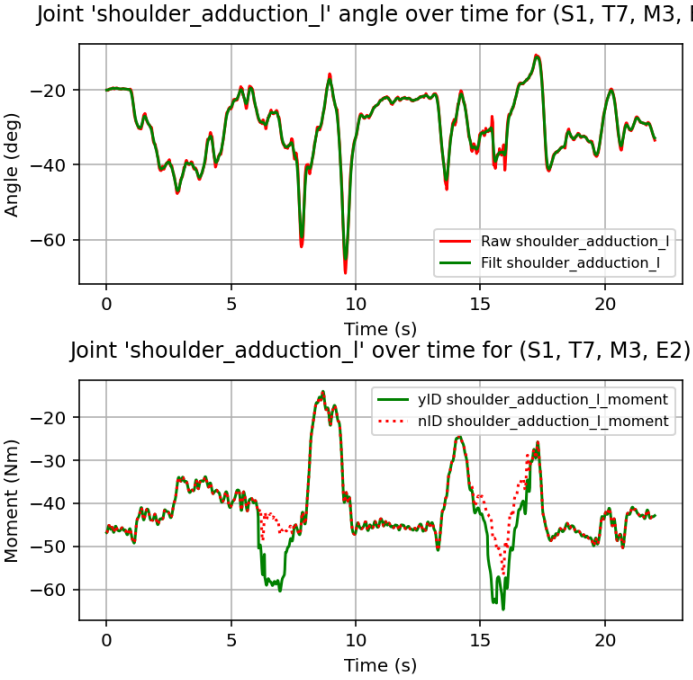
22.C: Lumbar rotation.



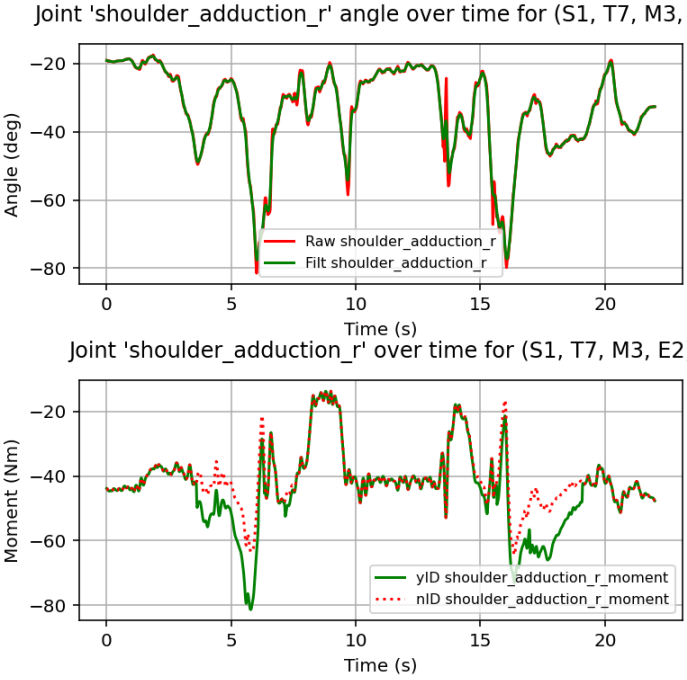
22.D: Shoulder flexion-extension of left arm.



22.E: Shoulder flexion-extension of right arm.

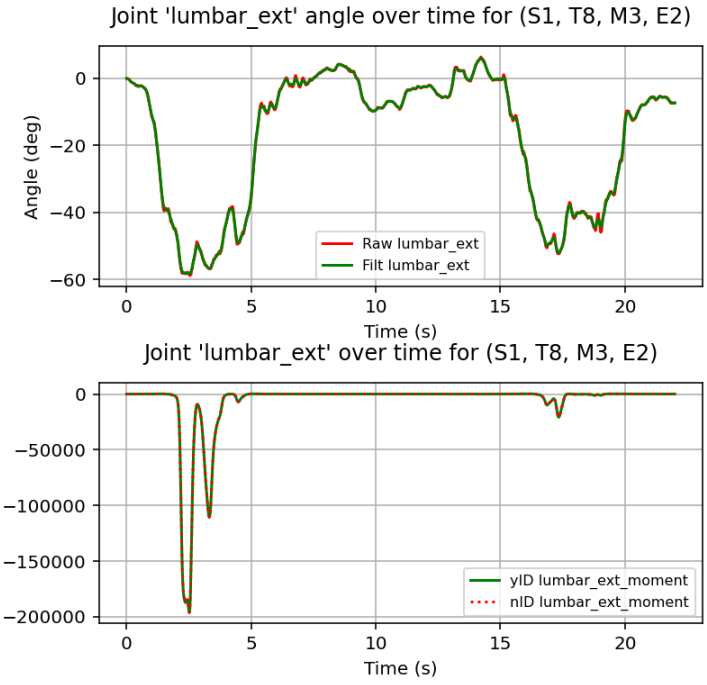


22.F: Shoulder adduction-abduction of left arm.



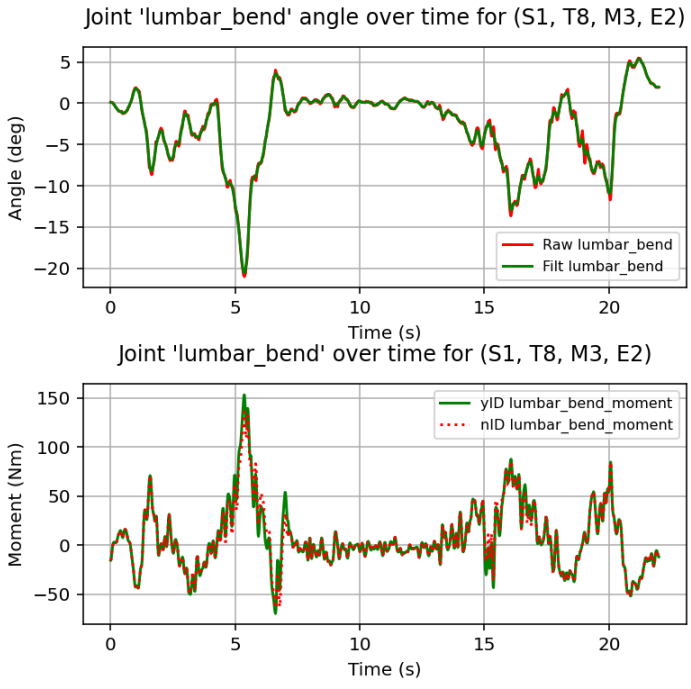
22.G: Shoulder adduction-abduction of right arm.

Fig. 22: Results from Inverse Dynamics for Subject 1, Trial 7. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

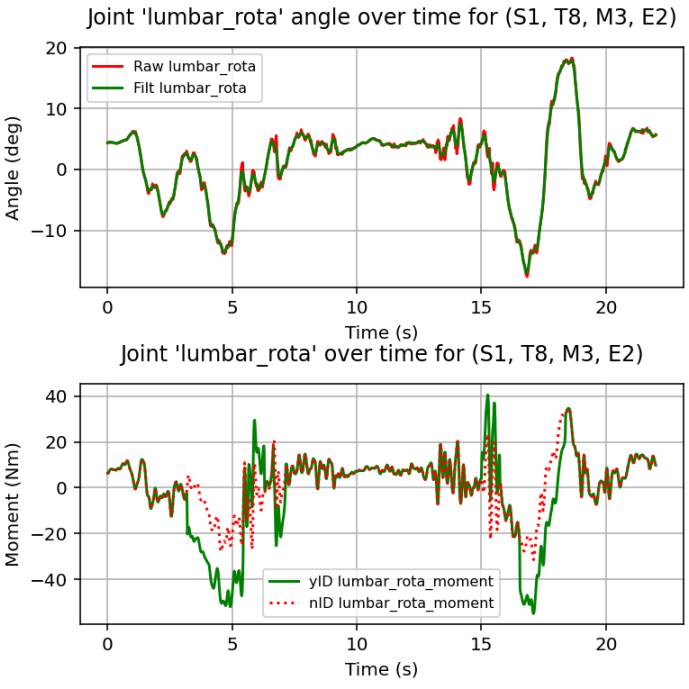


Joint Angle and Moments results for Subject 1, Trial 8

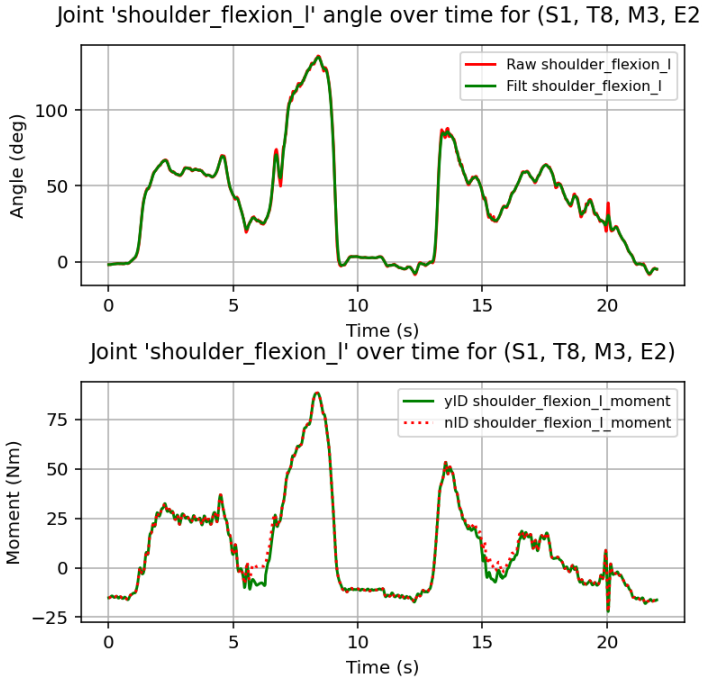
23.A: Lumbar extension.



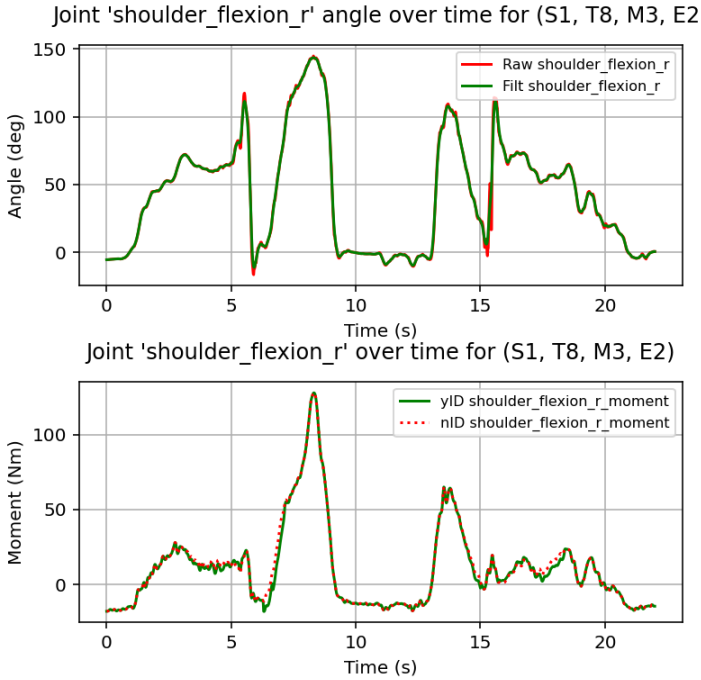
23.B: Lumbar bending.



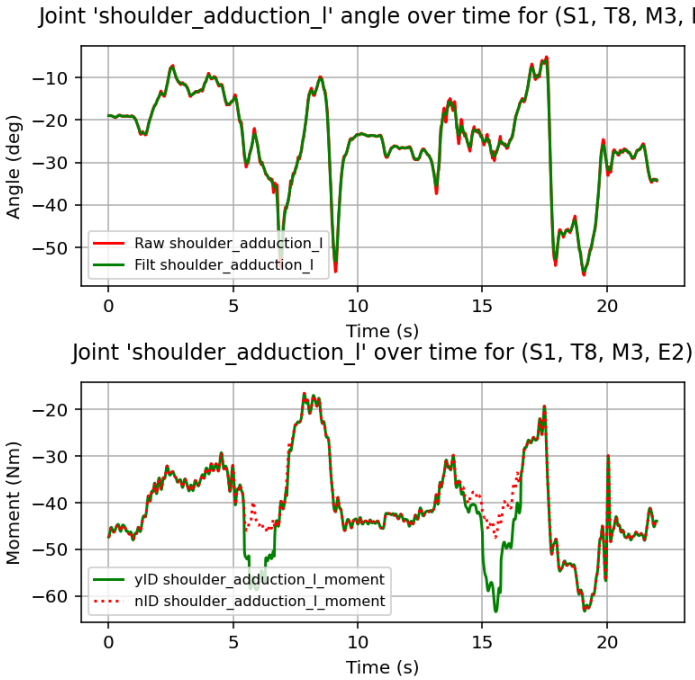
23.C: Lumbar rotation.



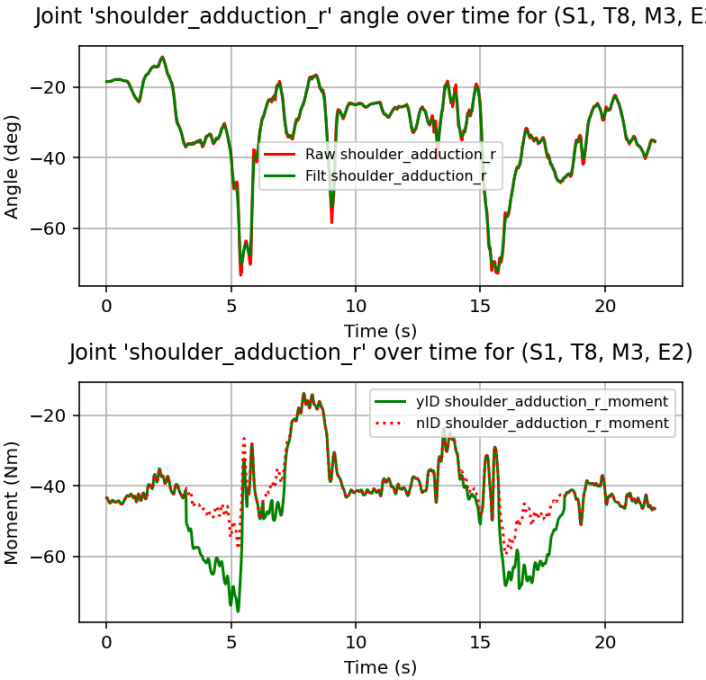
23.D: Shoulder flexion-extension of left arm.



23.E: Shoulder flexion-extension of right arm.

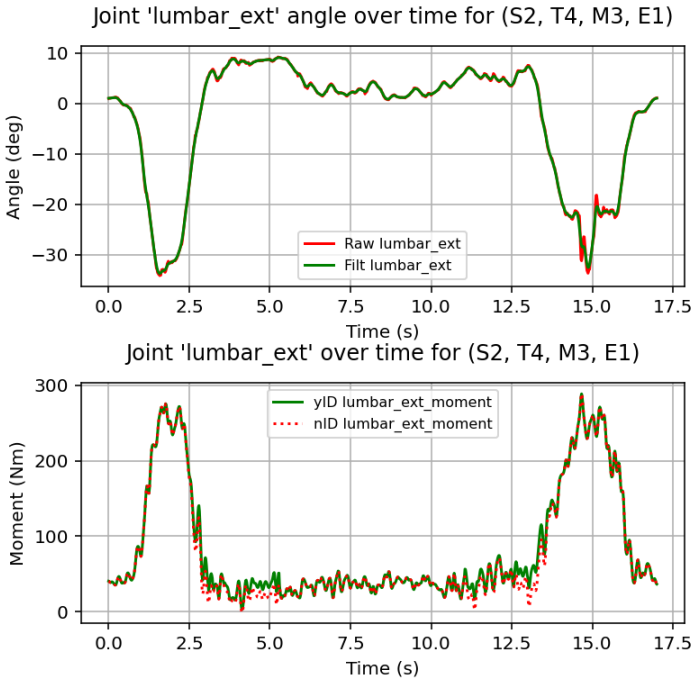


23.F: Shoulder adduction-abduction of left arm.



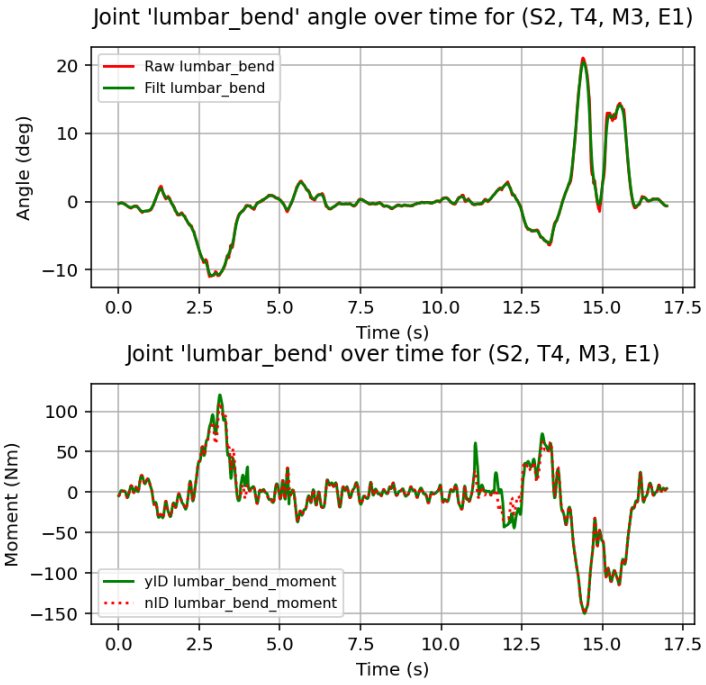
23.G: Shoulder adduction-abduction of right arm.

Fig. 23: Results from Inverse Dynamics for Subject 1, Trial 8. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

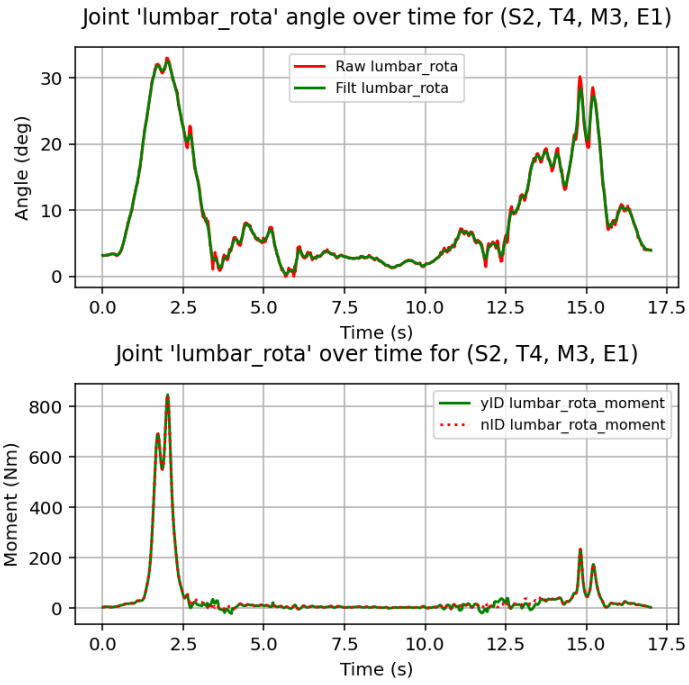


Joint Angle and Moments results for Subject 2, Trial 4

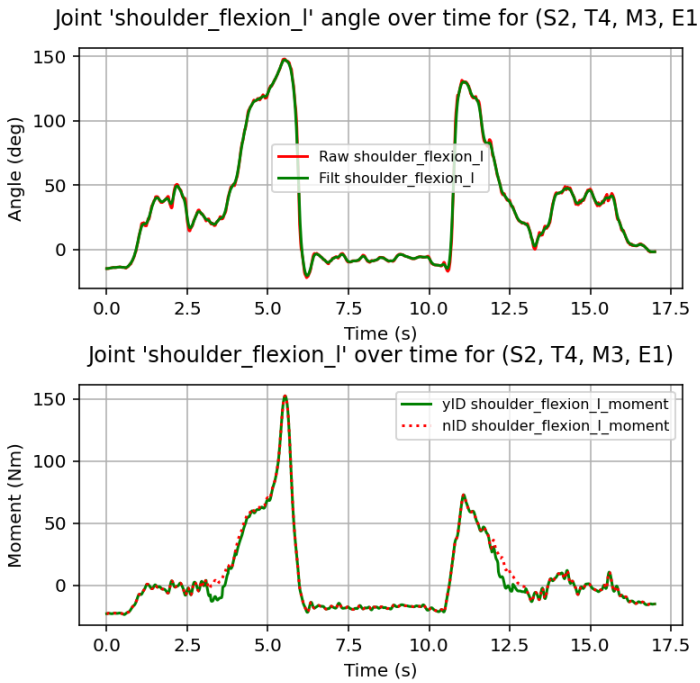
24.A: Lumbar extension.



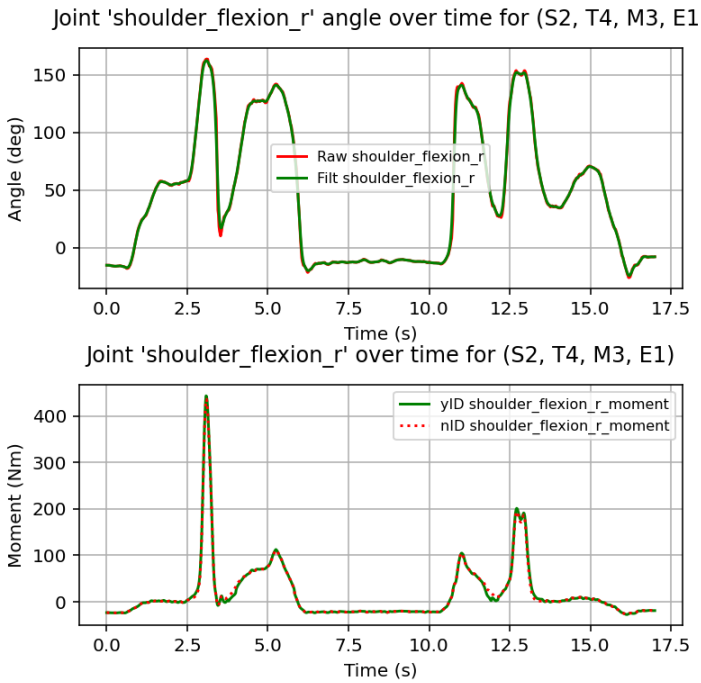
24.B: Lumbar bending.



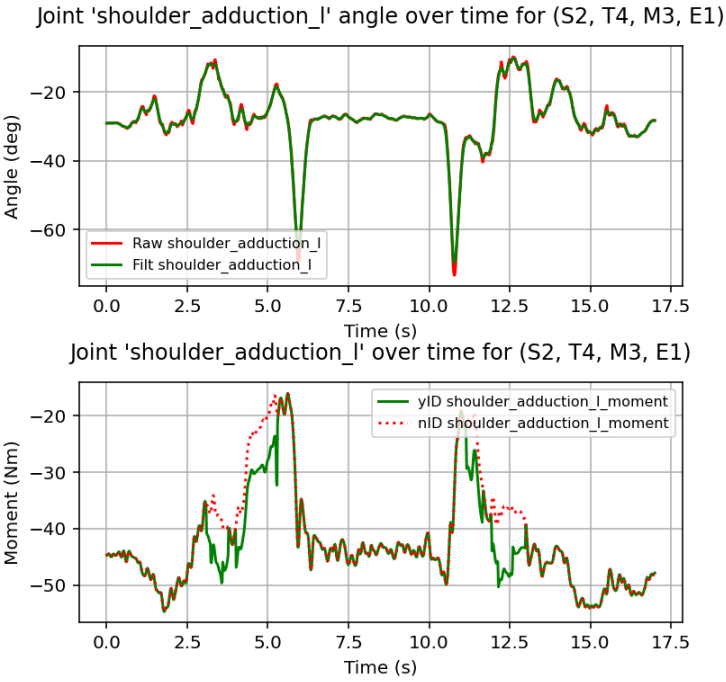
24.C: Lumbar rotation.



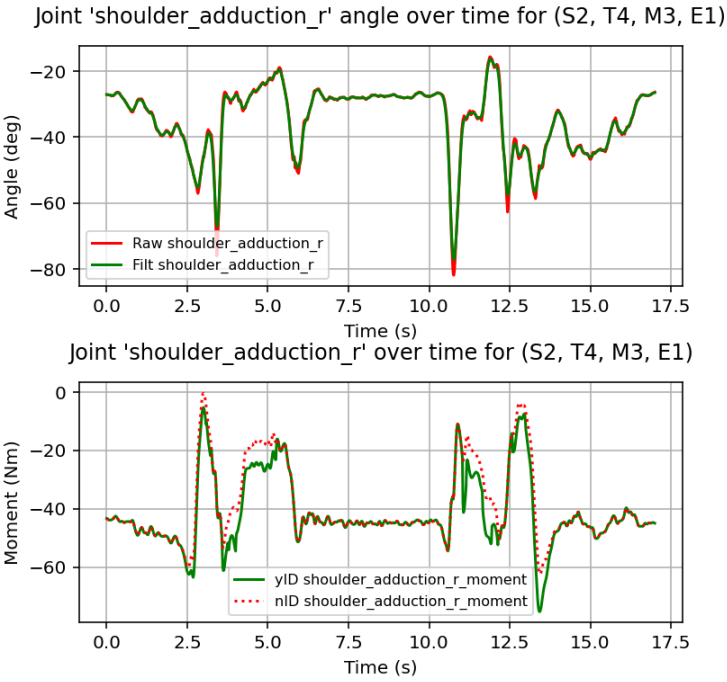
24.D: Shoulder flexion-extension of left arm.



24.E: Shoulder flexion-extension of right arm.

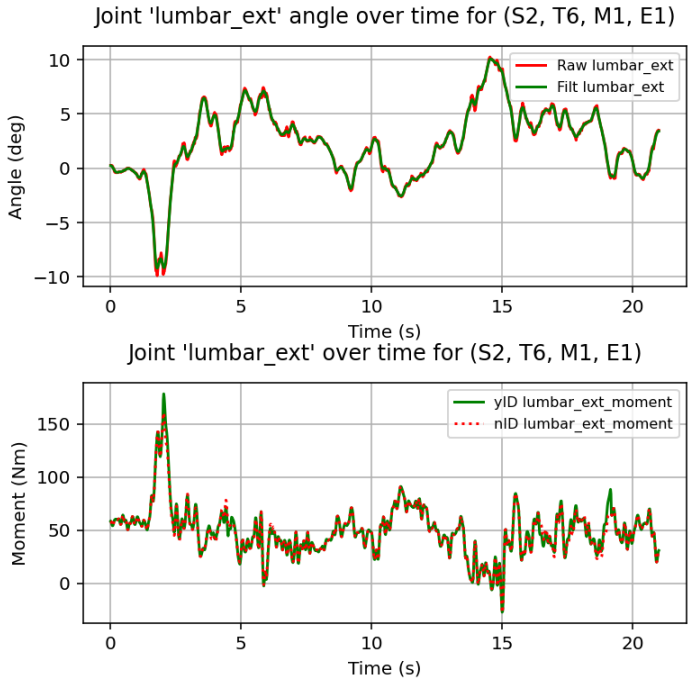


24.F: Shoulder adduction-abduction of left arm.



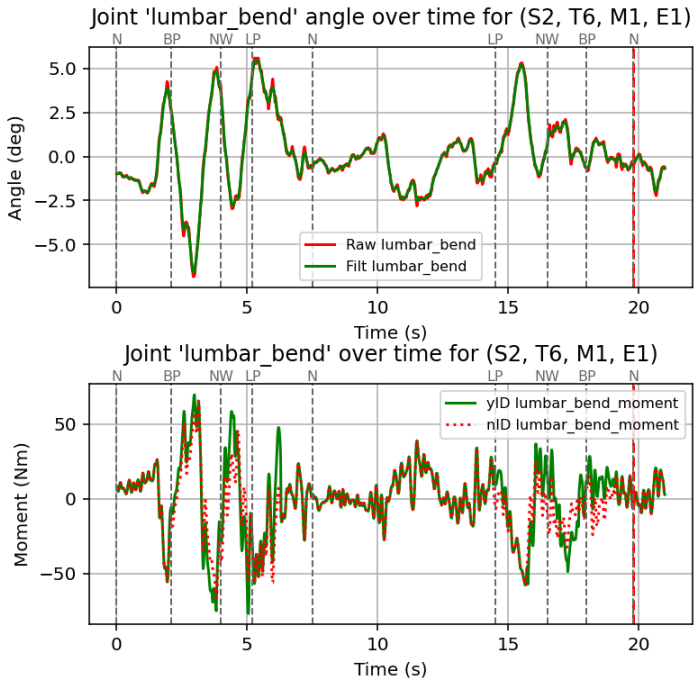
24.G: Shoulder adduction-abduction of right arm.

Fig. 24: Results from Inverse Dynamics for Subject 2, Trial 4. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

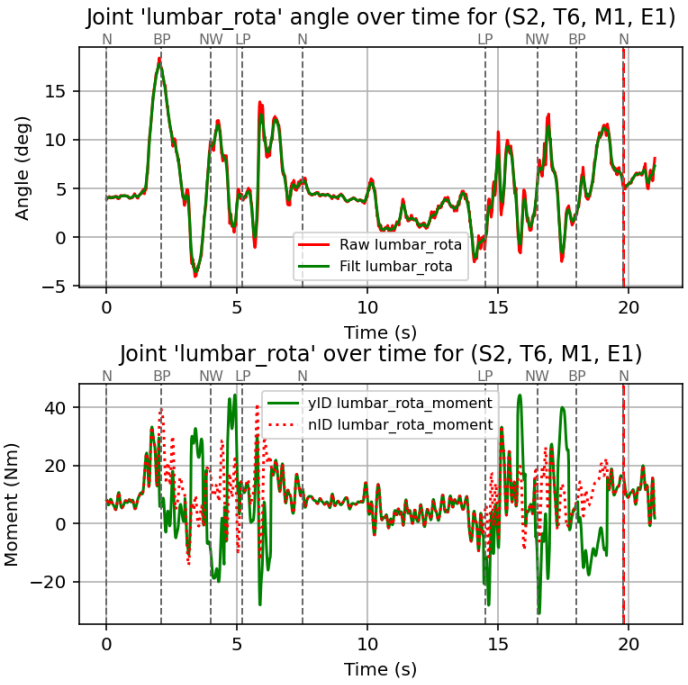


Joint Angle and Moments results for Subject 2, Trial 6

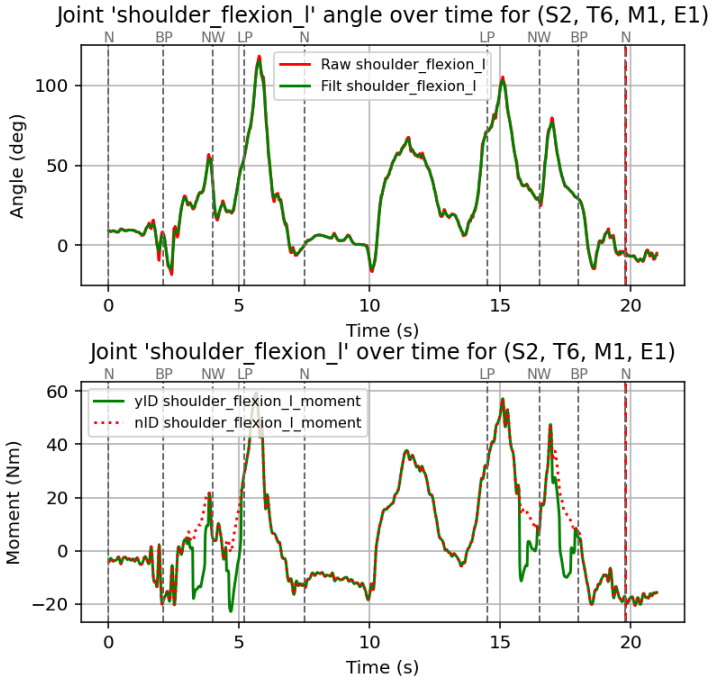
25.A: Lumbar extension.



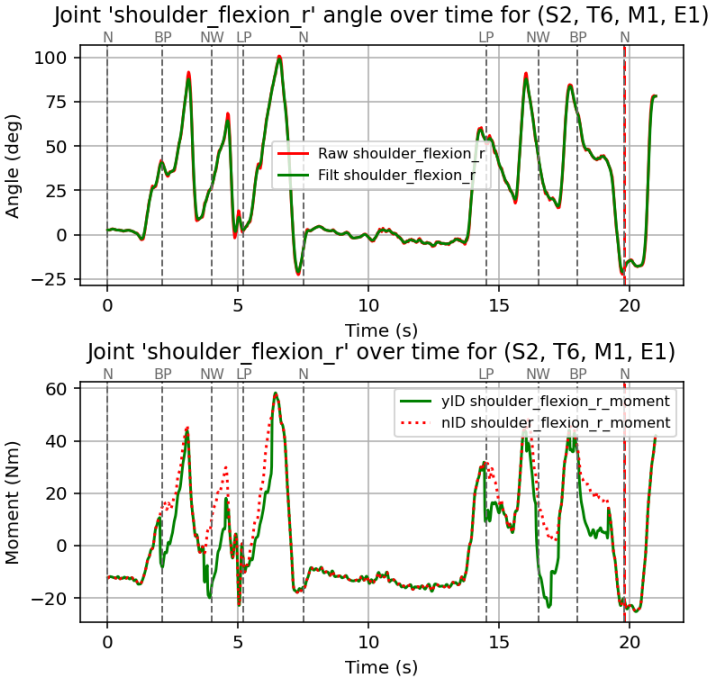
25.B: Lumbar bending.



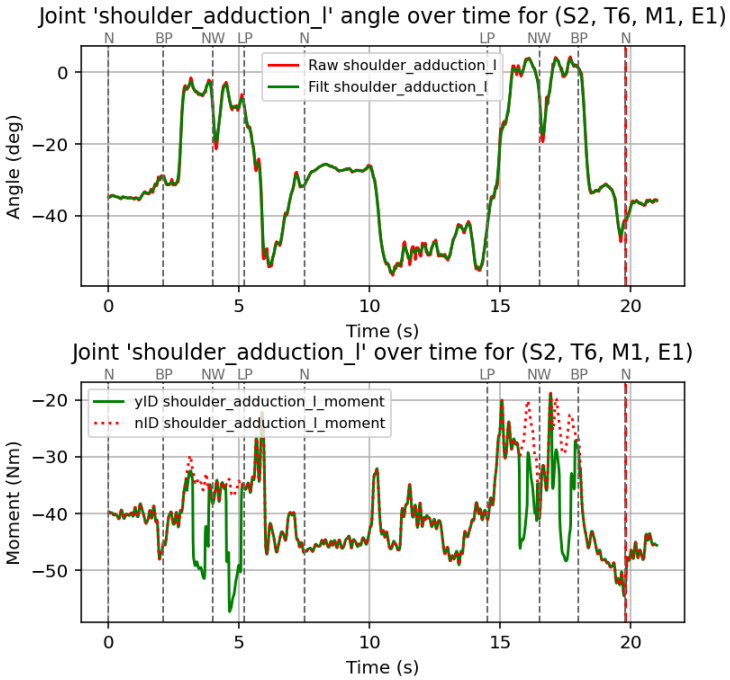
25.C: Lumbar rotation.



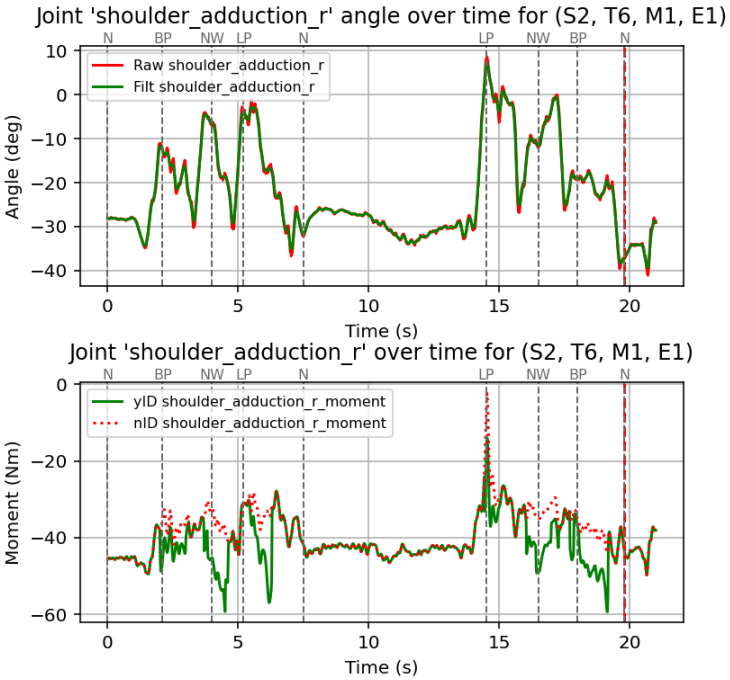
25.D: Shoulder flexion-extension of left arm.



25.E: Shoulder flexion-extension of right arm.

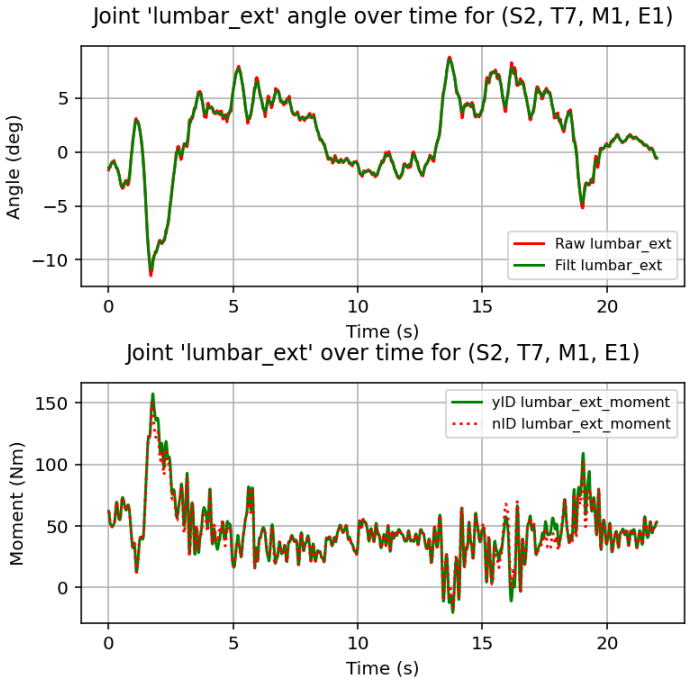


25.F: Shoulder adduction-abduction of left arm.



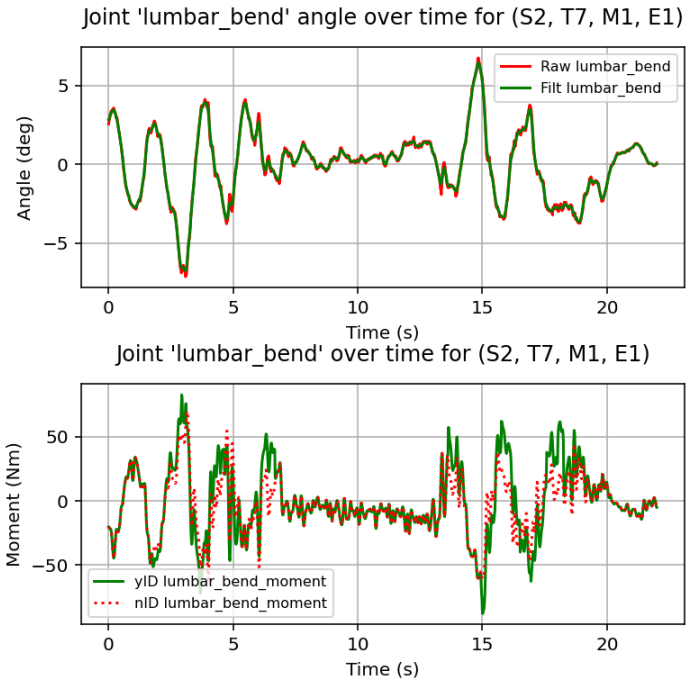
25.G: Shoulder adduction-abduction of right arm.

Fig. 25: Results from Inverse Dynamics for Subject 2, Trial 6. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

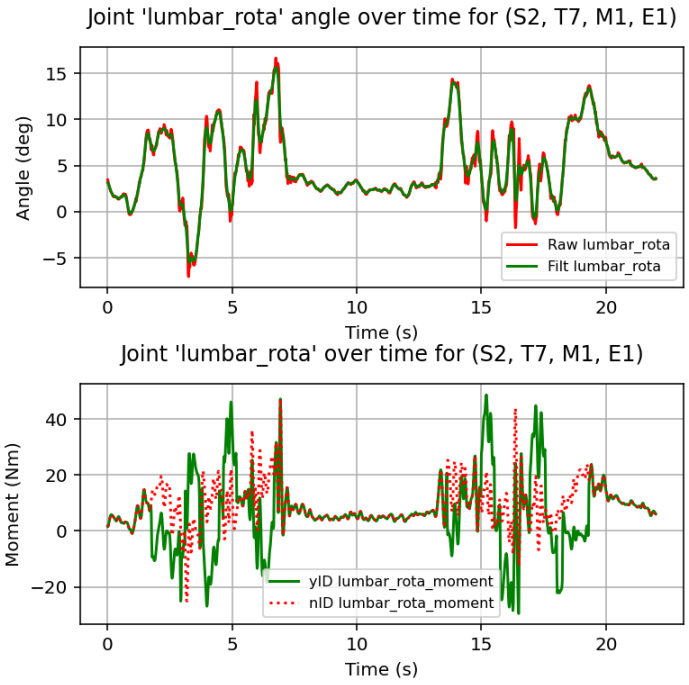


Joint Angle and Moments results for Subject 2, Trial 7

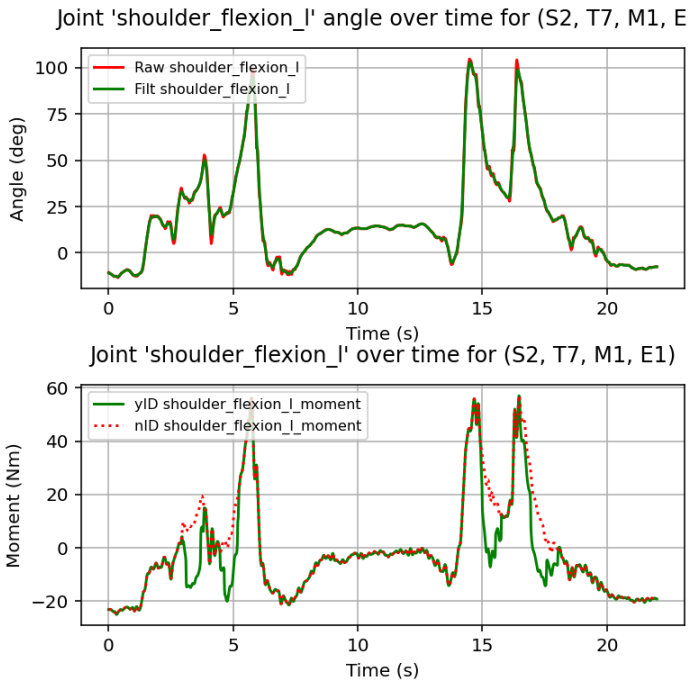
26.A: Lumbar extension.



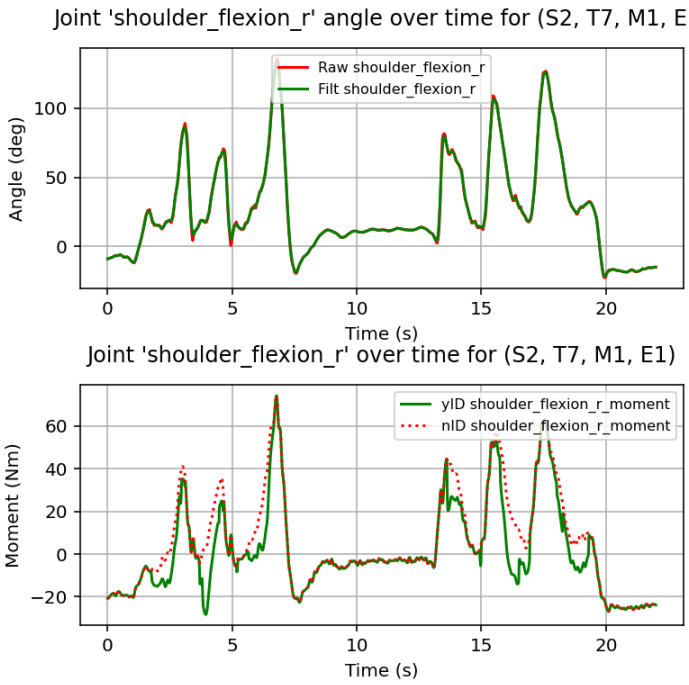
26.B: Lumbar bending.



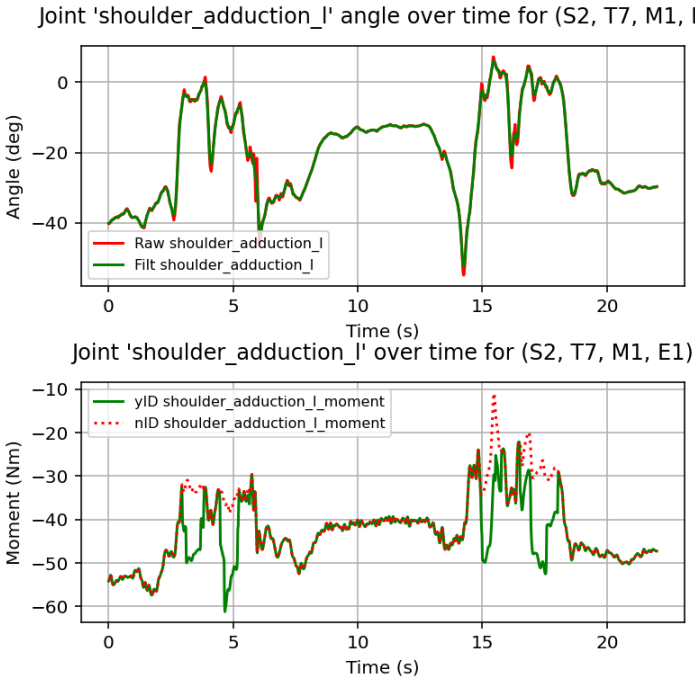
26.C: Lumbar rotation.



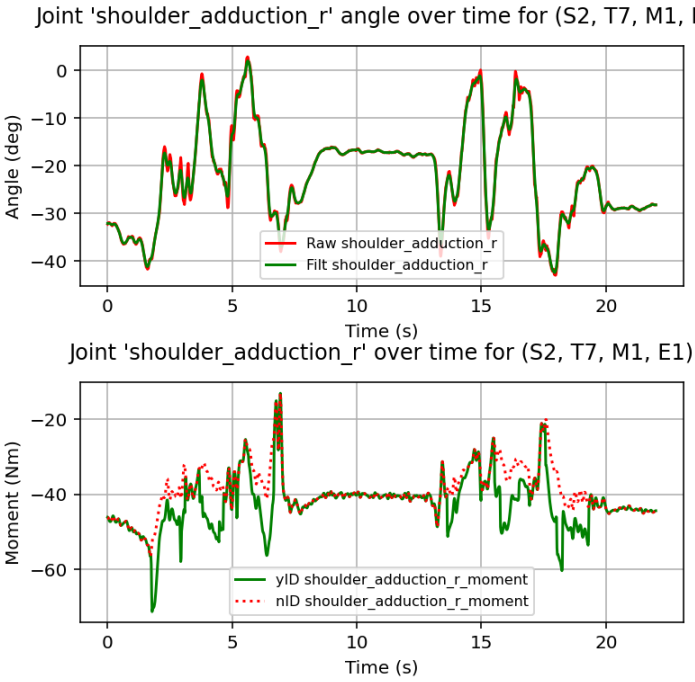
26.D: Shoulder flexion-extension of left arm.



26.E: Shoulder flexion-extension of right arm.

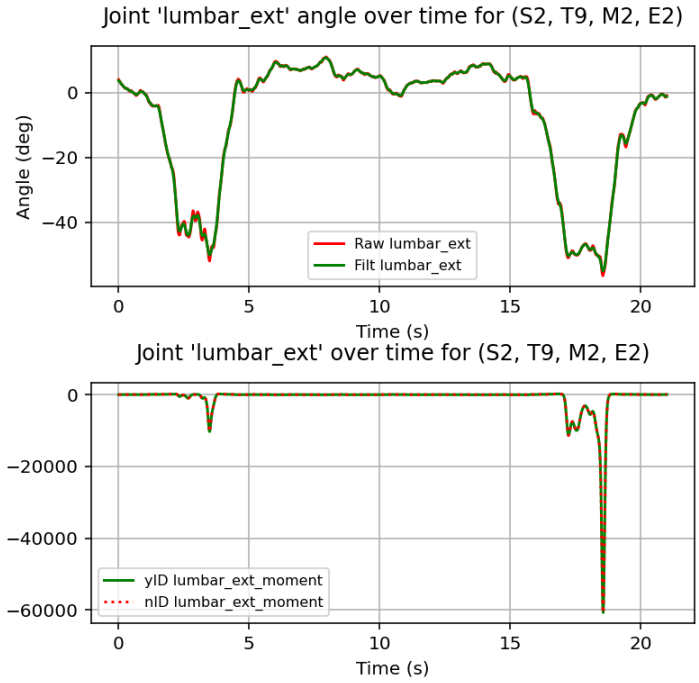


26.F: Shoulder adduction-abduction of left arm.



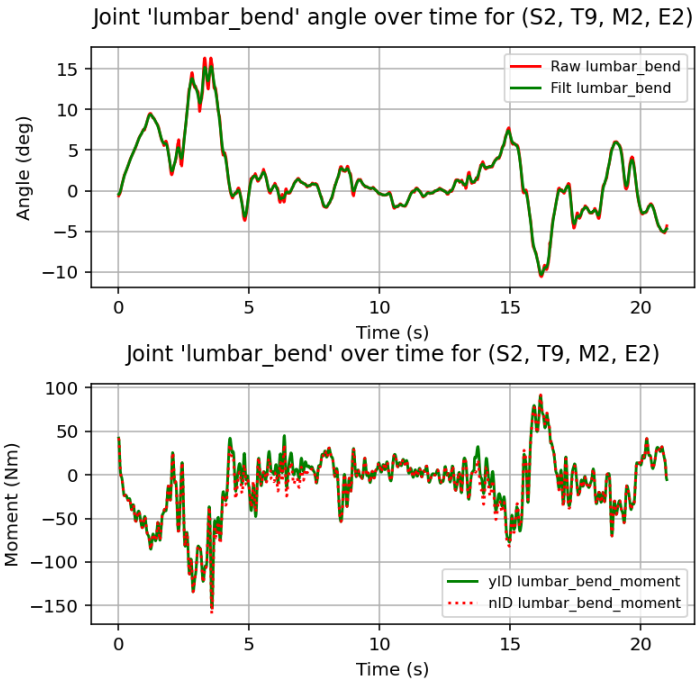
26.G: Shoulder adduction-abduction of right arm.

Fig. 26: Results from Inverse Dynamics for Subject 2, Trial 7. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

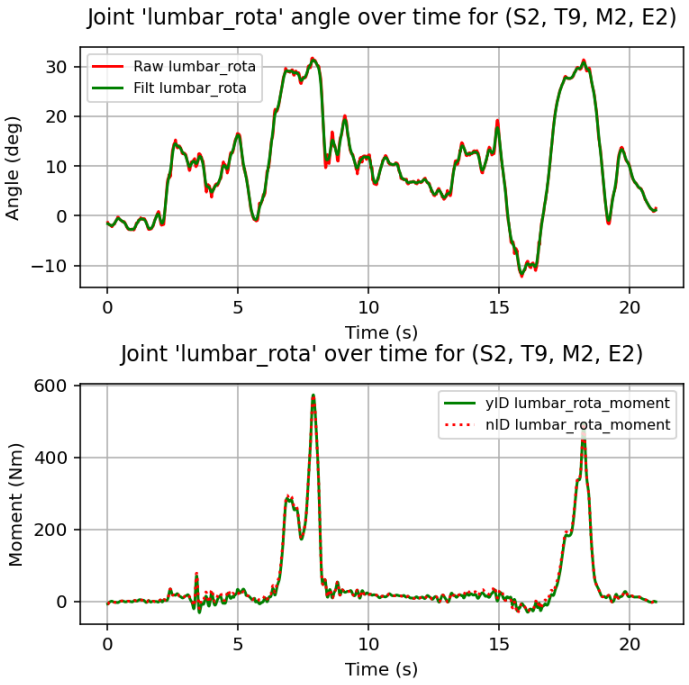


Joint Angle and Moments results for Subject 2, Trial 9

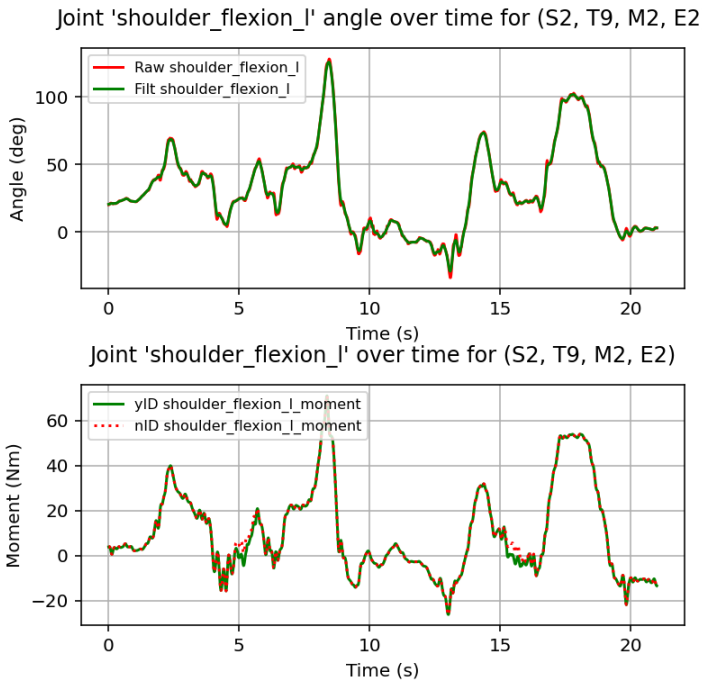
27.A: Lumbar extension.



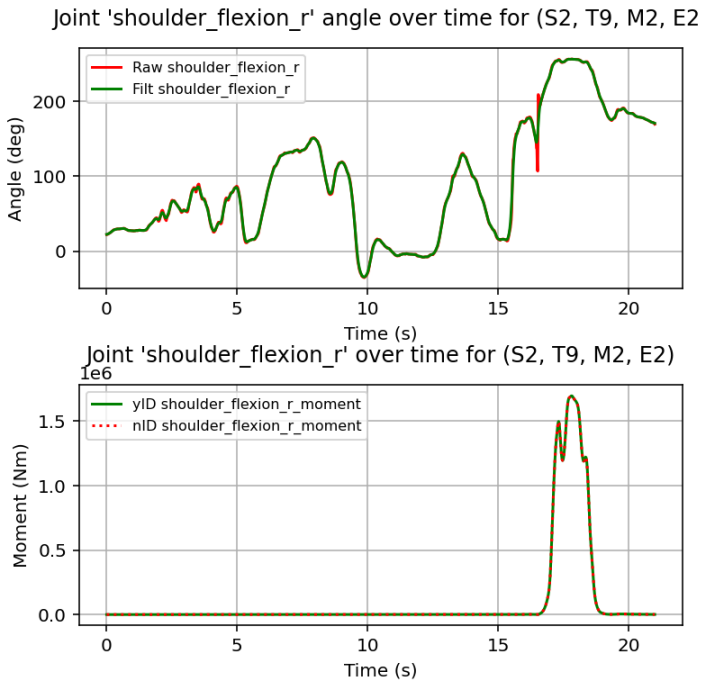
27.B: Lumbar bending.



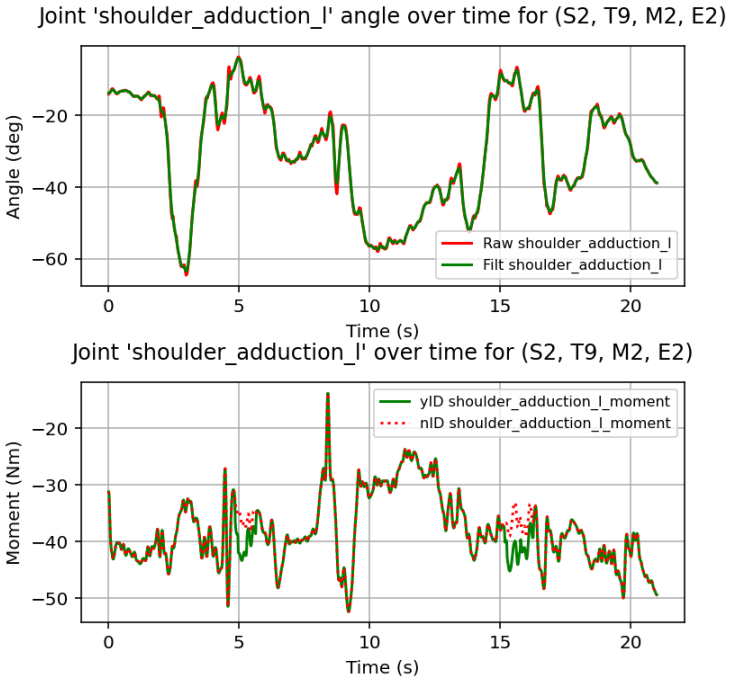
27.C: Lumbar rotation.



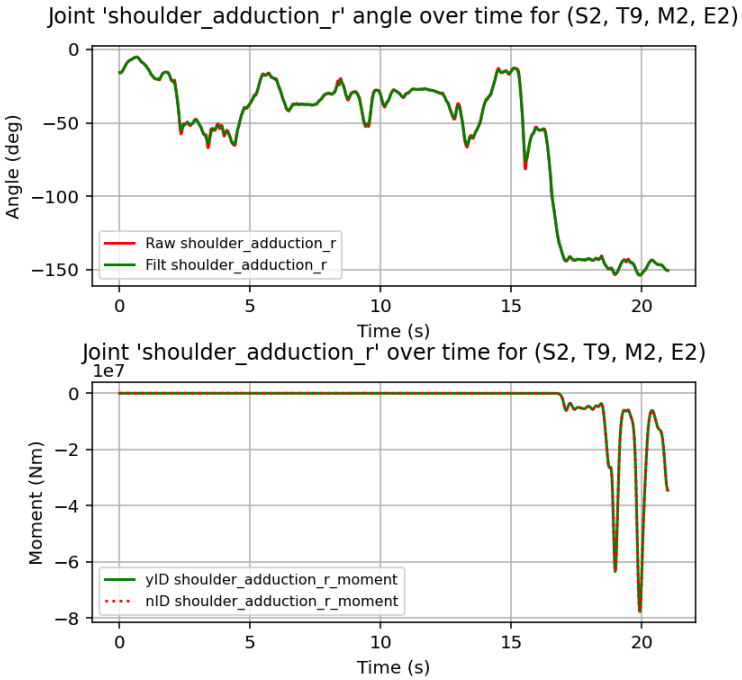
27.D: Shoulder flexion-extension of left arm.



27.E: Shoulder flexion-extension of right arm.

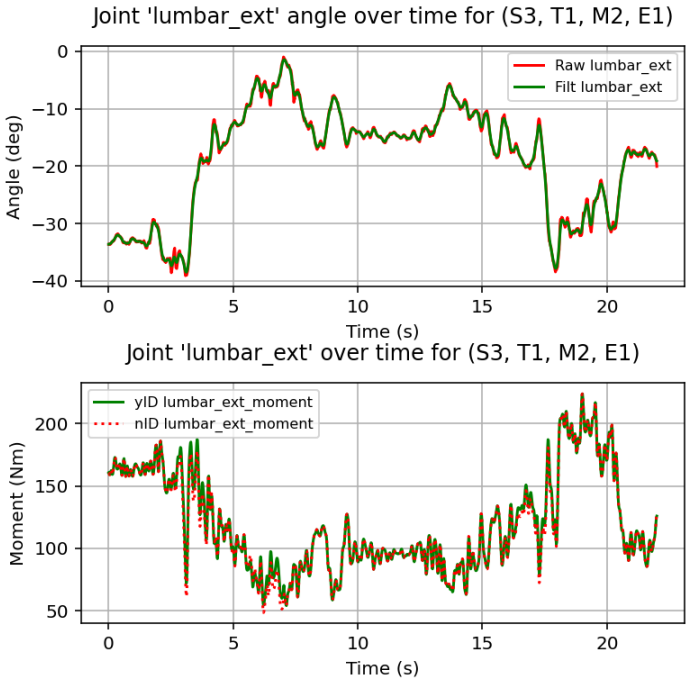


27.F: Shoulder adduction-abduction of left arm.



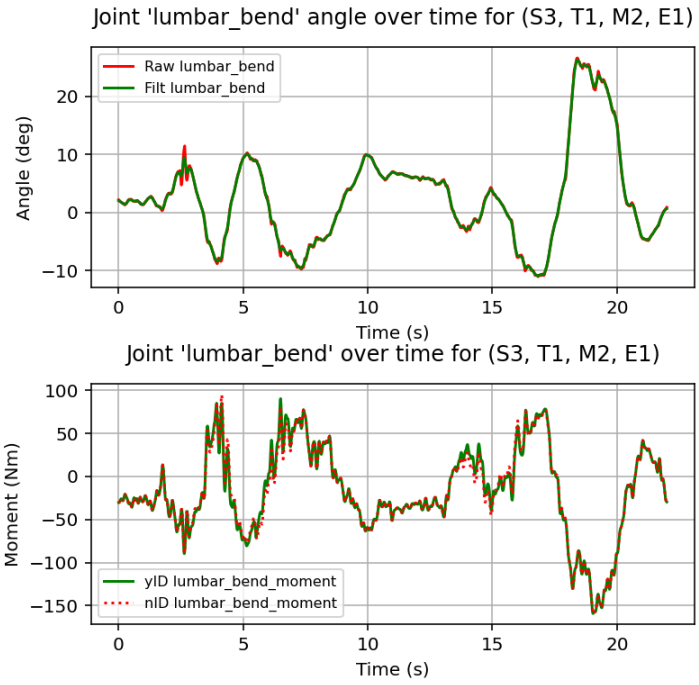
27.G: Shoulder adduction-abduction of right arm.

Fig. 27: Results from Inverse Dynamics for Subject 2, Trial 9. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

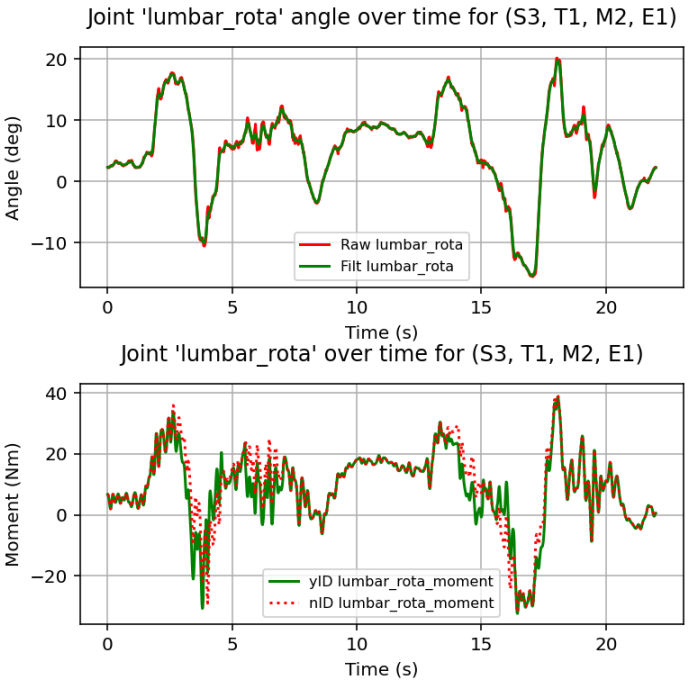


Joint Angle and Moments results for Subject 3, Trial 1

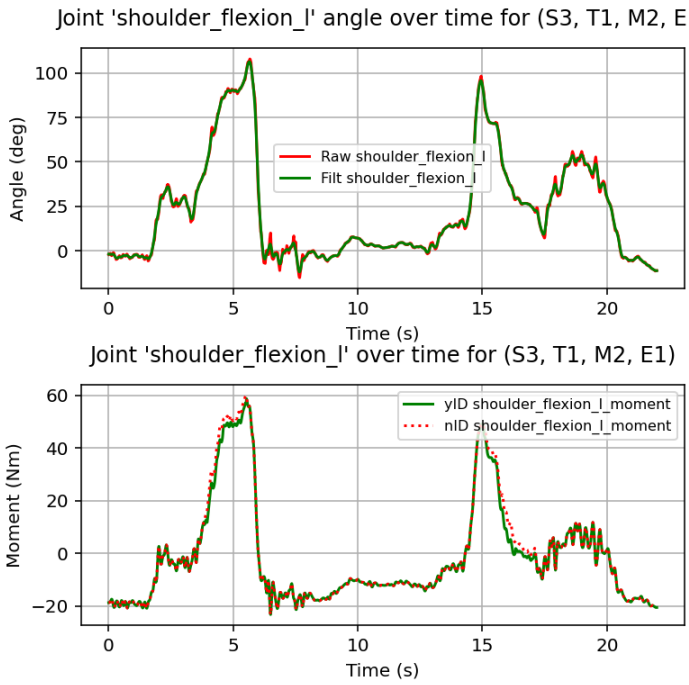
28.A: Lumbar extension.



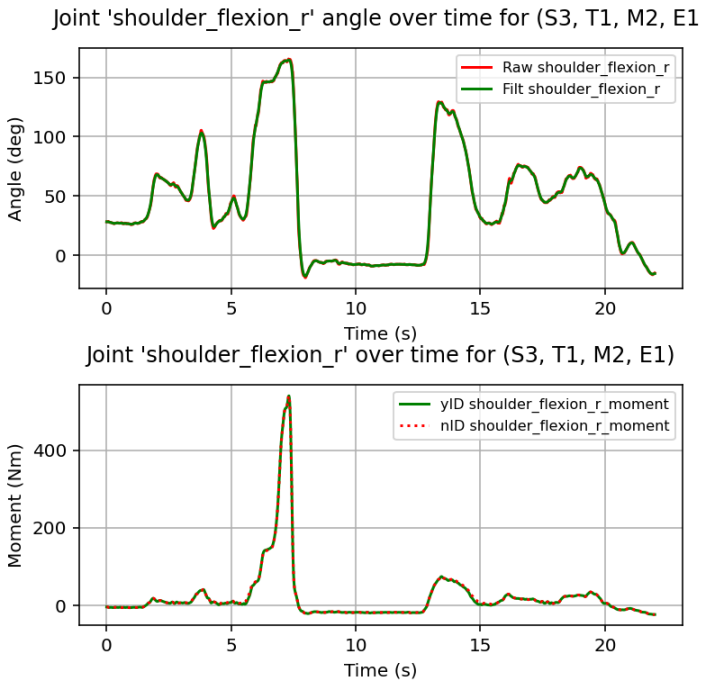
28.B: Lumbar bending.



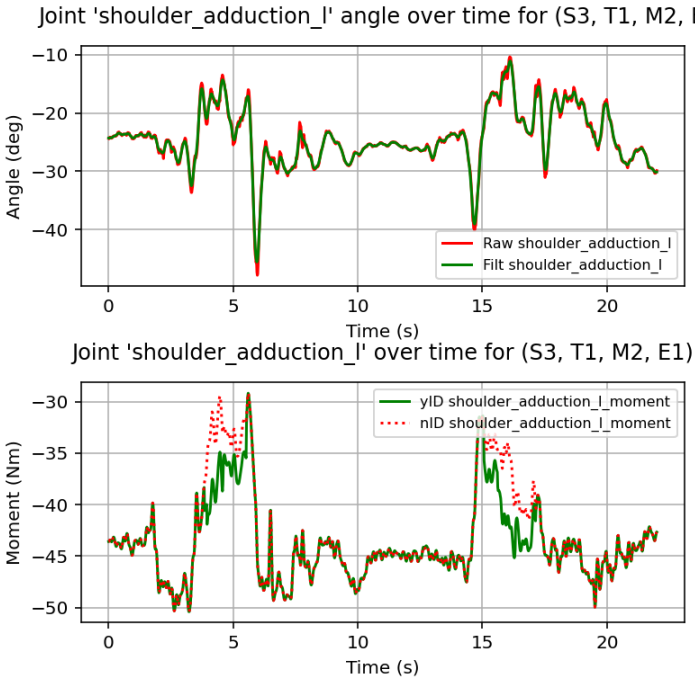
28.C: Lumbar rotation.



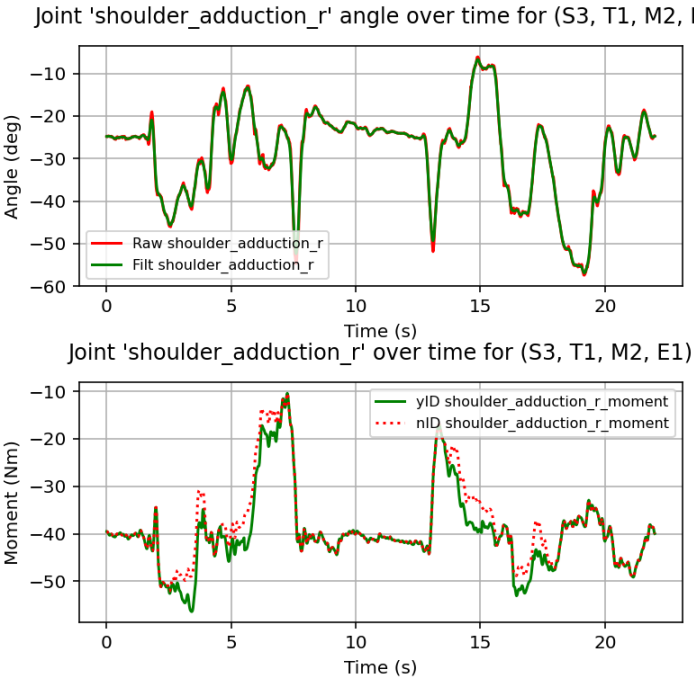
28.D: Shoulder flexion-extension of left arm.



28.E: Shoulder flexion-extension of right arm.

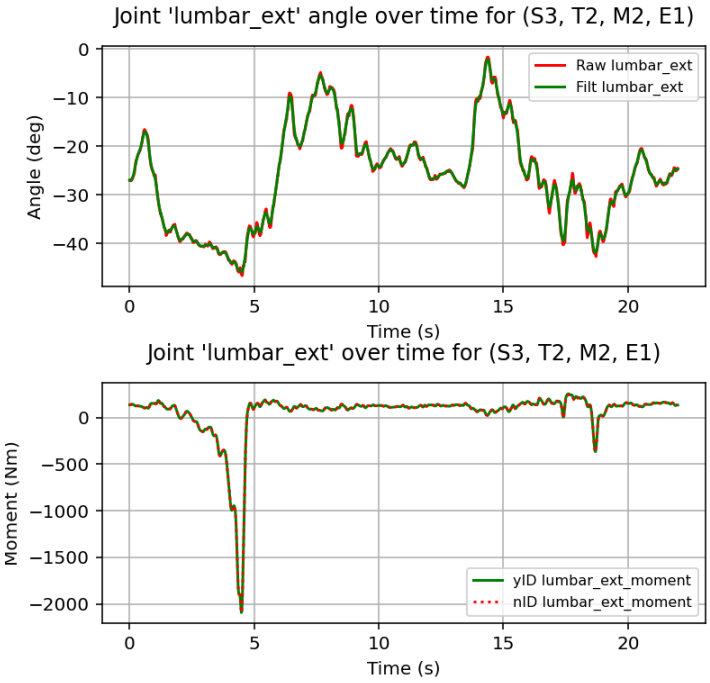


28.F: Shoulder adduction-abduction of left arm.



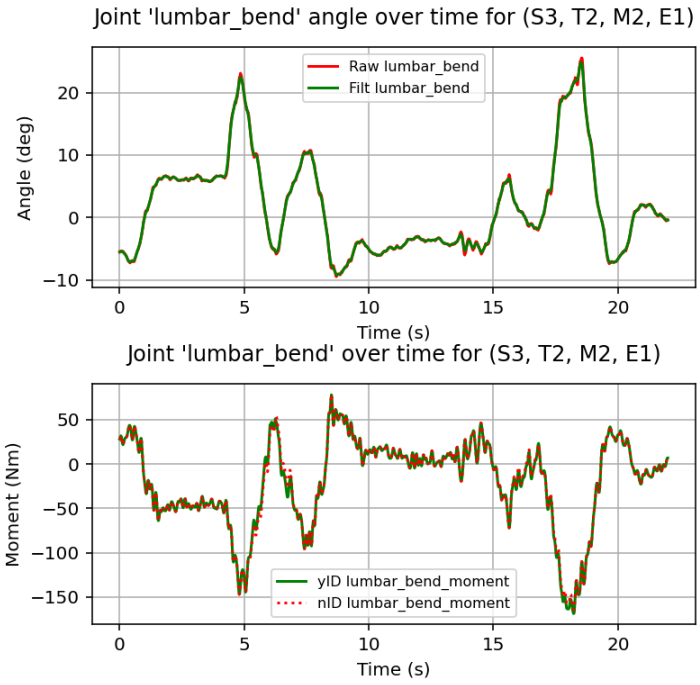
28.G: Shoulder adduction-abduction of right arm.

Fig. 28: Results from Inverse Dynamics for Subject 3, Trial 1. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

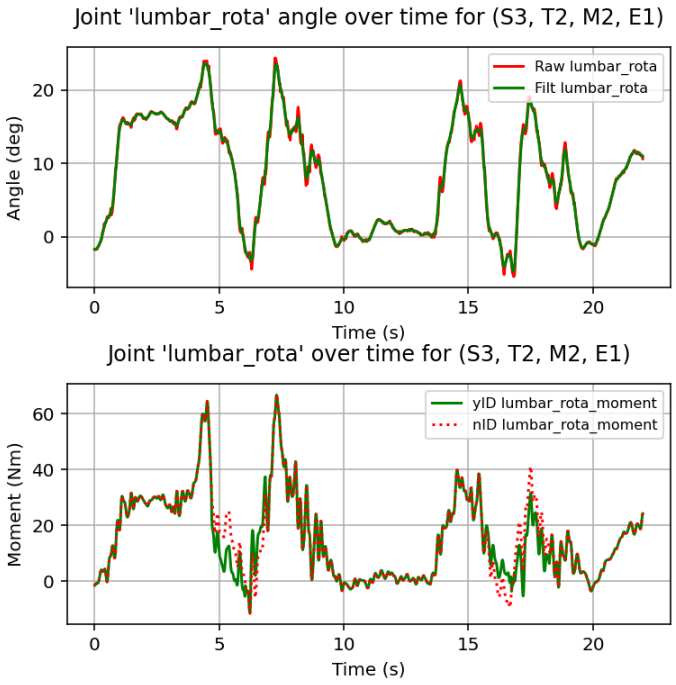


Joint Angle and Moments results for Subject 3, Trial 2

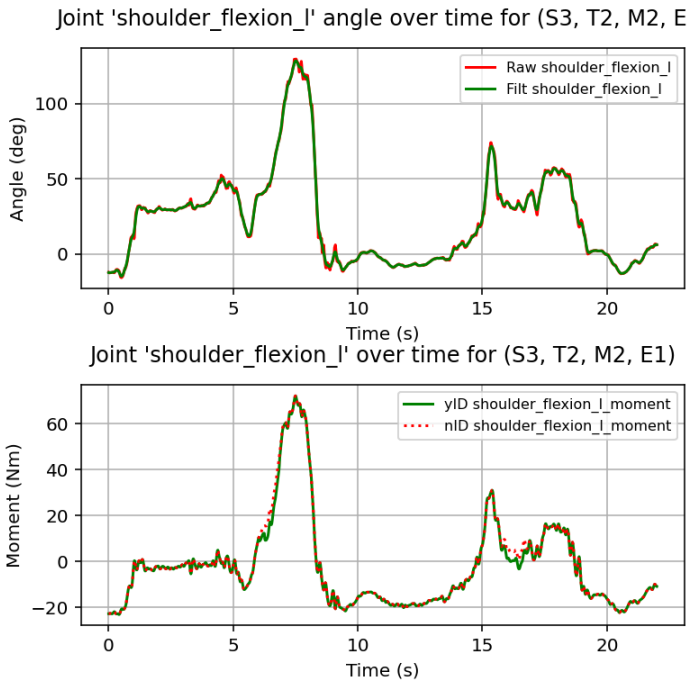
29.A: Lumbar extension.



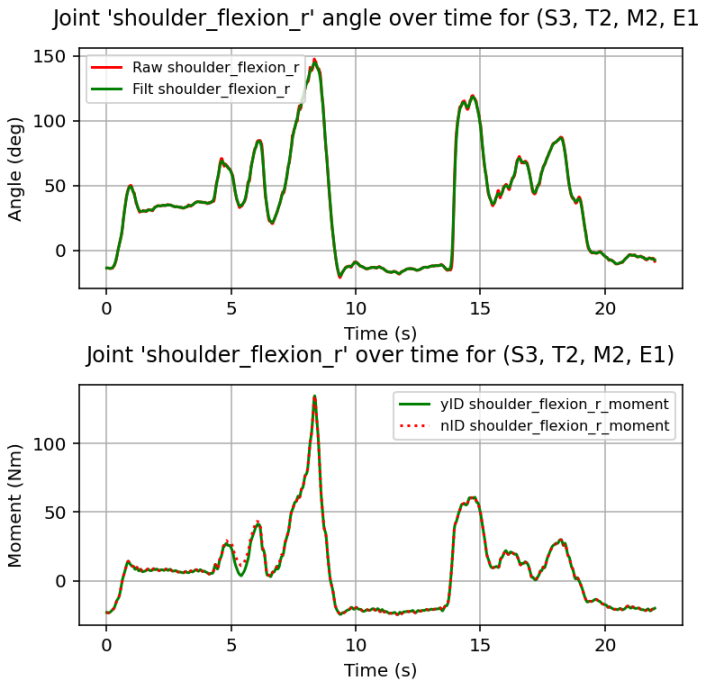
29.B: Lumbar bending.



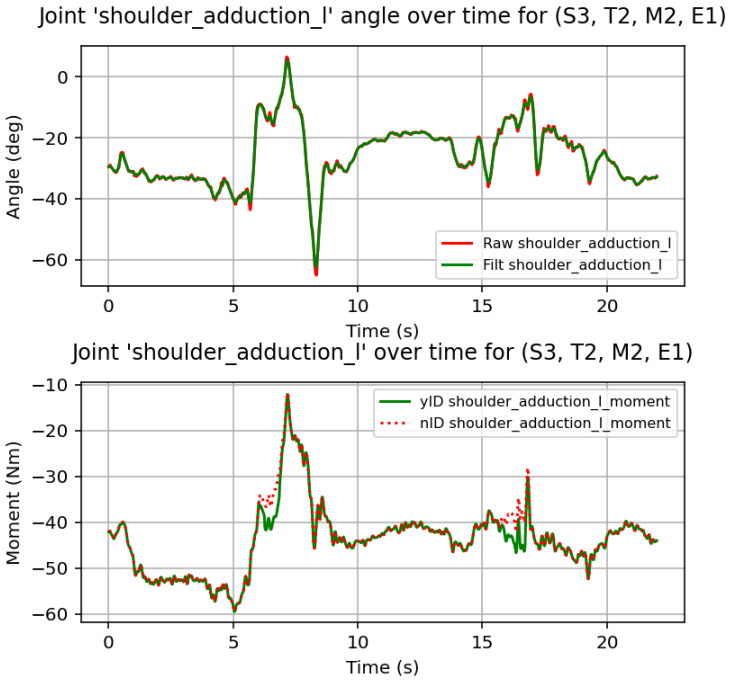
29.C: Lumbar rotation.



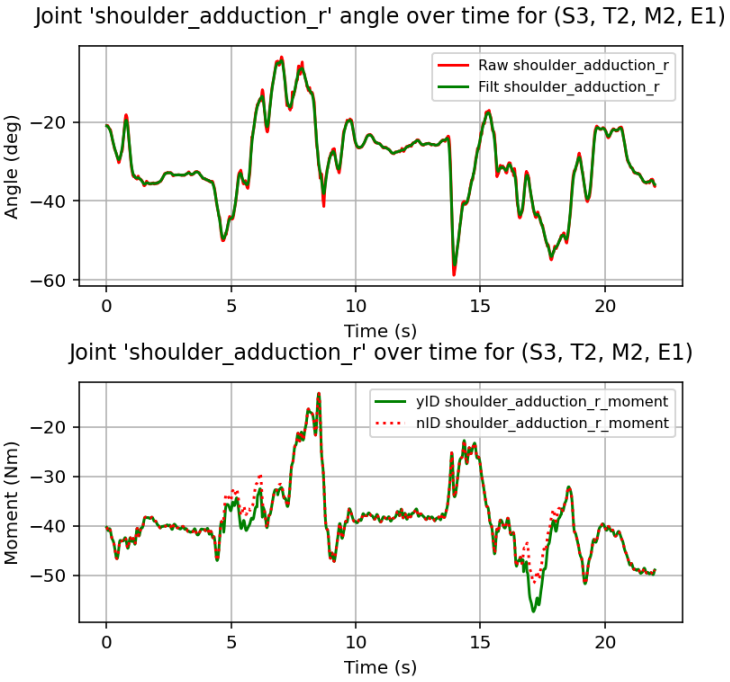
29.D: Shoulder flexion-extension of left arm.



29.E: Shoulder flexion-extension of right arm.

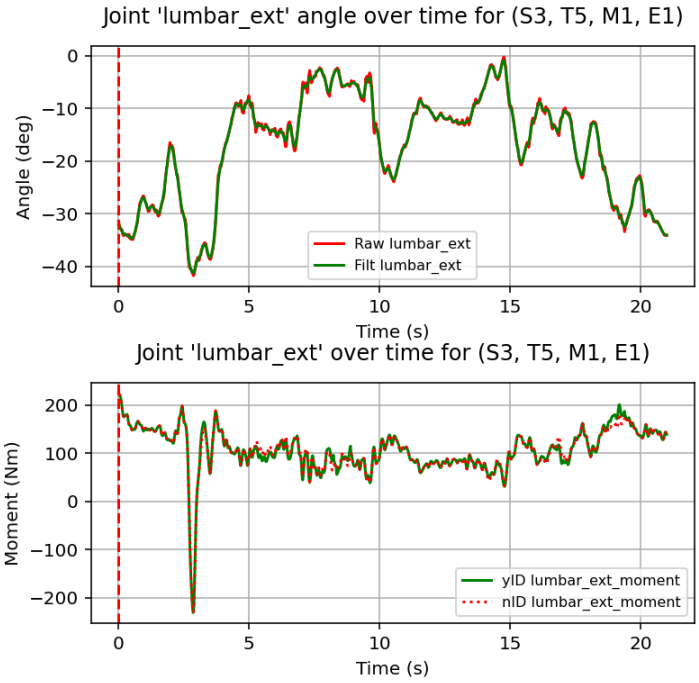


29.F: Shoulder adduction-abduction of left arm.



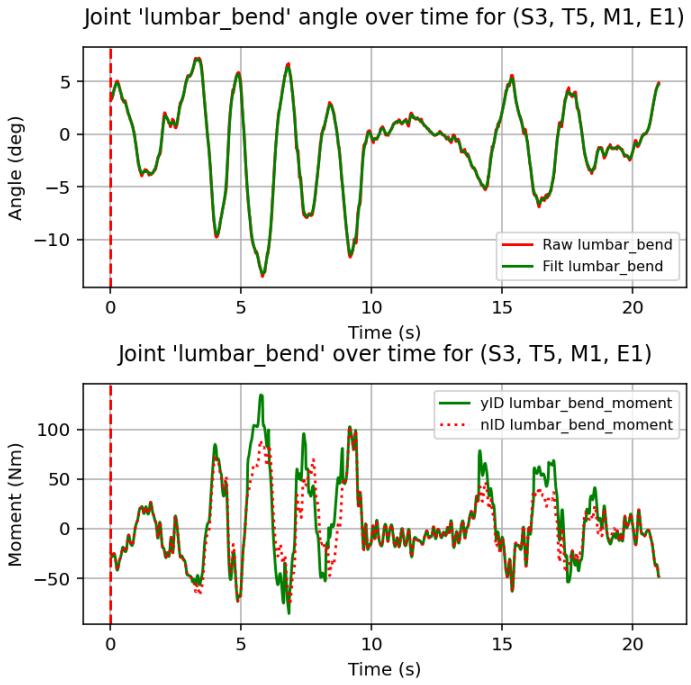
29.G: Shoulder adduction-abduction of right arm.

Fig. 29: Results from Inverse Dynamics for Subject 3, Trial 2. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

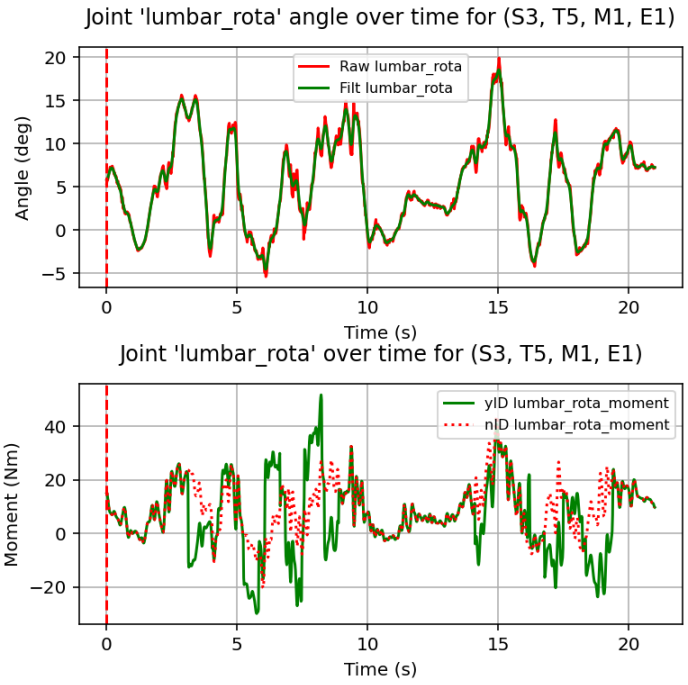


Joint Angle and Moments results for Subject 3, Trial 5

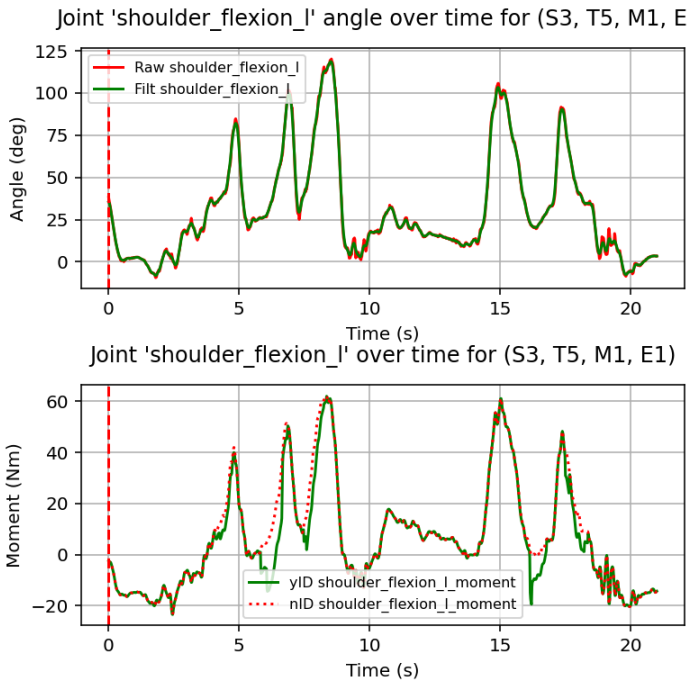
30.A: Lumbar extension.



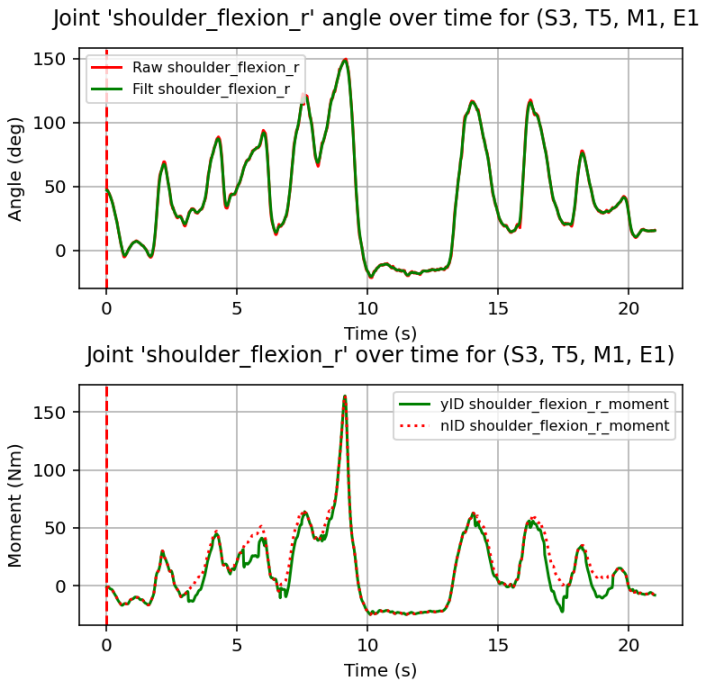
30.B: Lumbar bending.



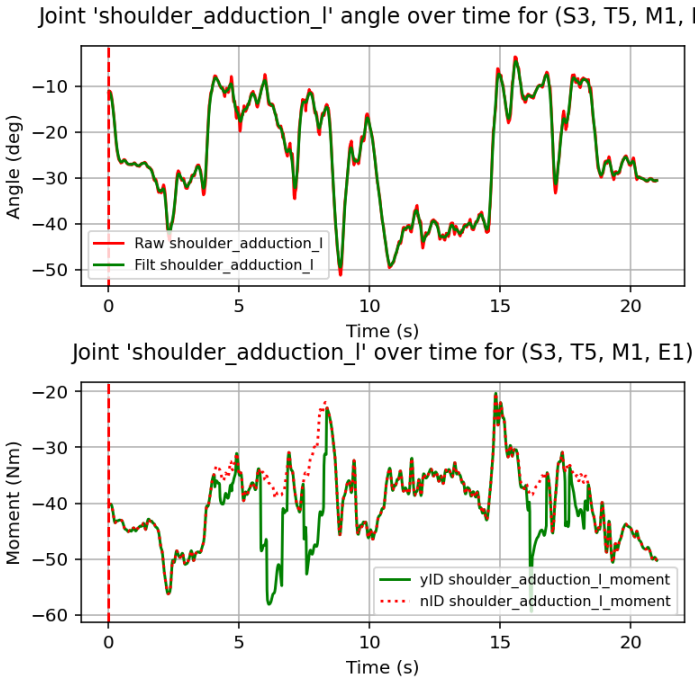
30.C: Lumbar rotation.



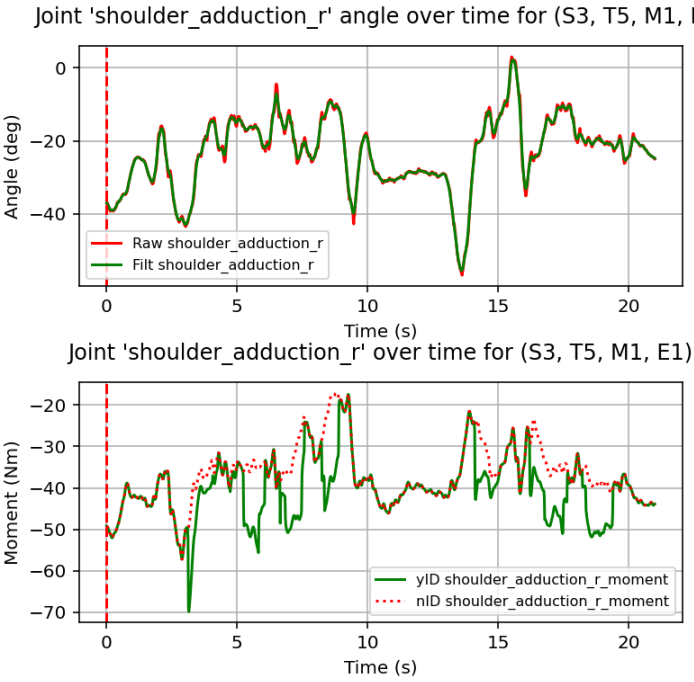
30.D: Shoulder flexion-extension of left arm.



30.E: Shoulder flexion-extension of right arm.



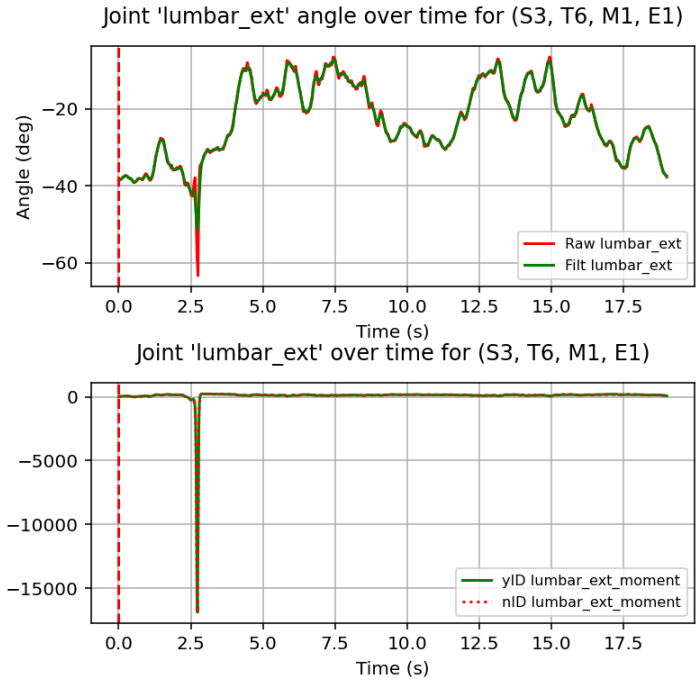
30.F: Shoulder adduction-abduction of left arm.



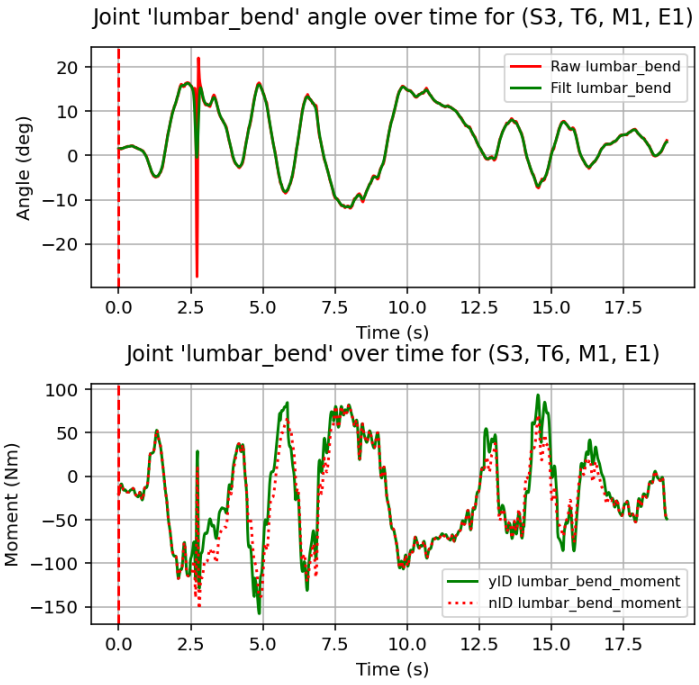
30.G: Shoulder adduction-abduction of right arm.

Fig. 30: Results from Inverse Dynamics for Subject 3, Trial 5. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

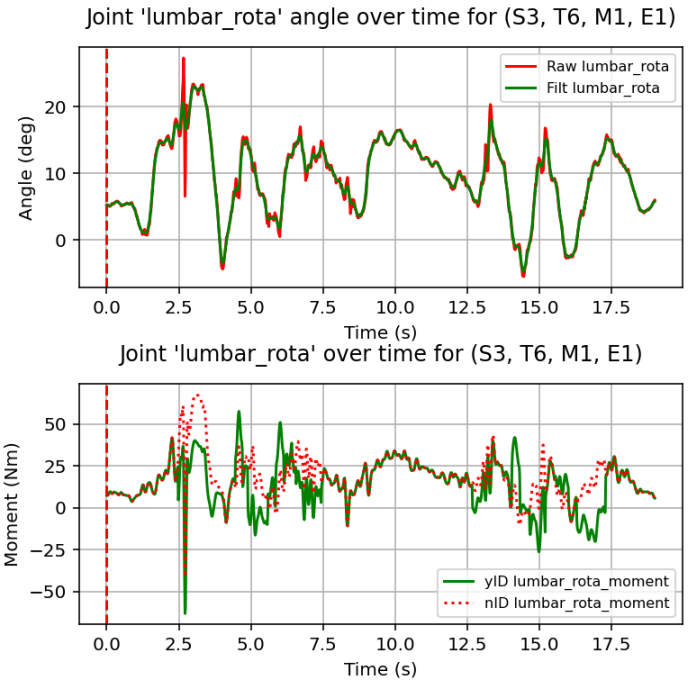
Joint Angle and Moments results for Subject 3, Trial 6



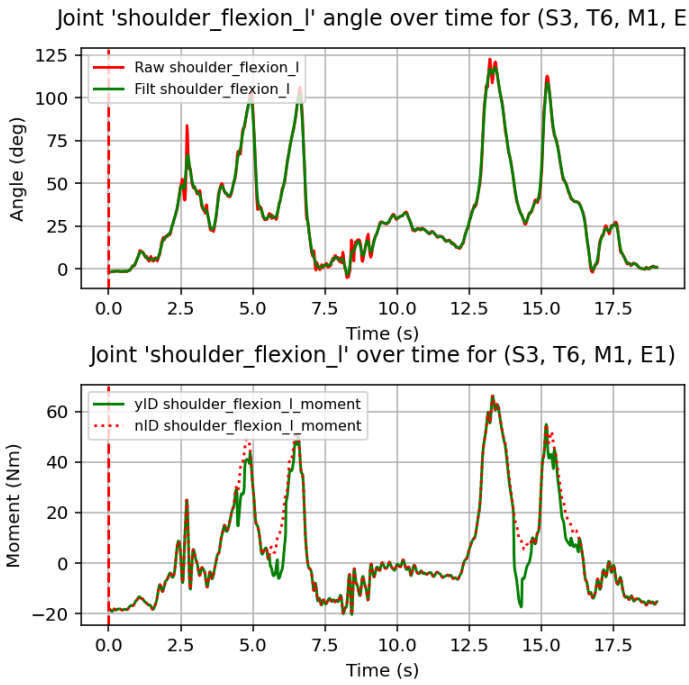
31.A: Lumbar extension.



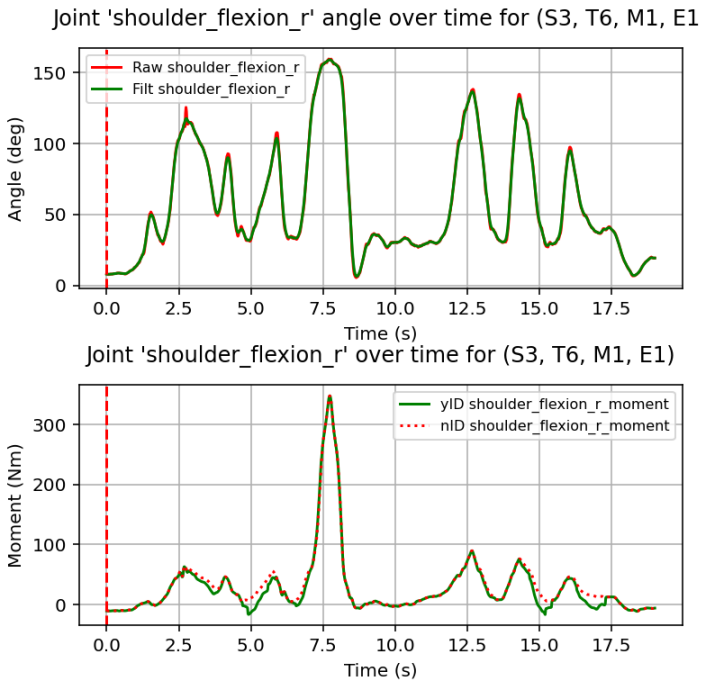
31.B: Lumbar bending.



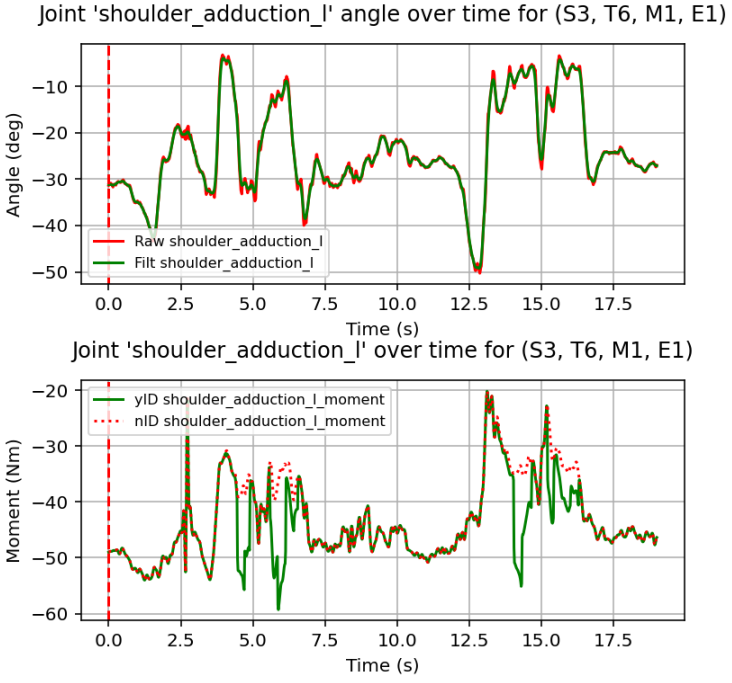
31.C: Lumbar rotation.



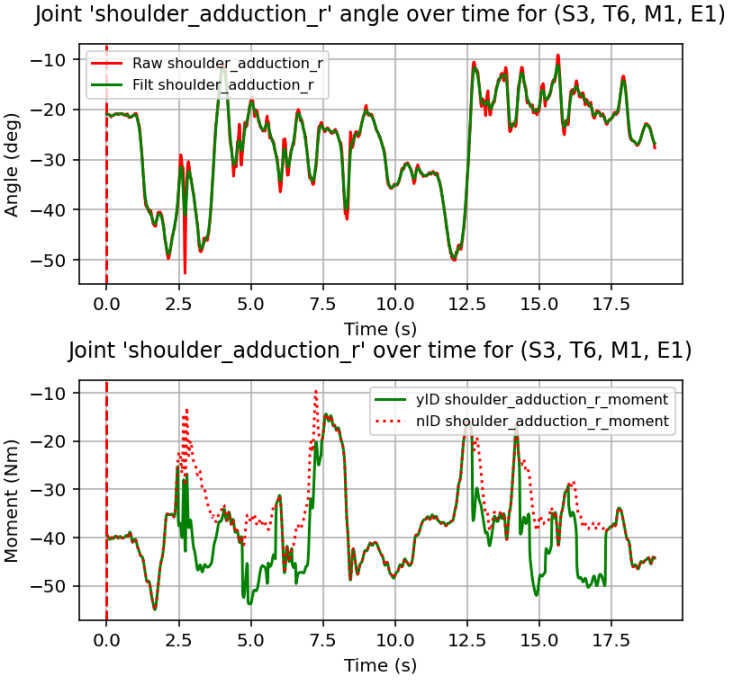
31.D: Shoulder flexion-extension of left arm.



31.E: Shoulder flexion-extension of right arm.

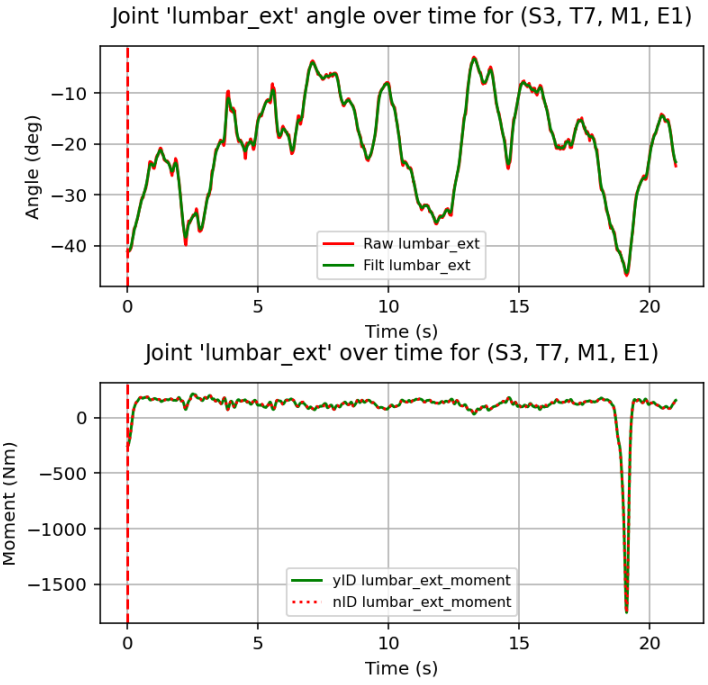


31.F: Shoulder adduction-abduction of left arm.



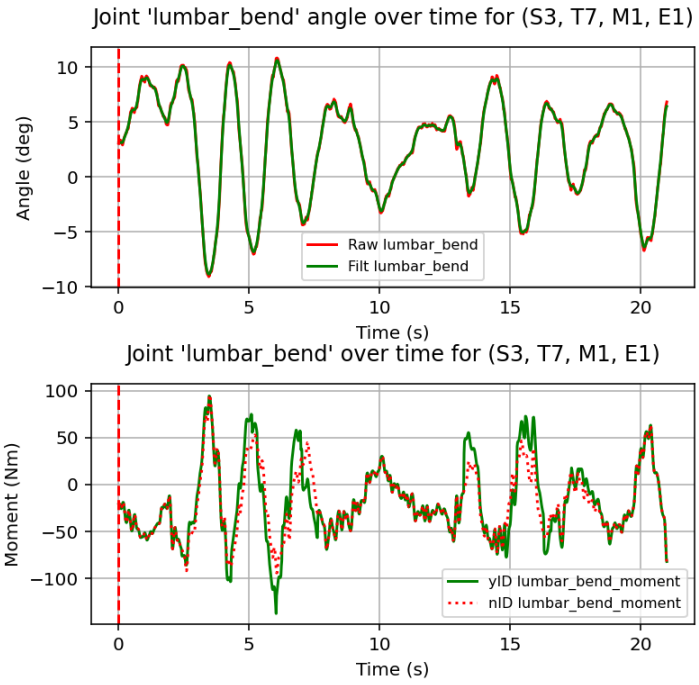
31.G: Shoulder adduction-abduction of right arm.

Fig. 31: Results from Inverse Dynamics for Subject 3, Trial 6. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

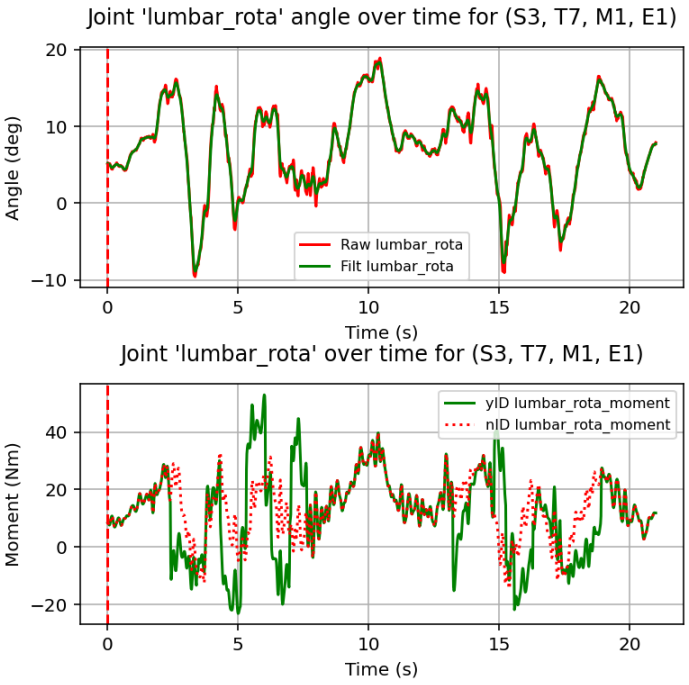


Joint Angle and Moments results for Subject 3, Trial 7

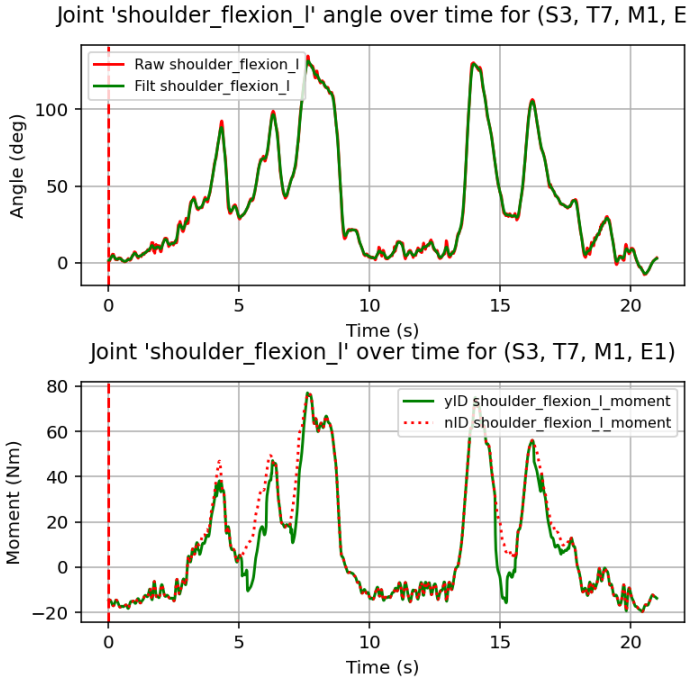
32.A: Lumbar extension.



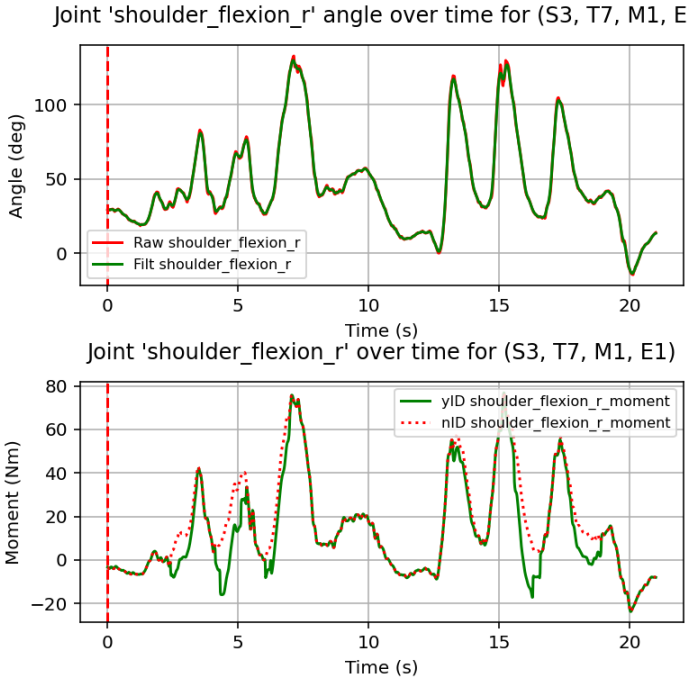
32.B: Lumbar bending.



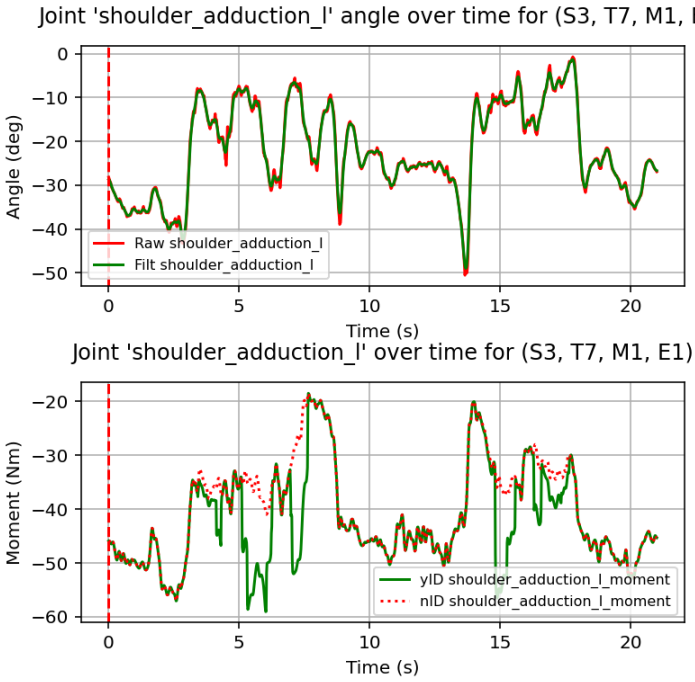
32.C: Lumbar rotation.



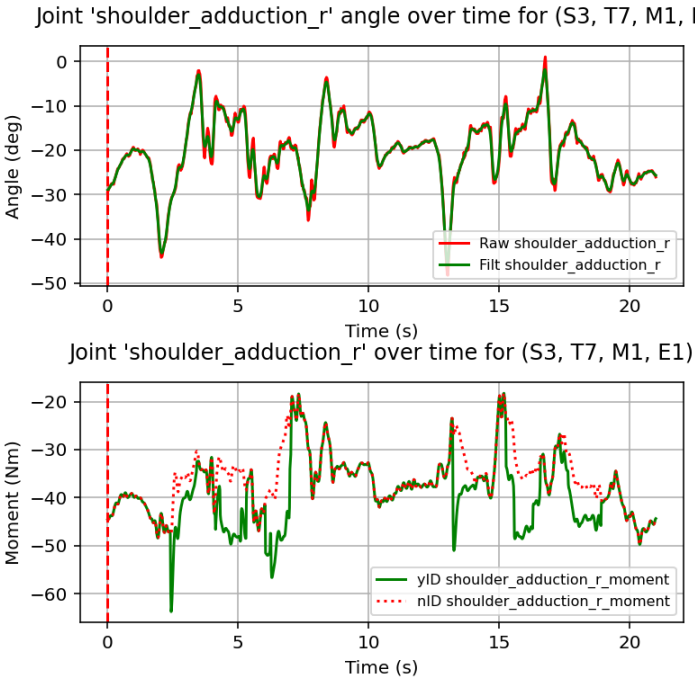
32.D: Shoulder flexion-extension of left arm.



32.E: Shoulder flexion-extension of right arm.

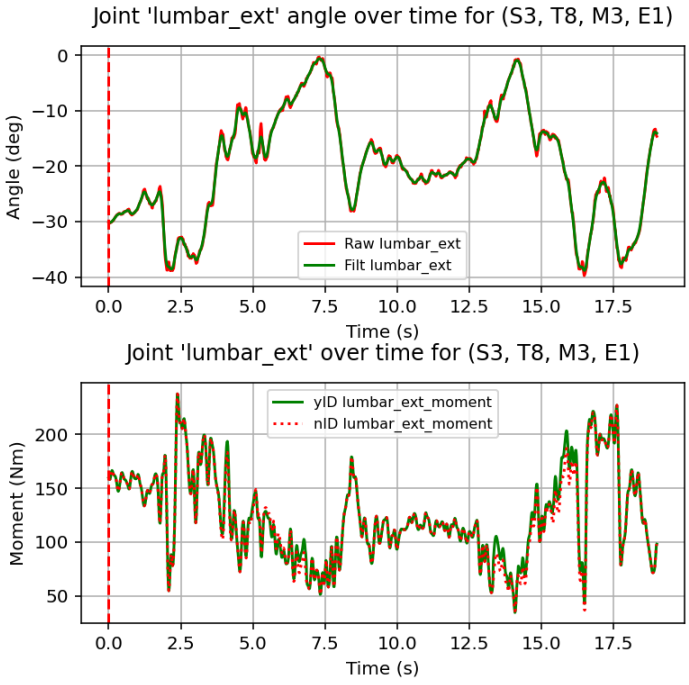


32.F: Shoulder adduction-abduction of left arm.



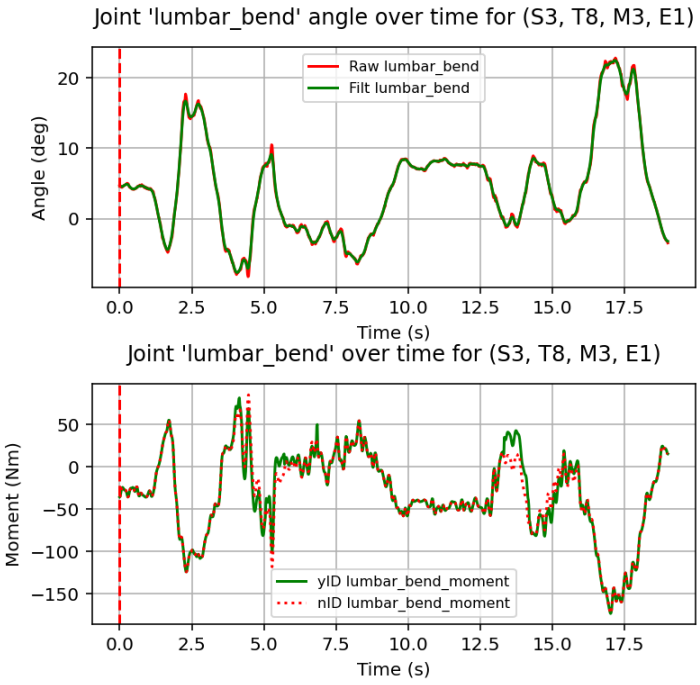
32.G: Shoulder adduction-abduction of right arm.

Fig. 32: Results from Inverse Dynamics for Subject 3, Trial 7. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

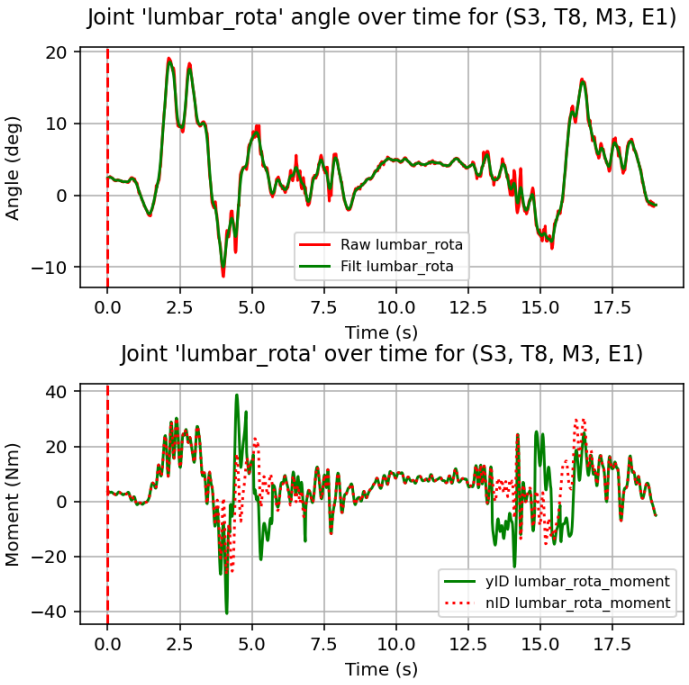


Joint Angle and Moments results for Subject 3, Trial 8

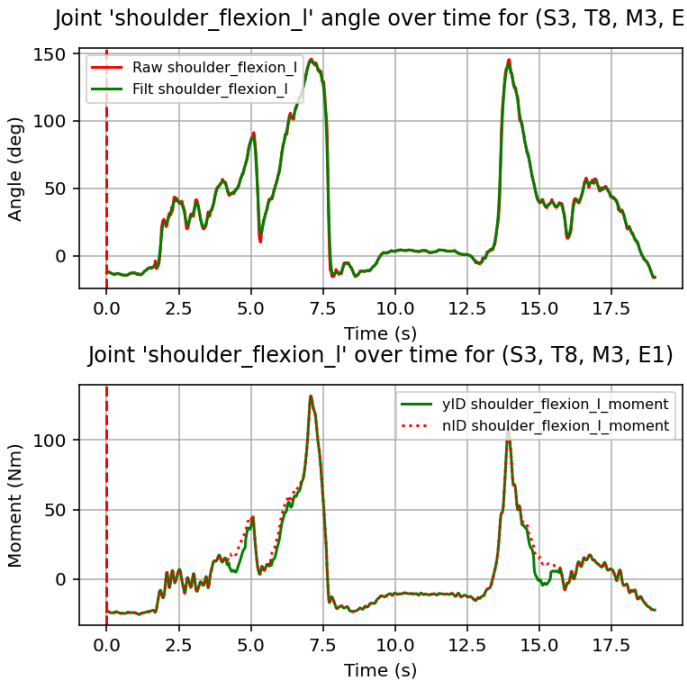
33.A: Lumbar extension.



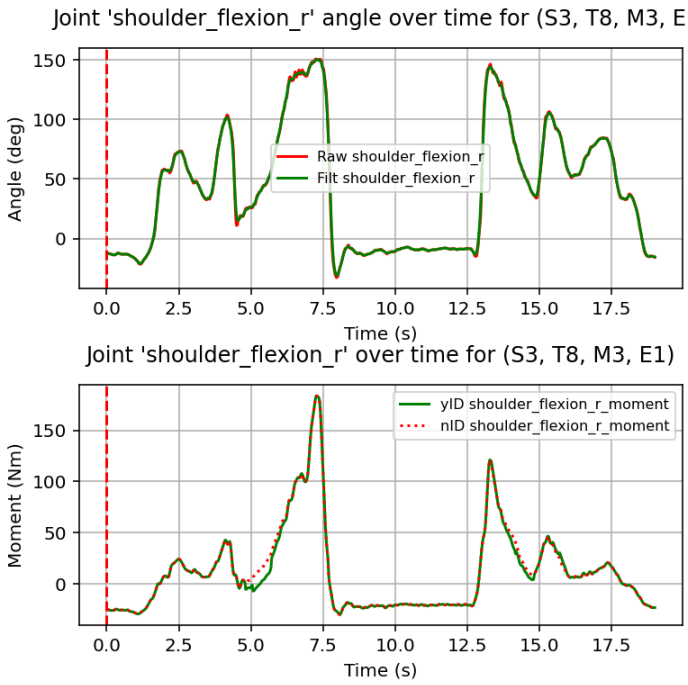
33.B: Lumbar bending.



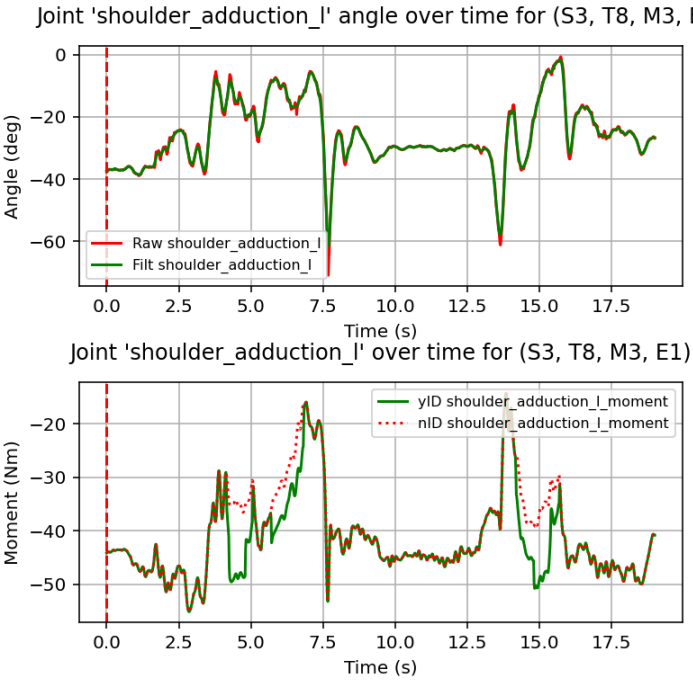
33.C: Lumbar rotation.



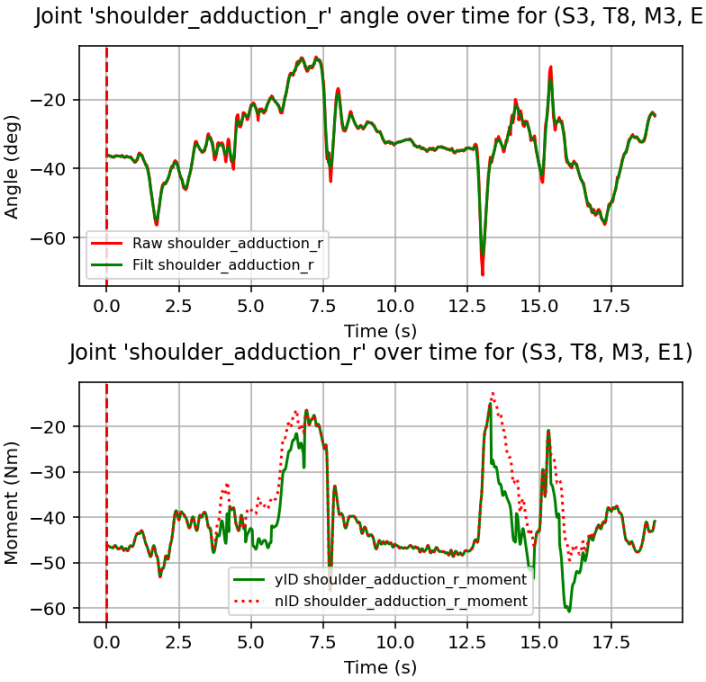
33.D: Shoulder flexion-extension of left arm.



33.E: Shoulder flexion-extension of right arm.

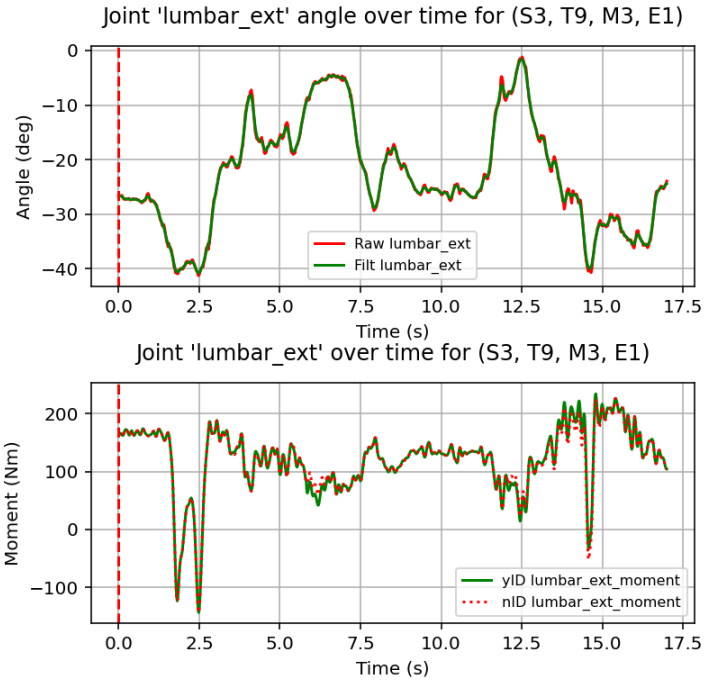


33.F: Shoulder adduction-abduction of left arm.



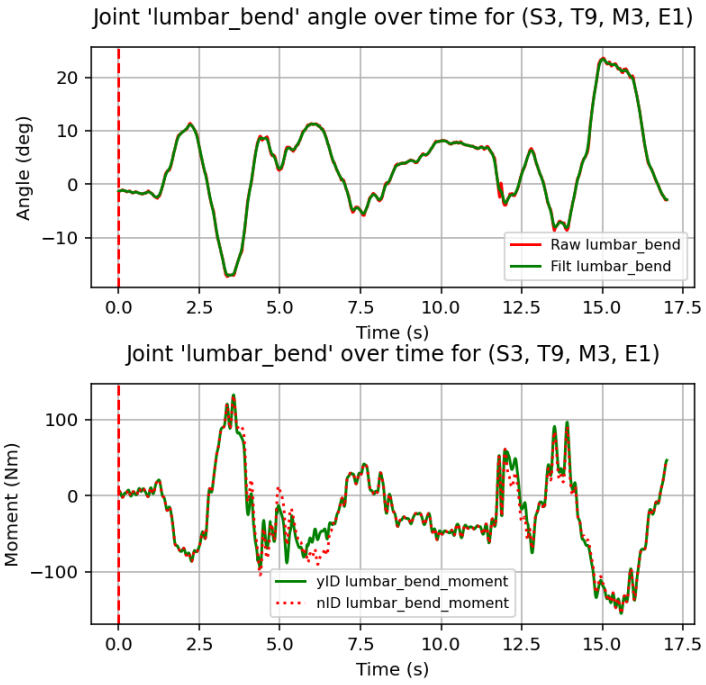
33.G: Shoulder adduction-abduction of right arm.

Fig. 33: Results from Inverse Dynamics for Subject 3, Trial 8. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

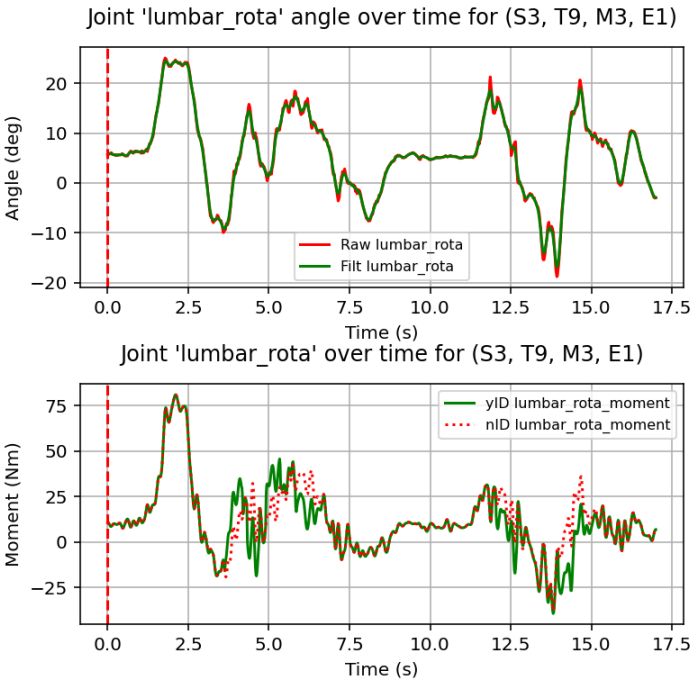


Joint Angle and Moments results for Subject 3, Trial 9

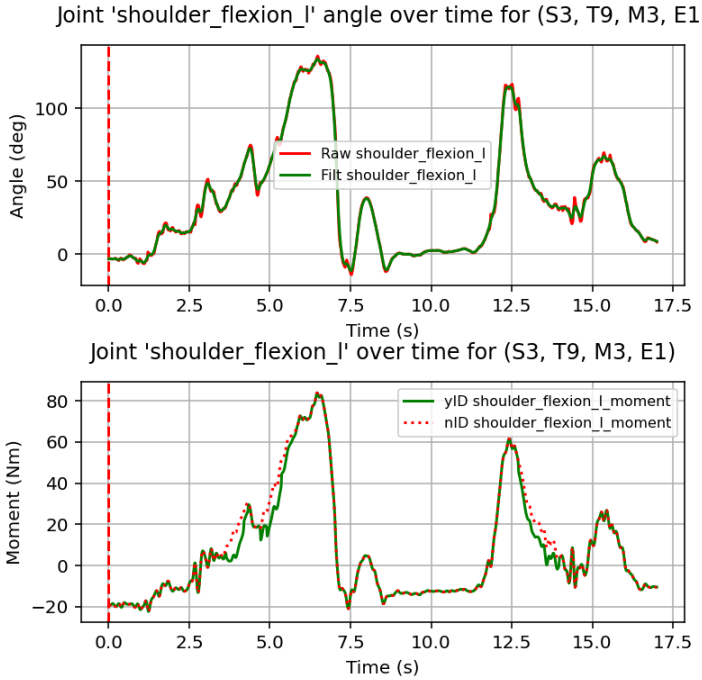
34.A: Lumbar extension.



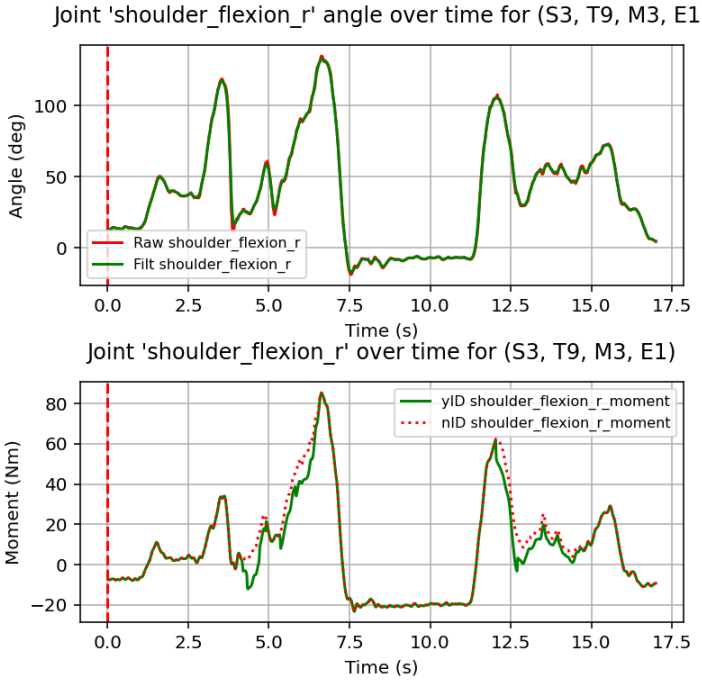
34.B: Lumbar bending.



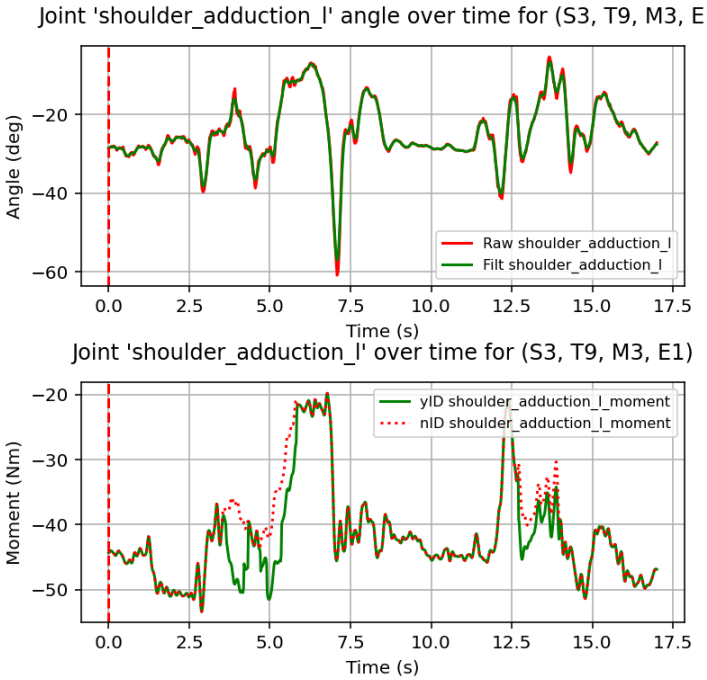
34.C: Lumbar rotation.



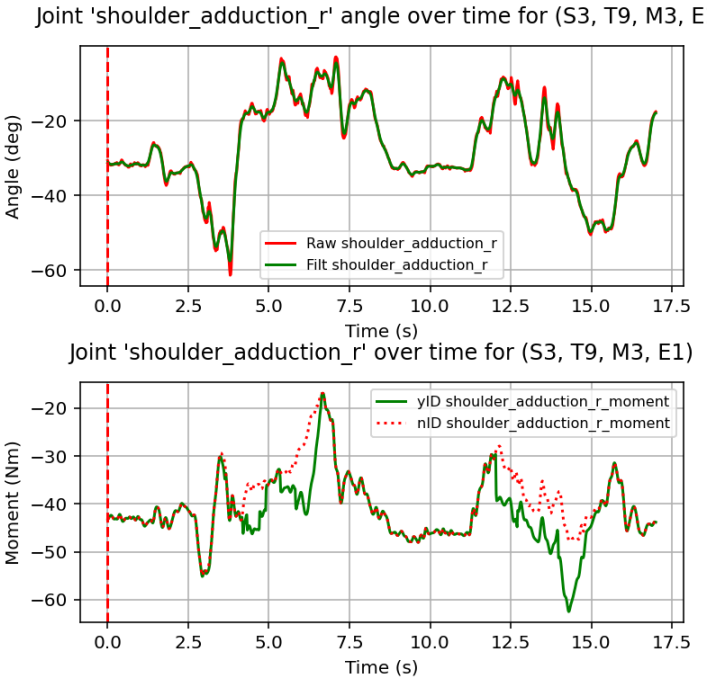
34.D: Shoulder flexion-extension of left arm.



34.E: Shoulder flexion-extension of right arm.



34.F: Shoulder adduction-abduction of left arm.



34.G: Shoulder adduction-abduction of right arm.

Fig. 34: Results from Inverse Dynamics for Subject 3, Trial 9. Each subfigure shows joint angle data (top) and corresponding moment comparisons (bottom). POT cycle events markings are also shown: N (neutral), BP (bend-pass), NW (neutral weighted), LP (lift-pass).

H. Arduino Code

```
#include <BMI160Gen.h>            //https://github.com/hanyazou/BMI160-Arduino
#include <Wire.h>

// I2C Configuration for ESP32
const int i2c_addr = 0x69;  // 0x69=3.3v connection  ,  0x68=GND connection

// Initialize
int ForceSens = A7; //analog pin 0
int F;
int gx, gy, gz; // Raw gyroscope values
int ax, ay, az; // Raw accelerometer values

const long interval = 8; // (ms) Interval at which to read sensor data, minimum of 7ms due to jankyness
int StartTime;

void setup() {
  // Initialize Serial communication at 115200 baud rate
  Serial.begin(115200);
  Serial.println("BMI160 initialization ...");

  while (!Serial); // Wait for Serial Monitor to connect (not required on ESP32)
  //Serial.println("....");

  // Initialize I2C with custom SDA and SCL pins
  Wire.begin();

  // Initialize the BMI160 device in I2C mode
  if (!BMI160.begin(BMI160GenClass::I2C_MODE, i2c_addr)) {
    Serial.println("BMI160 initialization failed!");
    while (1); // Halt if initialization fails
  }

  Serial.println("All set up. Type S in command to start reading, then P to stop");
}

void loop() {
if (Serial.available() > 0) {
    char command = Serial.read();

    if (command == 'S') {
      //Delays because arduino is jankywanky
      delay(200);
      //Set timer to start from 0
      StartTime = millis();
      delay(200);


      while (true) {
        unsigned long currentMillis = millis()-StartTime; // Get the current time

        // 100Hz: readout every 10 milliseconds
        if (currentMillis % interval == 0) {
          // Read your sensor values here
          F = analogRead(ForceSens); // Raw force reading

          BMI160.readAccelerometer(ax, ay, az);
          BMI160.readGyro(gx, gy, gz);


          // Send data to the serial port
          Serial.print(currentMillis); Serial.print(",");
          Serial.print(F); Serial.print(",");
          Serial.print(ax); Serial.print(",");
          Serial.print(ay); Serial.print(",");
          Serial.print(az); Serial.print(",");
          Serial.print(gx); Serial.print(",");
          Serial.print(gy); Serial.print(",");
          Serial.println(gz);

          // Check if stop command is received
        if (Serial.available() > 0 && Serial.read() == 'P') {
          break;
        }
      }
    }
  }
}
}


```

I. Validation code of UMocoD model

```

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d

#%% Load data

cut_LU_IK        =  pd.read_csv('cut_LU_IK.mot', sep='\t', skiprows=10).to_numpy()
cut_UMocoD_IK    =  pd.read_csv('cut_UMocoD_IK.mot', sep='\t', skiprows=10).to_numpy()
cut_LU_ID        =  pd.read_csv('cut_LU_ID.sto', sep='\t', skiprows=6).to_numpy()
cut_UMocoD_ID    =  pd.read_csv('cut_UMocoD_ID.sto', sep='\t', skiprows=6).to_numpy()

squats_LU_IK     =  pd.read_csv('squats_LU_IK.mot', sep='\t', skiprows=10).to_numpy()
squats_UMocoD_IK =  pd.read_csv('squats_UMocoD_IK.mot', sep='\t', skiprows=10).to_numpy()
squats_LU_ID     =  pd.read_csv('squats_LU_ID.sto', sep='\t', skiprows=6).to_numpy()
squats_UMocoD_ID =  pd.read_csv('squats_UMocoD_ID.sto', sep='\t', skiprows=6).to_numpy()


#%% functions for correlations


def GaussianFilter(data, COF):
    # Set up sigma value
    dt = data[1,0] - data[0,0]
    sigma = 0.132 / (COF * dt)

    # Initializing filtered array
    data_filt = data.copy()

    # Filter every column
    for i in range(1, data.shape[1]):
        data_filt[:, i] = gaussian_filter1d(data[:,i].astype(float), sigma=sigma)

    return data_filt

# Function for the IK cross-correlation
def normalized_cross_correlation_IK(a, b, COF):

    a = GaussianFilter(a, COF)
    a = GaussianFilter(b, COF)

    # Ignore time column
    a_cols = a[:, 1:]
    b_cols = b[:, 1:]

    # Between the models, some angles are opposite values. To correct this,
    # these angles are multiplied by -1.
    b_cols = b_cols*[1, 1, 1, 1, 1,
                     1, 1, 1, 1, 1,
                     -1,-1, 1,-1,-1,
                     1, 1, 1, 1, 1,
                     1, 1, 1, 1, 1,
                     1, 1, 1, 1, 1,
                     -1,-1, 1,-1,-1]

    corrs = []
    for i in range(a_cols.shape[1]):
        # Compute normalized cross-correlation for each column pair
        correlation = np.correlate(a_cols[:, i], b_cols[:, i], mode='full')
        norm_a = np.linalg.norm(a_cols[:, i])
        norm_b = np.linalg.norm(b_cols[:, i])

        if norm_a == 0 or norm_b == 0:
            continue  # Skip this column pair

        normalized_correlation = correlation / (norm_a * norm_b)
        corrs.append(normalized_correlation)

    max_correlations = [np.max(corr) for corr in corrs]

    avg = np.mean(max_correlations)
    std = np.std(max_correlations)
    print(f'{avg:.3f} +- {std:.3f}')

    return max_correlations, avg, std

def normalized_cross_correlation_ID(a, b):
    # Ignore time column
    a_cols = a[:, 1:]
    b_cols = b[:, 1:]

    # b_cols = b_cols*[1, 1, 1, 1, 1,
```

```
    #                   1, 1, 1, 1, 1,
    #                   1,1, 1,1,1,
    #                   1, 1, 1, 1, 1,
    #                   1, 1, 1, 1, 1,
    #                   1, 1, 1, 1, 1,
    #                   1,1, 1,1,1]

    corrs = []
    for i in range(a_cols.shape[1]):
        # Compute normalized cross-correlation for each column pair
        correlation = np.correlate(a_cols[:, i], b_cols[:, i], mode='full')
        norm_a = np.linalg.norm(a_cols[:, i])
        norm_b = np.linalg.norm(b_cols[:, i])

        if norm_a == 0 or norm_b == 0:
            continue  # Skip this column pair

        normalized_correlation = correlation / (norm_a * norm_b)
        corrs.append(normalized_correlation)

    max_correlations = [np.max(corr) for corr in corrs]

    avg = np.mean(max_correlations)
    std = np.std(max_correlations)
    print(f'{avg:.3f} +- {std:.3f}')

    return max_correlations, avg, std

#%% RUN

normalized_cross_correlation_IK(squats_LU_IK, squats_UMocoD_IK, 10)

#%%
normalized_cross_correlation_ID(squats_LU_ID, squats_UMocoD_ID)

#%%
normalized_cross_correlation_IK(cut_LU_IK, cut_UMocoD_IK, 20)

#%%
normalized_cross_correlation_ID(cut_LU_ID, cut_UMocoD_ID)

```

J. Pipeline Code

1) MAIN_Pipeline.py:

```
"""
Main Pipeline to easily run the code per step,
set up global data extraction,
or choose to specify which data needs processing.

Always run the below initialization, then each step can
be run independently
"""

#%% Pipeline initialization

import os, sys
import csv
from collections import defaultdict
import numpy as np

# Forces the pipeline to run from its location
PROJECT_ROOT = os.path.dirname(os.path.abspath(__file__))
if PROJECT_ROOT not in sys.path:
    sys.path.insert(0, PROJECT_ROOT)



from pipeline_scripts.DataExtracting_function import extract_data_from_files

# ================================================================================================================
#%% 0: Calibration Values
##  Needed values
#   F_C     Full compressive force on force sensor
#   F_T     Full tensile force on force sensor
#   m#      Total weight of each material
#   a_scale The conversion factor of the IMU


## OUTPUT
#   CalibValues.csv


# Adjust values inside definition to create calibValues.csv
def create_calibfile():
    F_C     = 486    #sensor readout at compressive load
    F_T     = 618    #sensor readout at tensile load
    a_scale = 16384     #sensor readout at acceleration=g
    m1      = 9.6        #kg
    m2      = 2.9        #kg
    m3      = 6.3        #kg
    m4      = 14.2       #kg


    data = {
    "Variable": ["F_C", "F_T", "a_scale", "m1", "m2", "m3", "m4"],
    "Value": [F_C, F_T, a_scale, m1, m2, m3, m4]
    }

    parent_dir = os.path.dirname(os.getcwd())
    CalibValues_file = os.path.join(parent_dir, "CalibValues.csv")


    with open(CalibValues_file, "w", newline="") as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(data["Variable"])  # Header row
        writer.writerow(data["Value"])     # Values row

    print('CalibValues created: ', CalibValues_file)

create_calibfile()



# ================================================================================================================
#%% 1: Record FIMU files
## OpenCap
#   OpenCap is a seperate entity, which cannot be included in the Pipeline.
#   From the downloaded data, the .trc file for the relevant trails,
#   and a mp4 video to be used for matching the shifting is needed

## OUTPUT OpenCap (manual renaming/placing)
#   rKIN_S#_T#_M#_E#.trc    Raw Kinematic tracer file (unfiltered)
#   VID_S#_T#_M#_E#.mp4     .mp4 file as reference video
#   Both placed inside Results\Subject#\Trial#\Material#\Ergonomic#\Raw


```

```
## ForceIMU sensor
#   Input into GUI:
#       Subject#
#            Trial# (1x up and down)
#            Material#
#            Ergo#

## OUTPUT
#   SD_S#.csv               Subject data, in the Results/Subject# folder
#   rFIF_S#_T#_M#_E#.csv    Raw ForceIMU-File, purely read from the arduino.

from pipeline_scripts.FIMU_Logger import ForceIMU_arduinoserial_logger
# Port characteristics
port = 'COM3'       # 'COM#'            COM Port used for arduino
baudrate = 115200   # Default:115200    Match your Arduino's baud rate

ForceIMU_arduinoserial_logger(port, baudrate)




# ================================================================================================================
#%% 2: Shift FIMU files to match kinematics, and change start-time to relevant start

## INPUT FILES
#   rKIN      .trc file to be used for inverse kinematics later
#   VID       .mp4 matching to trc file calibrate timings
#   rFIF      Raw ForceIMU-File

## OUTPUT FILES
#   KIN     .trc file that starts and stops at relevant parts
#   tVID    .mp4 trimmed to match the .trc times
#   FIF     ForceIMU-file which now matches the .trc times, and also starts/stop at same times



from pipeline_scripts.FIMU_TimeShifterStarter import run_marker_video_analysis

Step2_data = extract_data_from_files(
    # subjects=[1],  #range(1,100),
    # trials=[1], #range(1,100),
    # materials=[1,2,3,4],
    # ergonomics=[1,2],
    data_types = ['rKIN', 'VID', 'rFIF'],
)

# Group files by (subject, trial, material, ergonomic) to iterate over valid ones
file_groups2 = defaultdict(dict)
for key, filepath in Step2_data.items():
    subject, trial, material, ergonomic, data_type, file_format = key
    group_key = (subject, trial, material, ergonomic)
    file_groups2[group_key][data_type] = filepath

# Iterate over each group and run the function
for group_key, files in file_groups2.items():
    subject, trial, material, ergonomic = group_key
    if None in group_key:
        # Skip groups with None values (e.g., SubjectData)
        continue

    # Extract the required files
    raw_sensor_file = files.get('rFIF')
    raw_trc_file = files.get('rKIN')
    video_file = files.get('VID')

    # Check if all required files are present
    if raw_sensor_file and raw_trc_file and video_file:
        print(f"Running TimeShifter for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")
        run_marker_video_analysis(raw_sensor_file, raw_trc_file, video_file)
    else:
        print(f"Missing files for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")


# ================================================================================================================
#%% 3: Create Hand Grab File
# ================================================================================================================

## INPUT FILES
#   FIF     .csv Time adjusted sensor file
#   VID     .mp4 matching to trc file calibrate timings


## OUTPUT FILES
#   HGT     .csv Hand Grab Timings file


```

```
from pipeline_scripts.HGT_creator import HandGrab_creator


Step3_data = extract_data_from_files(
    #subjects=[2],  #range(1,100),
    #trials=[4], #range(1,100),
    # materials=[1,2,3,4],
    # ergonomics=[1,2],
    data_types = ['tVID', 'FIF'],
)

file_groups3 = defaultdict(dict)
for key, filepath in Step3_data.items():
    subject, trial, material, ergonomic, data_type, file_format = key
    group_key = (subject, trial, material, ergonomic)
    file_groups3[group_key][data_type] = filepath

# Iterate over each group and run the function
for group_key, files in file_groups3.items():
    subject, trial, material, ergonomic = group_key
    if None in group_key:
        # Skip groups with None values (e.g., SubjectData)
        continue
    # Extract the required files
    FIF_file = files.get('FIF')
    video_file = files.get('tVID')

    # Check if all required files are present
    if FIF_file and video_file:
        print(f"Running HandGrab Creator for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")
        HandGrab_creator(video_file, FIF_file)
    else:
        print(f"Missing files for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")



#%% 4: Create Scaled Model and Perform Inverse Kinematics

#
# IMPORTANT
# Running the first time per subject to create a scale model ends in an error.
# Its weird because the scaled model is correct.
# Run it again for it to loop through the IK (the model already exists, and
# running the scale tool again will ignore creating another scaled model)

## INPUT FILES
#   UMocuD_from_LU_editedMarkers.osim       Original model to be scaled per subject
#   SD                                      SubjectData
#   KIN                                     .trc motion file to be filtered

## OUTPUT FILES
#   OMS     OpenSim Model Scaled
#   IK      Inverse Kinematics (unfiltered)

from pipeline_scripts.Scale_and_IK import Scale_and_IK


#Load original model
current_dir = os.path.dirname(os.path.abspath(__file__))
original_model_file = os.path.join(current_dir, 'UMocuD_from_LU_editedMarkers.osim')


Step4_data = extract_data_from_files(
    #subjects=[3], #range(1,100),
    #trials=[1,2],
    # materials=[1,2,3,4],
    # ergonomics=[1,2],
    data_types = ['SubjectData', 'KIN'],
)

#pprint(Step3_data)

# Group files by (subject, trial, material, ergonomic) to iterate over valid ones
file_groups4 = defaultdict(dict)
for key, filepath in Step4_data.items():
    subject, trial, material, ergonomic, data_type, file_format = key
    group_key = (subject, trial, material, ergonomic)
    file_groups4[group_key][data_type] = filepath

# Iterate over each group and run the function
for group_key, files in file_groups4.items():
    subject, trial, material, ergonomic = group_key

    # Extract the required files
    trc_file = files.get('KIN')

```

```
    #Seperate for Subject
    subj_group = (subject, None, None, None)
    SubjectData_file = file_groups4.get(subj_group, {}).get('SubjectData')

    # Check if all required files are present
    if SubjectData_file and trc_file:
        print(f"Running Scaler and IK for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")
        Scale_and_IK(original_model_file, trc_file, SubjectData_file)
    else:
        print(f"Missing files for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")

#Kinematic_filter(original_model, trc_file, SubjectData_file)

# ============================================================================================================
#%% 5: Determine Cutoff Frequency
## INPUT
#   FIF     .csv
#   IK      .mot ()

## OUTPUT
#   (no files)
#   print out average and standard deviations of all trials

# Determine_cutoff_freqs output as:
#   IK_cof, F_cof, IMU_cof
from pipeline_scripts.COF_analysis import Determine_cutoff_freqs


Step5_data = extract_data_from_files(
    #subjects=[1], #range(1,100),
    #trials=[1,2],
    # materials=[1,2,3,4],
    # ergonomics=[1,2],
    data_types = ['IK', 'FIF'],
)

file_groups5 = defaultdict(dict)
for key, filepath in Step5_data.items():
    subject, trial, material, ergonomic, data_type, file_format = key
    group_key = (subject, trial, material, ergonomic)
    file_groups5[group_key][data_type] = filepath

# Create arrays of all COFs for each trial
all_IK_cof  = []
all_F_cof   = []
all_IMU_cof = []


# Iterate over each group and run the function
for group_key, files in file_groups5.items():
    subject, trial, material, ergonomic = group_key
    if None in group_key:
        # Skip groups with None values (e.g., SubjectData)
        continue
    # Extract the required files
    FIF_file = files.get('FIF')
    IK_file = files.get('IK')

    # Check if all required files are present
    if FIF_file and IK_file:
        print(f"Running cutoff analysis for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")
        IK_cof, F_cof, IMU_cof = Determine_cutoff_freqs(FIF_file, IK_file)
        all_IK_cof.append(IK_cof)
        all_F_cof.append(F_cof)
        all_IMU_cof.append(IMU_cof)

    else:
        print(f"Missing files for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")

mean_IK_cof = np.mean(all_IK_cof)
mean_F_cof = np.mean(all_F_cof)
mean_IMU_cof = np.mean(all_IMU_cof)
std_IK_cof = np.std(all_IK_cof)
std_F_cof = np.std(all_F_cof)
std_IMU_cof = np.std(all_IMU_cof)

print(f'IK_cof = {mean_IK_cof:.3f} +- {std_IK_cof:.3f}')
print(f'F_cof = {mean_F_cof:.3f} +- {std_F_cof:.3f}')
print(f'IMU_cof = {mean_IMU_cof:.3f} +- {std_IMU_cof:.3f}')

# If F and IMU cof's are >> IK
#   Choose a cof that is about 20% higher than IK_cof


# ============================================================================================================
#%% 6: Filter Data
```

```
## INPUT
#   COF     Cutoff frequency
#   FIF     Time adjusted sensor file
#   IK      Inverse Kinematics File
"""
Set cutoff frequency for all subject/trials here
"""
COF = 2.8

## OUTPUT (into 'Processed Data' folder)
# fFIF      Filtered time-adjusted sensor file
# fIK       Filtered time-adjusted Inverse Kinematics File

from pipeline_scripts.IK_FIMU_Filter import filter_data

Step6_data = extract_data_from_files(
    #subjects=[1], #range(1,100),
    #trials=[1,2],
    # materials=[1,2,3,4],
    # ergonomics=[1,2],
    data_types = ['IK', 'FIF'],
)

file_groups6 = defaultdict(dict)
for key, filepath in Step6_data.items():
    subject, trial, material, ergonomic, data_type, file_format = key
    group_key = (subject, trial, material, ergonomic)
    file_groups6[group_key][data_type] = filepath

for group_key, files in file_groups6.items():
    subject, trial, material, ergonomic = group_key
    if None in group_key:
        # Skip groups with None values (e.g., SubjectData)
        continue
    # Extract the required files
    FIF_file = files.get('FIF')
    IK_file = files.get('IK')

    # Check if all required files are present
    if FIF_file and IK_file:
        print(f"Running filtering for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")
        filter_data(COF, IK_file, FIF_file)

    else:
        print(f"Missing files for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")


# ============================================================================================================
#%% 7: Create External Force File
## INPUT
#   fFIF            filtered ForceIMU file
#   HGT             HandGrabTimes file
#   XMl_template    Base XML file for applying forces on hands
#   CalibValues     Calibrated sensor values

## OUTPUT
#   EFF             External Force File, containing linear force data for left and right hand
#   XML             XML file corresponding to EFF file, with data used for opensim running


from pipeline_scripts.EFF_creator import FIMU_to_EFF

# Plot the HGT, mF, and Fv functions to check
plot_HGt_mF_Fv = False


#Use the project_root to create
XML_base_file = os.path.join(PROJECT_ROOT, 'EFF_xml_template.xml')
CalibValues = os.path.join(PROJECT_ROOT, 'CalibValues.csv')

Step7_data = extract_data_from_files(
    #subjects=[1],
    #trials=[1,2],
    # materials=[1,2,3,4],
    # ergonomics=[1,2],
    data_types = [r'fFIF', 'HGT'],
)
from pprint import pprint
pprint(Step7_data)


file_groups7 = defaultdict(dict)
for key, filepath in Step7_data.items():
    subject, trial, material, ergonomic, data_type, file_format = key
    group_key = (subject, trial, material, ergonomic)
    file_groups7[group_key][data_type] = filepath
```

```
428
429 for group_key, files in file_groups7.items():
430     subject, trial, material, ergonomic = group_key
431
432     if None in group_key:
433         # Skip groups with None values (e.g., SubjectData)
434         continue
435
436     # Extract the required files
437     fFIF_file = files.get(r'fFIF')
438     HGT_file  = files.get('HGT')
439
440     # Check if all required files are present
441     if fFIF_file and HGT_file:
442         print(f"Running EFF creator for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")
443         FIMU_to_EFF(fFIF_file, HGT_file, XML_base_file, CalibValues, plot=plot_HGt_mF_Fv)
444     else:
445         print(f"Missing files for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")
446
447
448
449 # ============================================================================================================
450 #%% 8: Perform Inverse Dynamics
451 ## INPUT
452 #   fIK             Filtered Inverse Kinematics file
453 #   ScaledModel     OpenSim model corresponding to the subject
454 #   XML             XML file corresponding to EFF file, with data used for opensim running
455
456
457 ## OUTPUT
458 #   yID             Inverse Dynamics -with- EFF applied
459 #   nID             Inverse Dynamics -without- EFF applied
460
461 from pipeline_scripts.InverseDynamics import PerformID
462
463 Step8_data = extract_data_from_files(
464     #subjects=[1], #range(1,100),
465     #trials=[1,2],
466     # materials=[1,2,3,4],
467     # ergonomics=[1,2],
468     data_types = ['fIK', 'ScaledModel','XML'],
469 )
470
471 from pprint import pprint
472 pprint(Step8_data)
473
474 file_groups8 = defaultdict(dict)
475 for key, filepath in Step8_data.items():
476     subject, trial, material, ergonomic, data_type, file_format = key
477     group_key = (subject, trial, material, ergonomic)
478     file_groups8[group_key][data_type] = filepath
479
480 for group_key, files in file_groups8.items():
481     subject, trial, material, ergonomic = group_key
482
483     # Extract the required files
484     fIK_file  = files.get(r'fIK')
485     XML_file  = files.get('XML')
486
487     # Seperate for ScaledModel because SM_file has Nones
488     subj_group = (subject, None, None, None)
489     SM_file = file_groups8.get(subj_group, {}).get('ScaledModel')
490
491
492     # Check if all required files are present
493     if fIK_file and SM_file and XML_file:
494         print(f"Running Inverse Dynamics for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")
495         PerformID(SM_file, fIK_file, XML_file)
496     else:
497         print(f"Missing files for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")
498
499 # ============================================================================================================
500 #%% 9: Perform Static Optimization
501 ## INPUT
502 #   fIK             Filtered Inverse Kinematics file
503 #   ScaledModel     OpenSim model corresponding to the subject
504 #   XML             XML file corresponding to EFF file, with data used for opensim running
505
506
507 ## OUTPUT
508 #   ySO (folder)    Static Optimization -with- EFF applied
509 #   nSO (folder)    Static Optimization -without- EFF applied
510
511 from pipeline_scripts.StaticOptimization import PerformSO
512
513 Step9_data = extract_data_from_files(
```

```
    subjects=[3], #range(1,100),
    trials=[5,6,7,8,9],
    # materials=[1,2,3,4],
    # ergonomics=[1,2],
    data_types = ['fIK', 'XML', 'ScaledModel'],
)

from pprint import pprint
pprint(Step9_data)

file_groups9 = defaultdict(dict)
for key, filepath in Step9_data.items():
    subject, trial, material, ergonomic, data_type, file_format = key
    group_key = (subject, trial, material, ergonomic)
    file_groups9[group_key][data_type] = filepath

for group_key, files in file_groups9.items():
    subject, trial, material, ergonomic = group_key

    # Extract the required files
    fIK_file  = files.get(r'fIK')
    EFF_XML_file  = files.get('XML')

    # Seperate for ScaledModel because SM_file has Nones
    subj_group = (subject, None, None, None)
    SM_file = file_groups9.get(subj_group, {}).get('ScaledModel')


    # Check if all required files are present
    if fIK_file and EFF_XML_file and SM_file:
        print(f"Running Static Optimization for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")
        PerformSO(fIK_file, EFF_XML_file, SM_file)
    else:
        print(f"Missing files for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")
# ============================================================================================================

#%% 10: Plot Results
## INPUT
#   VID
#   fIK
#   ySO (folder)    Static Optimization -with- EFF applied
#   nSO (folder)    Static Optimization -without- EFF applied
#   yID             Inverse Dynamics -with- EFF applied
#   nID             Inverse Dynamics -without- EFF applied

open Plot_IK_ID_SO


# ============================================================================================================
```

---

2) DataExtracting_function.py:

```
"""
Use the code below inside of the normal scripts to extract the specific data needed:



from data_extractor import extract_and_save_data

# Example
extract_and_save_data(
    subjects=range(1,11),
    trials=range(1,31),
    materials=[1,2,3,4],
    ergonomics=[1,2],
    data_types = ['rKIN', 'rFIF'],
    base_dir=".."
)

"""
#%%
from pathlib import Path
import re

PROJECT_ROOT = Path(__file__).resolve().parent.parent  # up from pipeline_scripts

# Extract the numbers of the tags
_S = re.compile(r'(?:^|_)S(\d+)(?:_|$)', re.I)
_T = re.compile(r'(?:^|_)T(\d+)(?:_|$)', re.I)
_M = re.compile(r'(?:^|_)M(\d+)(?:_|$)', re.I)
_E = re.compile(r'(?:^|_)E(\d+)(?:_|$)', re.I)


def _find_int(rx, text):
    m = rx.search(text)
    return int(m.group(1)) if m else None

def extract_data_from_files(
    subjects=range(1, 100),
    trials=range(1, 100),
    materials=None,
    ergonomics=None,
    data_types=('rKIN','VID','tVID','rFIF','FIF','fFIF','SubjectData',
                'ScaledModel','KIN','EFF','XML','OMS','IK','fIK','yID',
                'nID','ySO','nSO'),
    base_dir=None,
    result_dir="Results",
):
    """
    key: (subject, trial, material, ergonomic, data_type, file_ext)
    val: absolute path
    - material/ergonomic are parsed from filename (_M#/_E#). If missing  None.
    - If 'materials' or 'ergonomics' lists are provided, only keep matches.

    rKIN            .trc Raw OpenCap kinematics data
    KIN             .trc Trimmed kinematics data
    VID             .mp4 Full OpenCap reference video
    tVID            .mp4 Trimmed reference video
    rFIF            .csv Raw ForceIMU data from arduino
    FIF             .csv Trimmed ForceIMU data, synced with KIN
    fFIF            .csv Filtered ForceIMU data
    SubjectData     .csv Height and Weight of Subject to be used in scaling model
    ScaledModel     .osim Scaled OpenSim model for subject
    EFF             .mot ExternalForceFile, with forces set on hands of subject
    XML             .xml EFF xml file, which is needed to run the EFF
    IK              .mot Inverse Kinematics file, based on KIN
    fIK             .mot Filtered IK file
    yID             .mot Inverse Dynamics based on fIK, with EFF applied
    nID             .mot Inverse Dynamics based on fIK, without EFF applied
    ySO             .mot Static Optimization based on fIK, with EFF applied
    nSO             .mot Static Optimization based on fIK, without EFF applied

    """
    base = Path(base_dir) if base_dir else PROJECT_ROOT
    results_path = (base / result_dir).resolve()
    if not results_path.exists():
        raise FileNotFoundError(f"Results path not found: {results_path}")

    data_types = set(data_types)
    mat_filter = set(materials) if materials else None
    ergo_filter = set(ergonomics) if ergonomics else None

    extracted = {}

    # Top-level Results files (e.g., CalibValues.csv)
```

```
    for f in results_path.iterdir():
        if f.is_file():
            ext = f.suffix.lower()
            key = (None, None, None, None, 'CalibValues', ext)
            extracted[key] = str(f)

    for subject in subjects:
        subject_dir = results_path / f"Subject{subject}"
        if not subject_dir.exists():
            continue

        # Files directly in Subject folder (SubjectData and ScaledModel)
        for f in subject_dir.iterdir():
            if f.is_file():
                ext = f.suffix.lower()
                prefix = f.name.split('_')[0]
                if prefix in data_types:  # eg SubjectData or ScaledModel
                    key = (subject, None, None, None, prefix, ext)
                    extracted[key] = str(f)

        for trial in trials:
            trial_dir = subject_dir / f"Trial{trial}"
            if not trial_dir.exists():
                continue

            for subfolder in ("Raw", "Processed_Data"):
                sub_dir = trial_dir / subfolder
                if not sub_dir.exists():
                    continue

                for f in sub_dir.iterdir():
                    if not f.is_file():
                        continue

                    prefix = f.name.split('_')[0]  # data_type

                    if prefix not in data_types:
                        continue

                    name_no_ext = f.stem  # e.g., VID_S69_T1_M1_E1
                    # Parse S/T/M/E from filename (fallback only; loop indices win)
                    S = _find_int(_S, name_no_ext)
                    T = _find_int(_T, name_no_ext)
                    M = _find_int(_M, name_no_ext)
                    E = _find_int(_E, name_no_ext)

                    subj_key = subject if subject is not None else S
                    trial_key = trial if trial is not None else T
                    mat_key = M
                    ergo_key = E

                    # Optional filtering by material/ergonomic values
                    if mat_filter is not None and mat_key not in mat_filter:
                        continue
                    if ergo_filter is not None and ergo_key not in ergo_filter:
                        continue

                    key = (subj_key, trial_key, mat_key, ergo_key, prefix, f.suffix.lower())
                    extracted[key] = str(f)


                # Process ySO/nSO folders
                for so_type in ('ySO', 'nSO'):
                    for so_dir in sub_dir.iterdir():
                        if so_dir.is_dir() and so_dir.name.startswith(so_type):

                            name_no_ext = so_dir.name
                            S = _find_int(_S, name_no_ext)
                            T = _find_int(_T, name_no_ext)
                            M = _find_int(_M, name_no_ext)
                            E = _find_int(_E, name_no_ext)

                            subj_key = subject if subject is not None else S
                            trial_key = trial if trial is not None else T
                            mat_key = M
                            ergo_key = E

                            if mat_filter is not None and mat_key not in mat_filter:
                                continue
                            if ergo_filter is not None and ergo_key not in ergo_filter:
                                continue

                            sto_file = so_dir / '_StaticOptimization_activation.sto'

                            if sto_file.exists():
                                key = (subj_key, trial_key, mat_key, ergo_key, so_type, '.sto')
```

```
171                                 extracted[key] = str(sto_file)
172
173
174
175         return extracted
176
177
```

3) 1) FIMU_Logger:

```
"""
Force-IMU Data Logger:

Script reads out serial port. Arduino script sends out prints in x,y,z,...
format.

Uses a GUI to fill in experiment details (Subject#, Trial#, Material#,
                                          Ergonomic#, Height, Weight)
Subject#:    Subject number
Trial#:      Trial number
Material#:   Material number (1:Stander, 2:Layer, 3:Console)
Ergonomic#:  1: Material passed through the outside of scaffold entirely
             2: Material passed through the outside to the one above,
                but the subject brings in the material before passing it up

Height in cm, Weight in kg

Force-IMU data Exports into
OpenCap_OpenSIM_PIPELINE/Subject#/Trial#/Raw folders
for ease of use in the data processing pipeline
Subject data exports into Subject# folder
"""


#%%
import serial
import csv
import threading
import time
import tkinter as tk
from tkinter import messagebox
import os
import subprocess
import sys


def ForceIMU_arduinoserial_logger(port='COM1', baudrate=115200):
    """
    Inputs the port used to start the serial connection,
    and baudrate (match this with arduino IDE code)

    Logs the serial data from an arduino. This corresponds with the Arduino
    IDE 'FIMU_SerialPrint.ino'.

    Then saves this data into a new chain folder, corresponding with subject
    number and trial number.

    """

    # Initialization, saves the time, forces, and linear and rotational acceleration
    csv_header = ['time', 'F', 'ax', 'ay', 'az', 'gx', 'gy', 'gz']
    ser = serial.Serial(port, baudrate, timeout=1)
    running = False
    data_log = []

    # Reads out the serial line, that the arduino is printing
    # And logs this data into data.csv
    def read_serial():
        nonlocal running
        while True:
            if running and ser.in_waiting > 0:
                try:
                    line = ser.readline().decode('utf-8').strip()
                    if line:
                        # Split the CSV line directly
                        row = line.split(',')
                        # Convert all elements to int (or float if needed)
                        row = [int(x) if x.lstrip('-').isdigit() else float(x) for x in row]
                        data_log.append(row)
                except Exception as e:
                    print(f"Error reading line: {e}")
            time.sleep(0.005)

    def update_filename_preview(*args):
        subject_str = subject_entry.get().strip()
        trial_num = trial_var.get()
        material_num = material_entry.get().strip()
        ergonomic_num = ergonomic_entry.get().strip()

        #preview of filenames
        if subject_str and material_num and ergonomic_num:
            fname_data = f"rFIF_S{subject_str}_T{trial_num}_M{material_num}_E{ergonomic_num}.csv"
            fname_subject = f"SD_S{subject_str}.csv"
```

```
            filename_preview.set(f"Data: {fname_data}\nSubject: {fname_subject}")
        else:
            filename_preview.set("Enter all fields to see filenames")

    # Starts the logging of
    def start_logging():
        nonlocal running, data_log
        if not running:
            fname_data = filename_preview.get().split("\n")[0].split(": ")[1]
            if os.path.exists(os.path.join("..", "Results", fname_data)):
                messagebox.showerror("File Exists", f"The file '{fname_data}' already exists.\n"
                                                    "Please change the subject, trial, material, or ergonomic number.")
                return
            data_log = []
            data_log.append(csv_header)

            #.ino program expects 'S' to start the serialprint
            ser.write(b'S')
            running = True
            start_button.config(state='disabled')
            quit_button.config(state='disabled')
            status_var.set("Running")
            print("Started logging...")

            # Disable editing during logging
            subject_entry.config(state='disabled')
            trial_entry.config(state='disabled')
            material_entry.config(state='disabled')
            ergonomic_entry.config(state='disabled')
            height_entry.config(state='disabled')
            weight_entry.config(state='disabled')

    # Function to have the data always start at t=o
    def normalize_time(data):
        if not data or len(data) <= 1:
            return data
        # The first row is the header, so the first data row is data[1]
        first_time = data[1][0]  # Get the first time value
        for i in range(1, len(data)):
            data[i][0] -= first_time  # Subtract the first time from all time values
        return data

    def stop_logging():
        nonlocal running, data_log
        if running:

            #.ino program expects 'P' to stop the serialprint
            ser.write(b'P')
            running = False
            start_button.config(state='normal')
            quit_button.config(state='normal')
            status_var.set("Not running")
            print("Stopped logging. Saving CSV...")

            # Get all input values
            subject_str     = subject_entry.get().strip()
            trial_num       = trial_var.get()
            material_num    = material_entry.get().strip()
            ergonomic_num   = ergonomic_entry.get().strip()
            height_cm       = height_entry.get().strip()
            weight_kg       = weight_entry.get().strip()

            # Create directory structure: Results/Subject#/
            base_dir    = os.path.join("..", "Results")
            subject_dir = os.path.join(base_dir, f"Subject{subject_str}")
            os.makedirs(subject_dir, exist_ok=True)

            # Save data file: rFIF_S#_T#_M#_E#.csv (in Subject#/Trial#/Raw folder)
            trial_dir       = os.path.join(subject_dir, f"Trial{trial_num}")
            raw_dir         = os.path.join(trial_dir, "Raw")
            os.makedirs(raw_dir, exist_ok=True)
            data_file = os.path.join(raw_dir, f"rFIF_S{subject_str}_T{trial_num}_M{material_num}_E{ergonomic_num}.csv")

            try:
                # Set all the forces to 0 if the material is not stander
                # The other materials dont use the force sensor
                if material_num != '1':
                    for i in range(1, len(data_log)):
                        data_log[i][1] = 0  # Force column is index 1

                with open(data_file, 'w', newline='') as f:
                    writer = csv.writer(f)
                    writer.writerows(data_log)
                messagebox.showinfo("Data Saved", f"Data saved to {data_file}")
            except Exception as e:
                messagebox.showerror("Error", f"Failed to save data: {e}")
```

```
171
172             # Save subject data file: SD_S#.csv (in Subject# folder)
173             subject_data = [["height_cm", "weight_kg"], [height_cm, weight_kg]]
174             subject_file = os.path.join(subject_dir, f"SubjectData_S{subject_str}.csv")
175             try:
176                 with open(subject_file, 'w', newline='') as f:
177                     writer = csv.writer(f)
178                     writer.writerows(subject_data)
179                 messagebox.showinfo("Subject Data Saved", f"Subject data saved to {subject_file}")
180             except Exception as e:
181                 messagebox.showerror("Error", f"Failed to save subject data: {e}")
182
183             # Auto-increment trial number
184             trial_var.set(trial_num + 1)
185
186             # Re-enable editing after logging stops
187             subject_entry.config(state='normal')
188             trial_entry.config(state='normal')
189             material_entry.config(state='normal')
190             ergonomic_entry.config(state='normal')
191             height_entry.config(state='normal')
192             weight_entry.config(state='normal')
193
194             # Update filename preview with new trial number
195             update_filename_preview()
196
197
198
199     def quit_program():
200         status_var.set("Quitting")
201         root.update()
202         ser.close()
203         root.destroy()
204
205
206     def open_and_highlight_file(filepath):
207         folder = os.path.dirname(os.path.abspath(filepath))
208         try:
209             if os.name == 'nt':  # Windows
210                 subprocess.Popen(f'explorer /select,"{filepath}"')
211             elif sys.platform == 'darwin':  # macOS
212                 subprocess.Popen(['open', '-R', filepath])
213             else:  # Linux
214                 subprocess.Popen(['xdg-open', folder])
215         except Exception as e:
216             messagebox.showerror("Error", f"Could not open folder: {e}")
217
218
219     #%% GUI SETUP
220     root = tk.Tk()
221     root.title("Arduino Serial Logger")
222     root.protocol("WM_DELETE_WINDOW", quit_program)
223
224     # Status label
225     status_var = tk.StringVar(value="Not running")
226     status_label = tk.Label(root, textvariable=status_var, font=("Arial", 14))
227     status_label.pack(pady=10)
228
229     # Subject field
230     subject_frame = tk.Frame(root)
231     subject_frame.pack(pady=5)
232     tk.Label(subject_frame, text="Subject #:").pack(side=tk.LEFT, padx=5)
233     subject_entry = tk.Entry(subject_frame, width=15)
234     subject_entry.pack(side=tk.LEFT)
235     subject_entry.insert(0, "")
236
237     # Trial number field
238     trial_frame = tk.Frame(root)
239     trial_frame.pack(pady=5)
240     tk.Label(trial_frame, text="Trial #:").pack(side=tk.LEFT, padx=5)
241     trial_var = tk.IntVar(value=1)
242     trial_entry = tk.Entry(trial_frame, textvariable=trial_var, width=5)
243     trial_entry.pack(side=tk.LEFT)
244
245     # Material# field
246     material_frame = tk.Frame(root)
247     material_frame.pack(pady=5)
248     tk.Label(material_frame, text="Material #:").pack(side=tk.LEFT, padx=5)
249     material_entry = tk.Entry(material_frame, width=15)
250     material_entry.pack(side=tk.LEFT)
251     material_entry.insert(0, "")
252
253     # Ergonomic# field
254     ergonomic_frame = tk.Frame(root)
255     ergonomic_frame.pack(pady=5)
256     tk.Label(ergonomic_frame, text="Ergonomic #:").pack(side=tk.LEFT, padx=5)
```

```
    ergonomic_entry = tk.Entry(ergonomic_frame, width=15)
    ergonomic_entry.pack(side=tk.LEFT)
    ergonomic_entry.insert(0, "")

    # Subject Height (cm) field
    height_frame = tk.Frame(root)
    height_frame.pack(pady=5)
    tk.Label(height_frame, text="Subject Height (cm):").pack(side=tk.LEFT, padx=5)
    height_entry = tk.Entry(height_frame, width=15)
    height_entry.pack(side=tk.LEFT)
    height_entry.insert(0, "")

    # Subject Weight (kg) field
    weight_frame = tk.Frame(root)
    weight_frame.pack(pady=5)
    tk.Label(weight_frame, text="Subject Weight (kg):").pack(side=tk.LEFT, padx=5)
    weight_entry = tk.Entry(weight_frame, width=15)
    weight_entry.pack(side=tk.LEFT)
    weight_entry.insert(0, "")


    # Filename preview label (changes color if file exists)
    filename_preview = tk.StringVar(value="")
    filename_label = tk.Label(root, textvariable=filename_preview, font=("Arial", 10))
    filename_label.pack(pady=5)

    # Update Preview Button
    update_button = tk.Button(root, text="Update Preview", command=update_filename_preview, width=15)
    update_button.pack(pady=5)

    # Bind updates to preview on user input
    subject_entry.bind("<KeyRelease>", update_filename_preview)
    trial_var.trace_add("write", update_filename_preview)

    # Initialize filename preview text
    update_filename_preview()

    # Buttons
    start_button = tk.Button(root, text="Start", command=start_logging, width=10)
    start_button.pack(pady=5)

    stop_button = tk.Button(root, text="Stop", command=stop_logging, width=10)
    stop_button.pack(pady=5)

    quit_button = tk.Button(root, text="Quit", command=quit_program, width=10)
    quit_button.pack(pady=5)

    # === THREAD FOR READING SERIAL ===
    threading.Thread(target=read_serial, daemon=True).start()

    root.mainloop()


```

4) 2) FIMU_TimeShifterStarter:

```
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
from tkinter import ttk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import cv2
from PIL import Image, ImageTk
import numpy as np
import os
import moviepy as mp

def run_marker_video_analysis(sensor_file, trc_file, video_file):
    """
    Inputs raw sensor and tracer data, and videocapture.

    Function to run a GUI. In this GUI, the video_file is shown, with a red
    axis showing the current video time on sensor file variables.

    Use the shifts to shift the sensor data onto the correct moment in
    the video. When filming, create some sort of predictable disturbance.

    In this thesis, the scaffold material is tapped 3 times against the
    scaffold in view of the video. These can be seen as three spikes in the
    acceleration data.

    When the shifting is done, the start-stop time can be adjusted.

    These all result in new sensor, trc, and video files corresponding to the
    new timeframe.

    """

    # Load data
    sf_data = pd.read_csv(sensor_file, sep=',').to_numpy()

    # Start times can be weird, skip first five rows, and set the starttime to 0
    # And transform to seconds
    t_skip = sf_data[4:,0]
    ts = (t_skip - t_skip[0]) / 1000

    # Load variables, similar to time skipping first few rows
    Fs, axs, ays, azs, gxs, gys, gzs = [sf_data[4:, i] for i in range(1, 8)]

    # Initialize video capture
    cap = cv2.VideoCapture(video_file)
    if not cap.isOpened():
        raise ValueError("Could not open video file")

    # Video properties
    fps = cap.get(cv2.CAP_PROP_FPS)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    duration = total_frames / fps

    # Create the main window
    root = tk.Tk()
    root.title(os.path.basename(sensor_file))

    # Create a frame for the video and plot
    frame = ttk.Frame(root)
    frame.pack(fill=tk.BOTH, expand=True)

    # Video display
    video_label = ttk.Label(frame)
    video_label.pack(side=tk.LEFT, padx=6, pady=3)

    # Matplotlib figure with two subplots
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(5, 3))

    # Default xlim
    xlim_vals = [0, 40]

    # Initial main plot
    ax1.plot(ts, azs, label='Acceleration')
    ax1.set_xlabel('Time')
    ax1.set_ylabel('Raw Acceleration Unit')
    ax1.legend()
    ax1.grid(True)
    ax1.set_xlim(xlim_vals)

    # Initial diff plot
    ax2.plot(ts, Fs, label='Force Sensor data')
    ax2.set_xlabel('Time')
    ax2.set_ylabel('Raw Force Unit')
```

```
    ax2.legend()
    ax2.grid(True)
    ax2.set_xlim(xlim_vals)

    # Embed the plot in Tkinter
    canvas = FigureCanvasTkAgg(fig, master=frame)
    canvas.draw()
    canvas.get_tk_widget().pack(side=tk.RIGHT, padx=10, pady=10)

    # Function to update plots when an action is done
    def update_plots(*args, t_val=None, start_time=None, end_time=None):
        ax1.clear()
        ax1.plot(ts, azs, label='Acceleration')
        ax1.set_xlabel('Time')
        ax1.set_ylabel('Raw Acceleration Unit')
        ax1.legend()
        ax1.grid(True)
        ax1.set_xlim(xlim_vals)

        ax2.clear()
        ax2.plot(ts, Fs, label='Force Sensor data')
        ax2.set_xlabel('Time')
        ax2.set_ylabel('Raw Force Unit')
        ax2.legend()
        ax2.grid(True)
        ax2.set_xlim(xlim_vals)

        # Add vertical lines for start and end times
        if start_time is not None:
            ax1.axvline(x=start_time, color='red', linestyle=':', linewidth=2)
            ax2.axvline(x=start_time, color='red', linestyle=':', linewidth=2)
        if end_time is not None:
            ax1.axvline(x=end_time, color='red', linestyle=':', linewidth=2)
            ax2.axvline(x=end_time, color='red', linestyle=':', linewidth=2)

        # Add vertical line for current time
        if t_val is not None:
            ax1.axvline(x=t_val, color='r', linestyle='--')
            ax2.axvline(x=t_val, color='r', linestyle='--')

        # Gray out areas outside the selected range
        if start_time is not None and end_time is not None:
            for ax in [ax1, ax2]:
                ax.axvspan(ts[0], start_time, color='lightgray', alpha=0.5)
                ax.axvspan(end_time, ts[-1], color='lightgray', alpha=0.5)

        canvas.draw()

    # Time selection box
    time_frame = ttk.Frame(root)
    time_frame.pack(pady=10)
    tk.Label(time_frame, text="Set Time (s):").pack(side=tk.LEFT, padx=5)
    time_entry = ttk.Entry(time_frame, width=10)
    time_entry.pack(side=tk.LEFT, padx=5)

    # Function to update video frame and plots based on set time
    def update_time(t_val=None):
        if t_val is None:
            try:
                t_val = float(time_entry.get())
            except ValueError:
                return
        else:
            time_entry.delete(0, tk.END)
            time_entry.insert(0, f"{t_val:.3f}")

        # Compute frame number
        frame_number = int(t_val * fps)
        frame_number = max(0, min(total_frames - 1, frame_number))
        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_number)
        ret, cv_frame = cap.read()
        if ret:
            cv_frame = cv2.cvtColor(cv_frame, cv2.COLOR_BGR2RGB)
            cv_frame = cv2.resize(cv_frame, (640, 480))
            img = Image.fromarray(cv_frame)
            imgtk = ImageTk.PhotoImage(image=img)
            video_label.imgtk = imgtk
            video_label.configure(image=imgtk)
        update_plots(t_val=t_val)

    # Button to update time
    set_time_button = ttk.Button(time_frame, text="Set", command=lambda: update_time())
    set_time_button.pack(side=tk.LEFT, padx=5)

    # Time slider
    time_slider = ttk.Scale(
```

```
        root, from_=0, to=duration, orient=tk.HORIZONTAL, length=400,
        command=lambda val: update_time(float(val))
    )
    time_slider.pack(pady=10)

    # Frame step buttons
    def step_frame(delta):
        try:
            current_time = float(time_entry.get())
        except ValueError:
            current_time = 0.0
        new_time = current_time + delta / fps
        new_time = max(0, min(duration, new_time))
        time_slider.set(new_time)
        update_time(new_time)

    button_frame = ttk.Frame(root)
    button_frame.pack(pady=5)
    prev_button = ttk.Button(button_frame, text="<< Prev Frame", command=lambda: step_frame(-1))
    prev_button.pack(side=tk.LEFT, padx=5)
    next_button = ttk.Button(button_frame, text="Next Frame >>", command=lambda: step_frame(1))
    next_button.pack(side=tk.LEFT, padx=5)

    # Xlim controls
    xlim_frame = ttk.Frame(root)
    xlim_frame.pack(pady=10)
    tk.Label(xlim_frame, text="Xlim Min:").pack(side=tk.LEFT, padx=5)
    xlim_min_entry = ttk.Entry(xlim_frame, width=10)
    xlim_min_entry.insert(0, str(xlim_vals[0]))
    xlim_min_entry.pack(side=tk.LEFT, padx=5)
    tk.Label(xlim_frame, text="Xlim Max:").pack(side=tk.LEFT, padx=5)
    xlim_max_entry = ttk.Entry(xlim_frame, width=10)
    xlim_max_entry.insert(0, str(xlim_vals[1]))
    xlim_max_entry.pack(side=tk.LEFT, padx=5)

    def update_xlim():
        nonlocal xlim_vals
        try:
            xmin = float(xlim_min_entry.get())
            xmax = float(xlim_max_entry.get())
            xlim_vals = [xmin, xmax]
            update_plots()
        except ValueError:
            pass

    xlim_button = ttk.Button(xlim_frame, text="Set Xlim", command=update_xlim)
    xlim_button.pack(side=tk.LEFT, padx=5)

    #  Manual shift for acceleration
    shift_frame = ttk.Frame(root)
    shift_frame.pack(pady=5)
    tk.Label(shift_frame, text="Shift Acceleration (samples):").pack(side=tk.LEFT, padx=5)
    shift_entry = ttk.Entry(shift_frame, width=10)
    shift_entry.pack(side=tk.LEFT, padx=5)

    def apply_manual_shift(shift=None):
        nonlocal azs, Fs, axs, ays, gxs, gys, gzs
        if shift is None:
            try:
                shift = int(shift_entry.get())
            except ValueError:
                return
        azs = np.roll(azs, shift)
        Fs = np.roll(Fs, shift)
        axs = np.roll(axs, shift)
        ays = np.roll(ays, shift)
        gxs = np.roll(gxs, shift)
        gys = np.roll(gys, shift)
        gzs = np.roll(gzs, shift)
        update_plots()


    shift_button = ttk.Button(shift_frame, text="Apply Shift", command=apply_manual_shift)
    shift_button.pack(side=tk.LEFT, padx=5)
    prev_sample_button = ttk.Button(shift_frame, text="<< Prev Sample", command=lambda: apply_manual_shift(-1))
    prev_sample_button.pack(side=tk.LEFT, padx=5)
    next_sample_button = ttk.Button(shift_frame, text="Next Sample >>", command=lambda: apply_manual_shift(1))
    next_sample_button.pack(side=tk.LEFT, padx=5)

    # Time range selection
    range_frame = ttk.Frame(root)
    range_frame.pack(pady=10)

    tk.Label(range_frame, text="Start Time (s):").pack(side=tk.LEFT, padx=5)
    start_time_entry = ttk.Entry(range_frame, width=10)
    start_time_entry.pack(side=tk.LEFT, padx=5)
```

```
    tk.Label(range_frame, text="End Time (s):").pack(side=tk.LEFT, padx=5)
    end_time_entry = ttk.Entry(range_frame, width=10)
    end_time_entry.pack(side=tk.LEFT, padx=5)

    def update_range_visualization():
        try:
            start_time = float(start_time_entry.get())
            end_time = float(end_time_entry.get())
            update_plots(start_time=start_time, end_time=end_time)
        except ValueError:
            pass

    update_range_button = ttk.Button(range_frame, text="Update Range", command=update_range_visualization)
    update_range_button.pack(side=tk.LEFT, padx=5)

    # Save files
    save_frame = ttk.Frame(root)
    save_frame.pack(pady=10)

    def load_trc_file(trc_file):
        with open(trc_file, 'r') as f:
            lines = f.readlines()

        # Extract header information
        data_rate_line = lines[2].strip().split('\t')
        num_frames = int(float(data_rate_line[2]))
        num_markers = int(float(data_rate_line[3]))

        # Extract column names from the 4th line
        column_names_line = lines[3].strip().split('\t')[2:]
        trc_column_names = [name.strip() for name in column_names_line]

        # Extract data
        data = np.genfromtxt(trc_file, skip_header=5, max_rows=num_frames)
        time_trc = data[:, 1]
        trc_data = data[:, 2:]
        return time_trc, trc_data, trc_column_names

    time_trc, trc_data, trc_column_names = load_trc_file(trc_file)



    def save_files():
        nonlocal ts, Fs, axs, ays, azs, gxs, gys, gzs, time_trc, trc_data, trc_column_names, trc_file, sensor_file, video_file

        try:
            start_time = float(start_time_entry.get())
            end_time = float(end_time_entry.get())
            print('start time printed is', start_time)
            print('end time printed is', end_time)
        except ValueError:
            print("Invalid start or end time.")
            return

        # Trim sensor data
        start_idx_sensor = np.searchsorted(ts, start_time, side='left')
        end_idx_sensor = np.searchsorted(ts, end_time, side='right')
        ts_trimmed = ts[start_idx_sensor:end_idx_sensor]
        Fs_trimmed, axs_trimmed, ays_trimmed, azs_trimmed, gxs_trimmed, gys_trimmed, gzs_trimmed = [
            arr[start_idx_sensor:end_idx_sensor] for arr in [Fs, axs, ays, azs, gxs, gys, gzs]
        ]

        # Trim TRC data
        start_idx_trc = np.searchsorted(time_trc, start_time, side='left')
        end_idx_trc = np.searchsorted(time_trc, end_time, side='right')
        time_trc_trimmed = time_trc[start_idx_trc:end_idx_trc]
        trc_data_trimmed = trc_data[start_idx_trc:end_idx_trc, :]

        # Adjust TRC time to start from 0
        time_trc_trimmed = time_trc_trimmed - time_trc_trimmed[0]

        # Save sensor data
        new_sensor_filename = sensor_file.replace('rFIF_', 'FIF_')
        sensor_df = pd.DataFrame({
            'time': ts_trimmed - ts_trimmed[0],
            'F': Fs_trimmed, 'ax': axs_trimmed, 'ay': ays_trimmed, 'az': azs_trimmed,
            'gx': gxs_trimmed, 'gy': gys_trimmed, 'gz': gzs_trimmed
        })
        sensor_df.to_csv(new_sensor_filename, index=False, float_format='%.3f')

        # Save TRC data
        new_trc_filename = trc_file.replace('rKIN_', 'KIN_')
        with open(trc_file, 'r') as original_file:
            header_lines = [original_file.readline() for _ in range(5)]

```

```
        with open(new_trc_filename, 'w') as new_file:
            new_file.writelines(header_lines)
            for i, t in enumerate(time_trc_trimmed):
                row = [str(i + 1), "{:.6f}".format(t)]
                for j in range(trc_data_trimmed.shape[1]):
                    row.append("{:.6f}".format(trc_data_trimmed[i, j]))
                new_file.write("\t".join(row) + "\n")

        # Trim the video using ffmpeg_extract_subclip
        new_video_filename = video_file.replace('VID_', 'tVID_')
        vid = mp.VideoFileClip(video_file)
        trim = vid.subclipped(start_time, end_time)
        trim.write_videofile(new_video_filename)

        print(f"Files saved: {new_sensor_filename}, {new_trc_filename}, {new_video_filename}")


    # Save file button
    save_button = ttk.Button(save_frame, text="Save Files", command=save_files)
    save_button.pack(side=tk.LEFT, padx=5)

    # Run the application
    root.mainloop()

    # Release the video capture when done
    cap.release()
```

---

5) 3) HGT_Creator:

```
import tkinter as tk
from tkinter import ttk, messagebox
import cv2
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from PIL import Image, ImageTk
from scipy.ndimage import gaussian_filter1d


#%%

def HandGrab_creator(video_file, sensor_file):
    """
    Inputs the trimmed video and sensor file.

    Function starts a GUI in which the video can be seen.
    The user can then set the times for which hand/person holds the material.

    Apply the 'start' when the hand is fully grasping the material, and 'stop'
    when the hand starts letting loose of the material.

    """
    # Load sensor data, only time column
    df = pd.read_csv(sensor_file, usecols=[0])

    # Convert the column to numpy
    t_sec = df.to_numpy().flatten()
    t_min = t_sec[0]
    t_max = t_sec[-1]
    print(t_sec)

    # Initialize video capture
    cap = cv2.VideoCapture(video_file)
    if not cap.isOpened():
        raise ValueError("Could not open video file")

    # Video properties
    fps = cap.get(cv2.CAP_PROP_FPS)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    duration = total_frames / fps
    frame_dt = 1.0 / fps
    time_epsilon = 0.5 * frame_dt   # require at least ~1 frame difference

    #playback range for video
    play_min = max(0.0, t_min)
    play_max = min(duration, t_max)

    # Create the main window
    root = tk.Tk()
    root.title(os.path.basename(sensor_file))

    # Create a frame for the video and plot
    frame = ttk.Frame(root)
    frame.pack(fill=tk.BOTH, expand=True)

    # Video display
    video_label = ttk.Label(frame)
    video_label.pack(side=tk.LEFT, padx=8, pady=6)

    #  Time selection box
    time_frame = ttk.Frame(root)
    time_frame.pack(pady=10)
    tk.Label(time_frame, text="Set Time (s):").pack(side=tk.LEFT, padx=5)
    time_entry = ttk.Entry(time_frame, width=10)
    time_entry.pack(side=tk.LEFT, padx=5)

    # Time slider
    time_slider = ttk.Scale(
        root, from_=play_min, to=play_max, orient=tk.HORIZONTAL, length=400,
        command=lambda val: update_time(float(val))
    )
    time_slider.pack(pady=10)

    # Frame step buttons
    button_frame = ttk.Frame(root)
    button_frame.pack(pady=5)
    prev_button = ttk.Button(button_frame, text="<< Prev Frame", command=lambda: step_frame(-1))
    prev_button.pack(side=tk.LEFT, padx=5)
    next_button = ttk.Button(button_frame, text="Next Frame >>", command=lambda: step_frame(1))
    next_button.pack(side=tk.LEFT, padx=5)
```

```
85
86      # Annotation boxes
87      annotation_frame = ttk.Frame(frame)
88      annotation_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=8, pady=6)
89
90      # Save button
91      save_frame = ttk.Frame(annotation_frame)
92      save_frame.pack(fill=tk.X, padx=8, pady=8)
93      save_button = ttk.Button(save_frame, text="Save and Exit", command=lambda: save_and_exit())
94      save_button.pack(side=tk.RIGHT)
95
96      # Left hand annotation
97      left_hand_frame = ttk.LabelFrame(annotation_frame, text="Left Hand")
98      left_hand_frame.pack(fill=tk.X, padx=8, pady=6)
99      left_start_button = ttk.Button(left_hand_frame, text="Starts Grabbing", command=lambda: start_grabbing("left"))
100     left_start_button.pack(side=tk.LEFT, padx=8, pady=6)
101     left_stop_button = ttk.Button(left_hand_frame, text="Stops Grabbing", command=lambda: stop_grabbing("left"))
102     left_stop_button.pack(side=tk.LEFT, padx=8, pady=6)
103     left_undo_button = ttk.Button(left_hand_frame, text="Undo Previous", command=lambda: undo("left"))
104     left_undo_button.pack(side=tk.LEFT, padx=8, pady=6)
105
106     # Right hand annotation
107     right_hand_frame = ttk.LabelFrame(annotation_frame, text="Right Hand")
108     right_hand_frame.pack(fill=tk.X, padx=8, pady=6)
109     right_start_button = ttk.Button(right_hand_frame, text="Starts Grabbing", command=lambda: start_grabbing("right"))
110     right_start_button.pack(side=tk.LEFT, padx=8, pady=6)
111     right_stop_button = ttk.Button(right_hand_frame, text="Stops Grabbing", command=lambda: stop_grabbing("right"))
112     right_stop_button.pack(side=tk.LEFT, padx=8, pady=6)
113     right_undo_button = ttk.Button(right_hand_frame, text="Undo Previous", command=lambda: undo("right"))
114     right_undo_button.pack(side=tk.LEFT, padx=8, pady=6)
115
116     # Below annotation
117     below_frame = ttk.LabelFrame(annotation_frame, text="Below")
118     below_frame.pack(fill=tk.X, padx=8, pady=6)
119     below_start_button = ttk.Button(below_frame, text="Starts Grabbing", command=lambda: start_grabbing("below"))
120     below_start_button.pack(side=tk.LEFT, padx=8, pady=6)
121     below_stop_button = ttk.Button(below_frame, text="Stops Grabbing", command=lambda: stop_grabbing("below"))
122     below_stop_button.pack(side=tk.LEFT, padx=8, pady=6)
123     below_undo_button = ttk.Button(below_frame, text="Undo Previous", command=lambda: undo("below"))
124     below_undo_button.pack(side=tk.LEFT, padx=8, pady=6)
125
126     # Above annotation
127     above_frame = ttk.LabelFrame(annotation_frame, text="Above")
128     above_frame.pack(fill=tk.X, padx=8, pady=6)
129     above_start_button = ttk.Button(above_frame, text="Starts Grabbing", command=lambda: start_grabbing("above"))
130     above_start_button.pack(side=tk.LEFT, padx=8, pady=6)
131     above_stop_button = ttk.Button(above_frame, text="Stops Grabbing", command=lambda: stop_grabbing("above"))
132     above_stop_button.pack(side=tk.LEFT, padx=8, pady=6)
133     above_undo_button = ttk.Button(above_frame, text="Undo Previous", command=lambda: undo("above"))
134     above_undo_button.pack(side=tk.LEFT, padx=8, pady=6)
135
136     # Button refernces
137     buttons = {
138         "left":  {"start": left_start_button,  "stop": left_stop_button,  "undo": left_undo_button},
139         "right": {"start": right_start_button, "stop": right_stop_button, "undo": right_undo_button},
140         "below": {"start": below_start_button, "stop": below_stop_button, "undo": below_undo_button},
141         "above": {"start": above_start_button, "stop": above_stop_button, "undo": above_undo_button},}
142
143     #  Plot
144     fig, axes = plt.subplots(
145     4, 1, sharex=True, figsize=(6, 4),
146     constrained_layout=True  # nicer spacing for stacked plots
147     )
148     canvas = FigureCanvasTkAgg(fig, master=annotation_frame)
149     canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
150
151     # Annotation data
152     L_start_times = []
153     L_end_times = []
154     R_start_times = []
155     R_end_times = []
156     B_start_times = []
157     B_end_times = []
158     A_start_times = []
159     A_end_times = []
160
161     # Map each track to an axis and color
162     tracks = [
163     ("Left Hand",   axes[0], "blue",   L_start_times, L_end_times),
164     ("Right Hand",  axes[1], "green",  R_start_times, R_end_times),
165     ("Below",       axes[2], "orange", B_start_times, B_end_times),
166     ("Above",       axes[3], "red",    A_start_times, A_end_times),]
167
168     # Initials
169     current_frame = 0
170     updating_ui = False
```

```
    active_hand = None

    def update_frame():
        cap.set(cv2.CAP_PROP_POS_FRAMES, current_frame)
        ret, cv_frame = cap.read()
        if ret:
            cv_frame = cv2.cvtColor(cv_frame, cv2.COLOR_BGR2RGB)
            cv_frame = cv2.resize(cv_frame, (640, 480))
            img = Image.fromarray(cv_frame)
            imgtk = ImageTk.PhotoImage(image=img)
            video_label.imgtk = imgtk
            video_label.configure(image=imgtk)

    def update_time(t_val=None):
        """Set current time from slider OR entry without re-triggering the slider callback."""
        nonlocal current_frame, updating_ui
        if t_val is None:
            try:
                t_val = float(time_entry.get())
            except ValueError:
                return

        #  clamp to sensor/video range
        t_val = max(play_min, min(play_max, t_val))

        # Convert to frame
        current_frame = int(t_val * fps)
        current_frame = max(0, min(total_frames - 1, current_frame))

        # Update entry & slider without re-triggering
        try:
            updating_ui = True
            time_entry.delete(0, tk.END)
            time_entry.insert(0, f"{t_val:.3f}")
            time_slider.set(t_val)
        finally:
            updating_ui = False

        update_frame()
        update_plot()


    # Hook the slider so it respects the guard
    def on_slider(val: str):
        nonlocal updating_ui
        if updating_ui:
            return  # ignore programmatic set()
        # val comes in as a string
        update_time(float(val))

    time_slider.configure(command=on_slider)

    def step_frame(delta):
        nonlocal current_frame
        try:
            current_time = float(time_entry.get())
        except ValueError:
            current_time = play_min
        new_time = current_time + delta / fps
        new_time = max(play_min, min(play_max, new_time))
        update_time(new_time)


    # Let Enter in the entry jump to that time
    def on_entry_return(_event=None):
        update_time(None)
    time_entry.bind("<Return>", on_entry_return)

    def set_annotation_buttons_state(active=None):
        """If active is set, only that hand's STOP remains enabled; all other annotation buttons disabled.
           If active is None, re-enable everything."""
        if active is None:
            for hand, b in buttons.items():
                b["start"].configure(state=tk.NORMAL)
                b["stop"].configure(state=tk.NORMAL)
                b["undo"].configure(state=tk.NORMAL)
            return

        # Disable everything first
        for hand, b in buttons.items():
            b["start"].configure(state=tk.DISABLED)
            b["stop"].configure(state=tk.DISABLED)
            b["undo"].configure(state=tk.DISABLED)

        # Only allow STOP for the active hand
        buttons[active]["stop"].configure(state=tk.NORMAL)
```

```
257
258         def start_grabbing(hand):
259             nonlocal active_hand
260             # Enforce one open segment at a time
261             if active_hand is not None:
262                 return  # ignore extra starts while recording another hand
263
264             current_time = current_frame / fps
265             if hand == "left":
266                 L_start_times.append(current_time)
267             elif hand == "right":
268                 R_start_times.append(current_time)
269             elif hand == "below":
270                 B_start_times.append(current_time)
271             elif hand == "above":
272                 A_start_times.append(current_time)
273
274             active_hand = hand
275             set_annotation_buttons_state(active=hand)  # lock UI to only STOP for this hand
276             update_plot()
277
278         def stop_grabbing(hand):
279             nonlocal active_hand
280             # Only allow stop if this hand is currently active
281             if active_hand != hand:
282                 return
283
284             current_time = current_frame / fps
285
286             # Get the last start for this hand (must exist because we're active)
287             if hand == "left":
288                 starts, ends = L_start_times, L_end_times
289             elif hand == "right":
290                 starts, ends = R_start_times, R_end_times
291             elif hand == "below":
292                 starts, ends = B_start_times, B_end_times
293             elif hand == "above":
294                 starts, ends = A_start_times, A_end_times
295             else:
296                 return
297
298             if not starts:
299                 return  # safety; shouldn't happen if active_hand logic is correct
300
301             last_start = starts[-1]
302
303             # Block zero-length (same-frame) segments
304             if abs(current_time - last_start) < time_epsilon:
305                 messagebox.showerror("Invalid annotation", "Cannot start and stop at the same time")
306                 return  # keep recording active; user can move time and try again
307
308             # OK|record the stop and unlock UI
309             ends.append(current_time)
310             active_hand = None
311             set_annotation_buttons_state(active=None)
312             update_plot()
313
314
315         def _undo_pair(starts, ends):
316             """Remove last completed pair. If there's a dangling start, remove it too."""
317             if len(ends) > 0 and len(starts) > 0:
318                 ends.pop()
319                 starts.pop()
320                 return True
321             # Fallback: dangling start without end
322             if len(starts) > 0:
323                 starts.pop()
324                 return True
325             return False
326
327         def undo(hand):
328             # No undo while recording; UI already disables it, but double-guard here
329             if active_hand is not None:
330                 return
331
332             changed = False
333             if hand == "left":
334                 changed = _undo_pair(L_start_times, L_end_times)
335             elif hand == "right":
336                 changed = _undo_pair(R_start_times, R_end_times)
337             elif hand == "below":
338                 changed = _undo_pair(B_start_times, B_end_times)
339             elif hand == "above":
340                 changed = _undo_pair(A_start_times, A_end_times)
341
342             if changed:
```

```
            update_plot()

    def update_plot():
        current_time = current_frame / fps

        for title, ax, color, starts, ends in tracks:
            ax.clear()

            # Draw spans up to current time for open intervals
            for start, end in zip(starts, ends + [current_time]):
                ax.axvspan(start, end, color=color, alpha=0.3)

            # Red dashed time cursor
            ax.axvline(x=current_time, color='red', linestyle='--')

            ax.set_xlim(play_min, play_max)
            ax.set_ylabel(title)
            # Hide y ticks since we're using spans (visual on/off)
            ax.set_yticks([])

        # Common x label on the bottom axis
        axes[-1].set_xlabel("Time (s)")

        canvas.draw_idle()


    # SAVE file part
    # This part processes the start/end times into continuous arrays
    # And applies a Gaussian filter to smooth out the handgrab,
    # As if a person is really grabbing it

    def create_binary_array(t, start_times, end_times):
        """t in seconds, start/end in seconds. Returns 0/1 array aligned to t."""
        binary = np.zeros_like(t, dtype=int)
        for start, end in zip(start_times, end_times):
            if end < start:  # guard (shouldn't happen)
                start, end = end, start
            mask = (t >= start) & (t <= end)
            binary[mask] = 1
        return binary

    def save_and_exit():
        # Confirm the user wants to save
        proceed = messagebox.askyesno(
        "Confirm Save",
        "Are you sure you want to save the annotations?"
        )
        if not proceed:
            return




        # Convert lists to numpy arrays in seconds
        L_starts = np.array(L_start_times, dtype=float)
        L_ends   = np.array(L_end_times,   dtype=float)
        R_starts = np.array(R_start_times, dtype=float)
        R_ends   = np.array(R_end_times,   dtype=float)
        B_starts = np.array(B_start_times, dtype=float)
        B_ends   = np.array(B_end_times,   dtype=float)
        A_starts = np.array(A_start_times, dtype=float)
        A_ends   = np.array(A_end_times,   dtype=float)


        # Curl timing / midpoint shift
        Curltime = 0.05
        midpoint = Curltime / 2.0  # 50 ms shift

        # Make the binary arrays. As they end up as column list, flatten them
        L_base = create_binary_array(t_sec, L_starts - midpoint, L_ends + midpoint).flatten()
        R_base = create_binary_array(t_sec, R_starts - midpoint, R_ends + midpoint).flatten()
        B_base = create_binary_array(t_sec, B_starts - midpoint, B_ends + midpoint).flatten()
        A_base = create_binary_array(t_sec, A_starts - midpoint, A_ends + midpoint).flatten()

        # Gaussian smoothing (cutoff at -3 dB)
        dt = (t_sec[1] - t_sec[0])
        cutoff_hz = 1.0 / Curltime
        sigma = 0.132 / (cutoff_hz * dt)

        # Smooth using gaussian, clip to [0,1], and then round to 6 decimals
        def smooth_clip(x, sigma):
            y = gaussian_filter1d(x.astype(float), sigma, mode="nearest")
            y = np.clip(y, 0.0, 1.0)
            return np.round(y, 6)

        L = smooth_clip(L_base, sigma)
```

```
            R = smooth_clip(R_base, sigma)
            B = smooth_clip(B_base, sigma)
            A = smooth_clip(A_base, sigma)

            # Append columns to the dataframe
            new_cols = {
                'Leftb': L,
                'Right': R,
                'Below': B,
                'Above': A,
            }
            for k, v in new_cols.items():
                df[k] = v

            # sensor_file_filtered is like: FIF_S#_T#_M#_E#.csv
            # Create HGT_S#_T#_M#_E#.csv
            HGT_file = sensor_file.replace('FIF_', 'HGT_')

            if os.path.exists(HGT_file):
                ok = messagebox.askyesno(
                    "File Exists",
                    f"'{HGT_file}' already exists.\nDo you want to overwrite it?"
                )
                if not ok:
                    return

            df.to_csv(HGT_file, index=False)

            print(f"Updated file saved as {HGT_file}")

            # Close the UI
            root.destroy()



        # Initial start time
        update_time(play_min)


        root.mainloop()
        cap.release()

        # Run the application
        root.mainloop()

        # Release the video capture when done
        cap.release()


    #%% TEST CODE
    #HandGrab_creator('reference_video.mp4' ,'FIF_S#_T#_M#_E#.csv')

```

---

6) 4) Scale_and_IK:

```
import os
import pandas as pd
import opensim as osim
import re
from pathlib import Path

def Scale_and_IK(original_model_file, trc_file, SubjectData_file):
    """
    Inputs the base model file, tracer file, and subject data.

    Program scales the model to the subjects data, and saves this scaled model
    in the Subject# folder.

    Program then performs Inverse Kinematics with this scaled model and
    tracer file, and saves this in the Raw folder where .trc is located

    """
    #Function to scale the original model. It is done once, when it exists
    #already it will be skipped
    def ScaleOpenSimModel():
        nonlocal original_model_file, SubjectData_file

        subj_num = re.findall("\d+", SubjectData_file)[0]

        #check if scaled file already exists
        SubjectData_file = Path(SubjectData_file)
        scaled_model_file = SubjectData_file.parent / f"ScaledModel_S{subj_num}.osim"

        if Path(scaled_model_file).exists():
            print(f'Scaled model for subject {subj_num} already exists, continuing..')
            return scaled_model_file

        # Extract subject data
        subject_data = pd.read_csv(SubjectData_file, skiprows=0).to_numpy()
        height = subject_data[0,0] / 100  # Convert cm to m
        mass = subject_data[0,1]          # Weight in kg

        sc_tool = osim.ScaleTool()

        sc_tool.setName(f"subject{subj_num}")
        sc_tool.setSubjectMass(float(mass))
        sc_tool.setSubjectHeight(height)

        # Tell the tool which model to load:
        gmm = sc_tool.getGenericModelMaker()
        gmm.setModelFileName(original_model_file)

        # Configure ModelScaler
        ms = sc_tool.getModelScaler()
        ms.setOutputModelFileName(str(scaled_model_file))
        try:
            ms.setApply(True)
        except Exception:
            pass

        print(f"[info] Scaling model for Subject {subj_num}")
        sc_tool.run()
        print(f"[done] Scaled model Subject {subj_num}")

        return scaled_model_file

    scaled_model_file = ScaleOpenSimModel()

    print(scaled_model_file)

    # Function to perform Inverse Kinematics
    def PerformIK():
        nonlocal scaled_model_file, trc_file
        # Get the directory to top put the files in
        parent_dir = os.path.dirname(trc_file)


        #create IK file based on the trc file
        trc_basename = os.path.basename(trc_file)
        ik_basename = trc_basename.replace("KIN_", "IK_").replace(".trc", ".mot")
        IK_file = os.path.join(parent_dir, ik_basename)

        loadedModel = osim.Model(str(scaled_model_file))
        _ = loadedModel.initSystem()

        # Set up the IK Tool
        ik_tool = osim.InverseKinematicsTool()
        ik_tool.setModel(loadedModel)
```

```
        ik_tool.setMarkerDataFileName(str(trc_file))
        ik_tool.setOutputMotionFileName(str(IK_file))

        # Run IK
        ik_tool.run()
        print(f"IK results saved to {IK_file}")

        return IK_file

    PerformIK()
```

7) 5) COF_analysis:

```
import pandas as pd
import numpy as np
from scipy.fft import fft, fftfreq

def Determine_cutoff_freqs(sensor_file, motion_file):
    """
    Inputs the raw sensor and raw inverse kinematics file.

    Program performs a Fourier analysis over each individual column, and
    finds the optimal cutoff frequency to retain 99.7% of the power for each
    column.
    """

    IK_data_full = pd.read_csv(motion_file, sep='\t', skiprows=10).to_numpy()
    SF_data_full = pd.read_csv(sensor_file, sep=',').to_numpy()

    # In order to determine the effective cutoff, only calculate the cutoff
    # at the time where the pass-on is done. This is about 2 to 7 seconds in
    # almost every video

    # 1s is about 60 indexes for IK
    IK_data = IK_data_full[120:420, :]

    # 1s is about 120 indexes for SF
    SF_data = SF_data_full[240:840, :]

    def Fourier_IK():
        nonlocal IK_data
        # Extract joint angle data, first column is time
        joint_angles = IK_data[:, 1:]   # Shape: (num_samples, num_joints)
        dt = IK_data[1,0]- IK_data[0,0] # Sample time (from time column)

        # Number of samples
        num_samples = joint_angles.shape[0]

        # Compute FFT for each joint and store magnitudes
        freqs = fftfreq(num_samples, dt)[:num_samples//2]  # Positive frequencies only
        all_magnitudes = np.zeros((joint_angles.shape[1], num_samples//2))

        for joint_idx in range(joint_angles.shape[1]):
            fft_result = fft(joint_angles[:, joint_idx])
            magnitude = np.abs(fft_result)[:num_samples//2]
            all_magnitudes[joint_idx, :] = magnitude

        # Calculate average across joints
        avg_magnitude = np.mean(all_magnitudes, axis=0)

        # Calculate the power spectrum (square of magnitude)
        power_spectrum = avg_magnitude**2

        # Normalize the power spectrum to sum to 1
        normalized_power = power_spectrum / np.sum(power_spectrum)

        # Calculate cumulative power
        cumulative_power = np.cumsum(normalized_power)

        # Find the frequency at which 99.7% of the cumulative power occurs
        target_power = 0.997
        i_cof = np.argmax(cumulative_power >= target_power)

        #cutoff frequency
        IK_cof = freqs[i_cof]

        return IK_cof


    def Fourier_SF():
        nonlocal SF_data

        #Extract force and acceleration data seperately
        Force_data  = SF_data[:,1]
        IMU_data    = SF_data[:,2:4]
        dt          = SF_data[1,0] - SF_data[0,0]

        # Number of samples
        num_samples = Force_data.shape[0]

        # Compute FFT for each joint and store magnitudes
        freqs = fftfreq(num_samples, dt)[:num_samples//2]  # Positive frequencies only
        #all_magnitudes = np.zeros((joint_angles.shape[1], num_samples//2))

        #Force cutoff
        #Similar steps as the IK fourier, but compacted
```

```
        magn_Force              = np.abs(fft(Force_data))[:num_samples//2]
        cumul_Force             = np.cumsum(magn_Force**2 / np.sum(magn_Force**2))

        target_power_Force = 0.997
        F_cof    = freqs[np.argmax(cumul_Force >= target_power_Force)]

        #Acceleration cutoff
        all_magn = np.zeros((IMU_data.shape[1], num_samples//2))

        for cardinal in range(IMU_data.shape[1]):
            fft_result = fft(IMU_data[:, cardinal])
            magnitude = np.abs(fft_result)[:num_samples//2]
            all_magn[cardinal, :] = magnitude

        magn_IMU    = np.mean(all_magn, axis=0)
        cumul_IMU   = np.cumsum(magn_IMU**2 / np.sum(magn_IMU**2))

        target_power_IMU = 0.997
        IMU_cof    = freqs[np.argmax(cumul_IMU >= target_power_IMU)]

        return F_cof, IMU_cof


    IK_cof          = Fourier_IK()
    F_cof, IMU_cof  = Fourier_SF()

    return IK_cof, F_cof, IMU_cof


```

---

8) 6) IK_FIMU_Filter:

```
1
2   import os
3   import pandas as pd
4   from scipy.ndimage import gaussian_filter1d
5
6   def filter_data(COF, motion_file, sensor_file):
7       """
8       Inputs the raw sensor and inverse kinematics data, and a user set cutoff
9       frequency.
10
11      Program applies a Gaussian -3dB filter using the cutoff frequency,
12      and outputs it as filtered sensor and IK files.
13
14      """
15
16      if COF == None or 0:
17          print('Error: Cutoff frequency is not set, or is 0.')
18          return
19
20      IK_data = pd.read_csv(motion_file, sep='\t', skiprows=10).to_numpy()
21      SF_data = pd.read_csv(sensor_file, sep=',').to_numpy()
22
23      def gaussian_filt(data):
24          nonlocal COF
25
26          # Set up sigma value
27          dt = data[1,0] - data[0,0]
28          sigma = 0.132 / (COF * dt)
29
30          # Initializing filtered array
31          data_filt = data.copy()
32
33          # Filter every column
34          for i in range(1, data.shape[1]):
35              data_filt[:, i] = gaussian_filter1d(data[:,i].astype(float), sigma=sigma)
36
37          return data_filt
38
39      #Apply filter
40      IK_filt = gaussian_filt(IK_data)
41      SF_filt = gaussian_filt(SF_data)
42
43      #Create 'Processed_Data' map next to Raw map
44      grandparent_dir = os.path.dirname(os.path.dirname(motion_file))
45      processed_data_dir = os.path.join(grandparent_dir, "Processed_Data")
46      os.makedirs(processed_data_dir, exist_ok=True)
47
48      #Save into new files
49      IK_filt_file = motion_file.replace('IK_', 'fIK_').replace('Raw', 'Processed_Data')
50
51      SF_filt_file = sensor_file.replace('FIF', 'fFIF').replace('Raw', 'Processed_Data')
52
53      #Read original files with headers
54      with open(motion_file, 'r') as f:
55          IK_header = [f.readline() for _ in range(11)]  # Adjust range if needed
56
57      with open(sensor_file, 'r') as f:
58          SF_header = [f.readline()]  # Only the first line for CSV
59
60      #Save IK_filt with original header
61      with open(IK_filt_file, 'w') as f:
62          f.writelines(IK_header)
63          for row in IK_filt:
64              f.write("\t".join([f"{val:.6f}" for val in row]) + "\n")
65
66      #Save SF_filt with original header
67      with open(SF_filt_file, 'w') as f:
68          f.writelines(SF_header)
69          for row in SF_filt:
70              f.write(",".join([f"{val:.6f}" for val in row]) + "\n")
71
72      print(f"Filtered files saved: {IK_filt_file}, {SF_filt_file}")
73
74
```

9) 7) EFF_creator:

```
1
2   import os
3   import re
4   import pandas as pd
5   import matplotlib.pyplot as plt
6   import numpy as np
7   from scipy.ndimage import gaussian_filter1d
8   import xml.etree.ElementTree as ET
9
10
11  def FIMU_to_EFF(sensor_filtered_file, handgrab_file, XML_base_file, CalibValues, plot=False):
12      """
13      Code used to create the external force file needed for OpenSim.
14
15      This code combines the force data (for stander materia), acceleration data, and
16      the handgrab file to determine when and how much the weight applied
17      to the subjects hands is.
18
19      For this thesis, the IMU is not accurate enough to transform from a local
20      frame to a world frame corresponding to OpenSim model. Therefore, the
21      acceleration data is instead taken as the magnitude of all accelerations,
22      and only applied in -y direction (gravity direction in OpenSim)
23      """
24
25      # Load data
26      SF_data = pd.read_csv(sensor_filtered_file, sep=',').to_numpy()
27      HG_data = pd.read_csv(handgrab_file, sep=',').to_numpy()
28      CV_data = pd.read_csv(CalibValues, sep=',').to_numpy()
29
30      # Extract variables
31      time  = SF_data[:,0]
32      F     = SF_data[:,1]
33      a_raw = SF_data[:,[2,3,4]]
34
35      Left  = HG_data[:,1]
36      Right = HG_data[:,2]
37      Below = HG_data[:,3]
38      Above = HG_data[:,4]
39
40      F_C     = CV_data[:,0]
41      F_T     = CV_data[:,1]
42      a_scale = CV_data[:,2]
43
44      # Plot the HGT,
45      if plot:
46          plt.plot(time, 1*Left, label='left')
47          plt.plot(time, 2*Right, label='right')
48          plt.plot(time, 3*Below, label='below')
49          plt.plot(time, 4*Above, label='above')
50          plt.legend()
51          plt.xlabel('time (s)')
52          plt.title(f'HGT for {os.path.basename(sensor_filtered_file)}')
53          plt.show()
54
55
56      #From sensor file name, determine which mass to use
57      basename = os.path.basename(sensor_filtered_file)
58      _M = re.compile(r'(?:^|_)M(\d+)(?:_|$)', re.I)
59
60      mass_number = int(_M.search(basename).group(1))
61
62      # m1, m2, m3, m4 are index 3,4,5,6, so add +2
63      m = CV_data[:,mass_number+2]
64
65      # Weight share function, returns m(F) in case of material 1, and also a mF
66      def weight_sharing_function():
67          nonlocal m, mass_number, time, F, F_T, F_C, Left, Right, Below, Above
68          # Part for material 1: Stander, which uses force sensor data
69
70          # Normalize the forces from 0 to 100%
71          # lower and upper limit are the compression/tensile forces, fill in
72          # whichever of the two is lowest/highest respectively
73
74          # There are three steps: 1 Below worker+subject grab, 2 only subject grab
75          # , 3 and above+subject grab.
76          # 1: Subject causes F_T, so at F_T mass for subject = m1
77          # 2: subject only holds object, so mass = m1
78          # 3: Subject causes F_C, so at F_C mass for subject = m1
79          def weight_sharing_stander(m):
80              nonlocal F, F_T, F_C, Below, Above
81
82              # F = F[:1200]
83              # F_T = F_T[:1200]
84              # F_C = F_C[:1200]
```

```
            # Below = Below[:1200]
            # Above = Above[:1200]

            def normalize_to_0_100perc(F, Lower_limit, Upper_limit):
                F_normalized = np.clip(F, Lower_limit, Upper_limit)
                F_normalized = (F_normalized - Lower_limit) / (Upper_limit - Lower_limit)
                return F_normalized


            def create_m1F(m, B, A, F_norm_1, F_norm_3):
                m_array = m*np.ones(len(B))
                mF      = np.ones(len(m_array))

                for i in range(len(m_array)):
                    mF[i] = (
                        m_array[i] * B[i] * F_norm_1[i] + # Part 1
                        m_array[i] * (1 - B[i] - A[i]) +  # Part 2
                        m_array[i] * A[i] * F_norm_3[i]   # Part 3
                    )
                return mF

            F_norm_1 = 1 - normalize_to_0_100perc(F, F_C, F_T) #inverted
            F_norm_3 = normalize_to_0_100perc(F, F_C, F_T)

            mF = create_m1F(m, Below, Above, F_norm_1, F_norm_3)

            return mF


        # Part for materials 2,3,4
        # Same logic as the handgrabfile will be used here, where the m(F) will
        # increase with a gaussian function in the period where the below/above
        # function and either left/right overlap, as a smooth weight transition
        # to the subject

        # Same principle as stander,
        # 1: Transition period below-subject (mF from 0 to 1)
        # 2: subject holds all weight
        # 3: Transition period subject-above (mF from 1 to 0)

        def weight_sharing_other(m):
            nonlocal time, Left, Right, Below, Above


            def part_0_to_1(Left, Right, worker):
                #This is where the subject starts carrying weight

                #Check which of the two overlaps is largest, and use the largest
                #of the two to calculate the midpoint and width
                Overlap_l = Left * worker
                Overlap_r = Right * worker

                if sum(Overlap_l) >= sum(Overlap_r):
                    Overlap = np.copy(Overlap_l)
                else:
                    Overlap = np.copy(Overlap_r)

                mid_0_1 = np.where(Overlap != 0)[0][-1]
                width_0_1 = mid_0_1 - np.where(Overlap != 0)[0][0]


                return mid_0_1, width_0_1


            def part_1_to_0(Left, Right, worker):
                #This is where the subject stops carrying weight

                #Check which of the two overlaps is largest, and use the largest
                #of the two to calculate the midpoint and width
                Overlap_l = Left * worker
                Overlap_r = Right * worker

                if sum(Overlap_l) >= sum(Overlap_r):
                    Overlap = np.copy(Overlap_l)
                else:
                    Overlap = np.copy(Overlap_r)

                mid_1_0 = np.where(Overlap != 0)[0][0]
                width_1_0 = np.where(Overlap != 0)[0][-1] - mid_1_0

                return mid_1_0, width_1_0


            def create_block(time, Left, Right, worker1, worker2):

                mid_0_1, width_0_1 = part_0_to_1(Left, Right, worker1)
```

```
                mid_1_0, width_1_0 = part_1_to_0(Left, Right, worker2)

                trans_0_1 = int(mid_0_1 - 0.5 * width_0_1)
                trans_1_0 = int(mid_1_0 + 0.5 * width_1_0)

                block = np.ones(len(time))

                block[trans_0_1:] = 0
                block[:trans_1_0]  = 0


                return block, mid_0_1, width_0_1, mid_1_0, width_1_0


            def apply_gaussian_filter(block, mid_0_1, width_0_1, mid_1_0, width_1_0):

                # Apply Gaussian filter to the entire block
                # The sigma for the Gaussian filter can be derived from the widths
                sigma_0_1 = width_0_1 / (2 * np.sqrt(2 * np.log(2)))
                sigma_1_0 = width_1_0 / (2 * np.sqrt(2 * np.log(2)))

                smoothed_block = np.copy(block)

                # Apply the filter to the entire block
                smoothed_block[:mid_0_1+width_0_1] = gaussian_filter1d(block[:mid_0_1+width_0_1], sigma=sigma_0_1)
                smoothed_block[mid_1_0-width_1_0:] = gaussian_filter1d(block[mid_1_0-width_1_0:], sigma=sigma_1_0)

                return smoothed_block


            turning_point = int(len(time)/2)

            # Pre: Before the middle of the video (so passing up)
            pre_time = time[turning_point:]
            pre_Left = Left[turning_point:]
            pre_Right = Right[turning_point:]
            pre_Below = Below[turning_point:]
            pre_Above = Above[turning_point:]

            block, mid_0_1, width_0_1, mid_1_0, width_1_0 = create_block(pre_time, pre_Left, pre_Right, pre_Below, pre_Above)
            pre_mF = apply_gaussian_filter(block, mid_0_1, width_0_1, mid_1_0, width_1_0)

            # Post: After the middle of the video (so passing down)
            post_time = time[:turning_point]
            post_Left = Left[:turning_point]
            post_Right = Right[:turning_point]
            post_Below = Below[:turning_point]
            post_Above = Above[:turning_point]

            block, mid_0_1, width_0_1, mid_1_0, width_1_0 = create_block(post_time, post_Left, post_Right, post_Above, post_Below)
            post_mF = apply_gaussian_filter(block, mid_0_1, width_0_1, mid_1_0, width_1_0)

            #Combine the two into one mF again, and multiply with material weight
            mF = m*np.concatenate((pre_mF, post_mF))

            return mF


        if mass_number == 1:
            mF = weight_sharing_stander(m)

        elif mass_number in (2, 3, 4):
            mF = weight_sharing_other(m)

        return mF


    mF = weight_sharing_function()

    # Plot the mF function
    if plot:
        plt.plot(time, mF, label='mF function')
        plt.plot(time, Above, label='Above')
        plt.plot(time, Below, label='Below')
        plt.title(f'mF function for {os.path.basename(sensor_filtered_file)}')
        plt.legend()
        plt.xlabel('time (s)')
        plt.ylabel('Weight (kg) for mF')
        plt.show()

    # Scale a to m/s2
    a = a_raw * 9.81 / a_scale

    """
    Due to IMU local-world frame annoyances, take the total magnitude
    and have it only apply to the negative y (so gravity direction)
```

```
    This can be done due to limited horizontal plane motion.

    This block converts it into a single axis. Remove this block if the IMU
    can be effectively transformed to world-frame.
    """
    a_magn = np.sqrt(
    a[:, 0]**2 +
    a[:, 1]**2 +
    a[:, 2]**2
    )

    # set up new a-array
    a_y = np.zeros_like(a)

    # Y-axis is negative of magnitude (gravity direction)
    a_y[:, 1] = -a_magn

    """
    End of block
    """

    # Force vector via F=ma
    na = np.newaxis

    Fv = mF[:, na] * a_y
    #Fv = mF[:, na] * a

    # Split the force vector into a lefthand and righthand
    # When both hands are holding an object, the force is halved for each hand
    def Fv_Hands():
        nonlocal Fv, Left, Right, na

        #at Overlap of both hands, half the forces
        LR = np.ones(len(Left)) - 0.5*Left*Right

        #Apply to left and right hand
        Fv_L = Fv*Left[:,na] * LR[:,na]
        Fv_R = Fv*Right[:,na] * LR[:,na]

        # Fv_L = np.vstack((Fv_L, Fv[1]*L * LR, Fv[2]*L * LR))
        # Fv_R = np.vstack((Fv_R, Fv[1]*R * LR, Fv[2]*R * LR))

        return Fv_L, Fv_R

    Fv_L, Fv_R = Fv_Hands()

    # Plot the resulting Fv forces on right and left hand
    if plot:
        plt.plot(time, Fv_L[:,1], label='Fv left')
        plt.plot(time, Fv_R[:,1], label='Fv right')
        plt.title(f'Fv functions in y-for {os.path.basename(sensor_filtered_file)}')
        plt.legend()
        plt.xlabel('time (s)')
        plt.ylabel('Force in OpenSim y-direction (N)')
        plt.show()

    #Create .mot external force file
    EFF_file = sensor_filtered_file.replace(r'fFIF_', 'EFF_').replace('.csv', '.mot')

    #Determine headers
    datacolumns = 7  # time + 6 force components (Left and right xyz)
    datarows = len(time)
    time_range = (time[0], time[-1])

    # Open file for writing
    with open(EFF_file, 'w') as f:
        f.write(f"name {os.path.basename(EFF_file)}\n")
        f.write(f"datacolumns {datacolumns}\n")
        f.write(f"datarows {datarows}\n")
        f.write(f"range {time_range[0]:.6f} {time_range[1]:.6f}\n")
        f.write("endheader\n")
        f.write(
            "time\t"
            "R_Force_x\tR_Force_y\tR_Force_z\t"
            "L_Force_x\tL_Force_y\tL_Force_z\n"
        )
        for i in range(datarows):
            f.write(
                f"{time[i]:.6f}\t"
                f"{Fv_R[i,0]:.6f}\t{Fv_R[i,1]:.6f}\t{Fv_R[i,2]:.6f}\t"
                f"{Fv_L[i,0]:.6f}\t{Fv_L[i,1]:.6f}\t{Fv_L[i,2]:.6f}\n"
            )

    print(f"MOT file saved as {EFF_file}")

    def Create_XML_file():
```

```
343         nonlocal EFF_file, XML_base_file
344 
345         # Extract the filename from the full path
346         EFF_file_name = os.path.basename(EFF_file)
347 
348         # Load the XML file
349         tree = ET.parse(XML_base_file)
350         root = tree.getroot()
351 
352         # Find the ExternalForce elements for RightHand and LeftHand
353         for ext_force in root.findall(".//ExternalForce"):
354             data_source = ext_force.find("data_source_name")
355             if data_source is not None:
356                 data_source.text = EFF_file_name #Filename is only base name
357 
358         # Find the datafile element and update its text
359         datafile = root.find(".//datafile")
360         if datafile is not None:
361             datafile.text = EFF_file #Full path for the eff file
362 
363         # Generate the new XML filename
364         XML_file = EFF_file.replace('EFF_', 'XML_').replace('.mot', '.xml')
365 
366         # Save the modified XML file
367         tree.write(XML_file, encoding='UTF-8', xml_declaration=True)
368 
369         print(f"XML file saved as {XML_file}")
370         return XML_file
371 
372 
373     Create_XML_file()
374 
```

10) 8) InverseDynamics:

```python

import os
import opensim as osim
import pandas as pd


def PerformID(subject_model, filt_kin_file, EFF_XML_file):
    """
    Perform Inverse Dynamics on the filtered kinematics using
    the same scaled model, and both apply and not apply the EFF file in the process.

    Then save the results.

    """
    # Get the directory to put the files in
    parent_dir = os.path.dirname(filt_kin_file)

    # Determine the start/end time from the IK file
    IK_data = pd.read_csv(filt_kin_file, sep='\t', skiprows=10).to_numpy()
    start_time = IK_data[0, 0]
    end_time = IK_data[-1, 0]


    # yID is with EFF
    # nID is without EFF
    #yID_file = os.path.join(parent_dir, os.path.basename(filt_kin_file).replace('fIK_', 'yID_').replace('.mot', '.sto'))
    #nID_file = os.path.join(parent_dir, os.path.basename(filt_kin_file).replace('fIK_', 'nID_').replace('.mot', '.sto'))
    yID_file = os.path.basename(filt_kin_file).replace('fIK_', 'yID_').replace('.mot', '.sto')
    nID_file = os.path.basename(filt_kin_file).replace('fIK_', 'nID_').replace('.mot', '.sto')

    # Load the model (assuming it's already scaled and available)
    # Note: You need to pass the correct model file here
    # This is a placeholder; replace with your actual model file
    loadedModel = osim.Model(subject_model)
    _ = loadedModel.initSystem()


    # To have the ID files be put in the correct directory, change working dir
    parent_dir = os.path.dirname(filt_kin_file)
    os.chdir(parent_dir)

    # ID with EFF
    id_tool_with = osim.InverseDynamicsTool()
    id_tool_with.setModel(loadedModel)
    id_tool_with.setCoordinatesFileName(filt_kin_file)
    id_tool_with.setOutputGenForceFileName(yID_file)
    id_tool_with.setExternalLoadsFileName(EFF_XML_file)
    id_tool_with.setLowpassCutoffFrequency(-1)  # = No filtering
    id_tool_with.setStartTime(start_time)
    id_tool_with.setEndTime(end_time)
    id_tool_with.run()


    # ID without EFF
    id_tool_without = osim.InverseDynamicsTool()
    id_tool_without.setModel(loadedModel)
    id_tool_without.setCoordinatesFileName(filt_kin_file)
    id_tool_without.setOutputGenForceFileName(nID_file)
    id_tool_without.setLowpassCutoffFrequency(-1)
    id_tool_without.setStartTime(start_time)
    id_tool_without.setEndTime(end_time)
    id_tool_without.run()

    print(f"ID results with external forces saved to {yID_file}")
    print(f"ID results without external forces saved to {nID_file}")

    return yID_file, nID_file


#
#
# The static optimizations are not used in the result, but the code below does show the steps taken for potential future usage
#
#

import os
import opensim as osim
import pandas as pd
import xml.etree.ElementTree as ET

def PerformSO(IK_file, EFF_XML_file, scaled_model_file):
    # Create reserve actuators to support
    Actuator_Force = 20

```

```
    IK_data = pd.read_csv(IK_file, sep='\t', skiprows=10).to_numpy()
    start_time = IK_data[0, 0]
    end_time =   IK_data[-1, 0]

    # Create the output file name for Static Optimization
    parent_dir = os.path.dirname(IK_file)
    id_basename = os.path.basename(IK_file)
    ySO_basename = id_basename.replace("fIK_", "ySO_").replace(".mot", "")
    ySO_file = os.path.join(parent_dir, ySO_basename)

    nSO_basename = id_basename.replace("fIK_", "nSO_").replace(".mot", "")
    nSO_file = os.path.join(parent_dir, nSO_basename)

    # Load the model
    loadedModel = osim.Model(str(scaled_model_file))

    # Loop over all coordinates in the model
    for i in range(loadedModel.getCoordinateSet().getSize()):
        coord = loadedModel.getCoordinateSet().get(i)
        name = coord.getName()

        # Create a reserve actuator for this coordinate
        reserve = osim.CoordinateActuator(name)
        reserve.setName(f"reserve_{name}")

        # Typical max control (Nm). You can tweak this!
        reserve.setOptimalForce(Actuator_Force)

        # Allow positive and negative torques
        reserve.setMinControl(-1)
        reserve.setMaxControl(1)

        # Add it to the model
        loadedModel.addForce(reserve)

    # Save this new actuated model
    scaled_actuated_model_file = scaled_model_file.replace("ScaledModel", "ScaledActuatedModel")
    loadedModel.printToXML(scaled_actuated_model_file)
    print("Actuated model saved to:", scaled_actuated_model_file)

    _ = loadedModel.initSystem()

    # Define SO object
    so = osim.StaticOptimization()
    so.setStartTime(start_time)
    so.setEndTime(end_time)

    # Set up SO analyze tool with EFF
    so_with = osim.AnalyzeTool()
    so_with.setModelFilename(scaled_actuated_model_file)
    so_with.setCoordinatesFileName(IK_file)
    so_with.setExternalLoadsFileName(EFF_XML_file)

    # Then add SO to analyze tool
    so_with.updAnalysisSet().cloneAndAppend(so)
    so_with.setReplaceForceSet(False)
    so_with.setStartTime(start_time)
    so_with.setFinalTime(end_time)
    so_with.setResultsDir(ySO_file)

    # Print configuration of analyze tool to a xml file
    SO_setup_with_file = os.path.join(parent_dir, "SO_setup_with_EFF.xml")
    so_with.printToXML(SO_setup_with_file)
    print('SO setup file saved to: ', SO_setup_with_file)

    # Load configuration and run the analyses
    so_with = osim.AnalyzeTool(SO_setup_with_file, True)
    so_with.run()

    # Set up SO analyze tool without EFF
    so_without = osim.AnalyzeTool()
    so_without.setModelFilename(scaled_actuated_model_file)
    so_without.setCoordinatesFileName(IK_file)

    # Then add SO to analyze tool
    so_without.updAnalysisSet().cloneAndAppend(so)
    so_without.setReplaceForceSet(False)
    so_without.setStartTime(start_time)
    so_without.setFinalTime(end_time)
    so_without.setResultsDir(nSO_file)

    # Save configuration of analyze tool to a xml file
    SO_setup_without_file = os.path.join(parent_dir, "SO_setup_without_EFF.xml")
    so_without.printToXML(SO_setup_without_file)
    print('SO setup file saved to: ', SO_setup_without_file)
```

```
171     # Load configuration and run the analyses
172     so_without = osim.AnalyzeTool(SO_setup_without_file, True)
173     so_without.run()
174 
175     return ySO_file, nSO_file
176 
177 
```

11) 9) Plot_IK_ID_SO:

```

import tkinter as tk
from tkinter import ttk
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.backends.backend_tkagg import NavigationToolbar2Tk
import moviepy as mp
import os
import sys
import numpy as np
import scipy.stats as sp
from collections import defaultdict
from PIL import Image, ImageTk


#%%

# Forces the pipeline to run from its location
PROJECT_ROOT = os.path.dirname(os.path.abspath(__file__))
if PROJECT_ROOT not in sys.path:
    sys.path.insert(0, PROJECT_ROOT)

from DataExtracting_function import extract_data_from_files

StepPlot_data = extract_data_from_files(
    #subjects=[1], #range(1,100),
    #trials=[1,2],
    # materials=[1,2,3,4],
    # ergonomics=[1,2],
    data_types = ['tVID', 'IK', 'fIK', 'yID', 'nID', 'ySO', 'nSO'],
)

file_groups_plot = defaultdict(dict)
for key, filepath in StepPlot_data.items():
    subject, trial, material, ergonomic, data_type, file_format = key
    group_key = (subject, trial, material, ergonomic)
    file_groups_plot[group_key][data_type] = filepath


for group_key, files in file_groups_plot.items():
    subject, trial, material, ergonomic = group_key

    # Extract the required files
    tVID_file = files.get('tVID')
    fIK_file = files.get(r'fIK')
    yID_file = files.get('yID')
    nID_file = files.get('nID')
    ySO_file = files.get('ySO')
    nSO_file = files.get('nSO')


    # Check if all required files are present
    if tVID_file and fIK_file and yID_file and nID_file and ySO_file and nSO_file:
        print(f"Running Step 3 for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}")
    else:
        missing_files = []
        if not tVID_file:
            missing_files.append("tVID")
        if not fIK_file:
            missing_files.append("fIK")
        if not yID_file:
            missing_files.append("yID")
        if not nID_file:
            missing_files.append("nID")
        if not ySO_file:
            missing_files.append("ySO")
        if not nSO_file:
            missing_files.append("nSO")

        print(f"Missing files for Subject {subject}, Trial {trial}, Material {material}, Ergonomic {ergonomic}: {', '.join(missing_file

#%%
from pprint import pprint

#pprint(StepPlot_data)


#%%
class TrialGUI:
    def __init__(self, root, file_groups):
        self.root = root
        self.file_groups = file_groups
        self.loaded_data = None
```

```
        self.current_subject_trial = None
        self.current_time = 0
        self.video_clip = None
        self.joint_names = []
        self.muscle_names = []
        self.show_vline = True  # <--- New toggle flag

        # Include muscles in GUI
        self.include_muscles = True

        # Load data at startup
        self.load_data()

        # GUI Elements
        self.create_widgets(10,10) # padx=10, pady=10 normally


    def load_data(self):
        """Load all data files into memory."""
        loading_dialog = tk.Toplevel(self.root)
        loading_dialog.title("Loading Data")
        tk.Label(loading_dialog, text="Loading data files, please wait...").pack(padx=20, pady=20)
        loading_dialog.update()

        self.loaded_data = defaultdict(dict)
        for group_key, files in self.file_groups.items():
            subject, trial, material, ergonomic = group_key
            group_data = {}

            # Load IK (.mot)
            IK_path = files.get('IK')
            if IK_path and os.path.exists(IK_path):
                group_data['IK'] = pd.read_csv(IK_path, sep='\t', skiprows=10, header=0)

            # Load fIK (.mot)
            fIK_path = files.get('fIK')
            if fIK_path and os.path.exists(fIK_path):
                group_data['fIK'] = pd.read_csv(fIK_path, sep='\t', skiprows=10, header=0)

            # Load yID and nID (.sto)
            for data_type in ['yID', 'nID']:
                path = files.get(data_type)
                if path and os.path.exists(path):
                    group_data[data_type] = pd.read_csv(path, sep='\t', skiprows=6, header=0)

            # Load ySO and nSO (.sto)
            if self.include_muscles:
                for data_type in ['ySO', 'nSO']:
                    path = files.get(data_type)
                    if path and os.path.exists(path):
                        group_data[data_type] = pd.read_csv(path, sep='\t', skiprows=8, header=0)

            # Store tVID path (video)
            tVID_path = files.get('tVID')
            if tVID_path and os.path.exists(tVID_path):
                group_data['tVID'] = tVID_path

            if group_data:
                self.loaded_data[group_key] = group_data

        loading_dialog.destroy()

    def create_widgets(self, padxval=10, padyval=10):
        """Create all GUI widgets."""
        # Subject/Trial Dropdown
        self.subject_trial_var = tk.StringVar()
        self.subject_trial_dropdown = ttk.Combobox(
            self.root, textvariable=self.subject_trial_var, state="readonly"
        )
        self.subject_trial_dropdown.grid(row=0, column=0, columnspan=3, padx=padxval, pady=padyval)
        self.subject_trial_dropdown.bind("<<ComboboxSelected>>", self.on_subject_trial_select)

        # Toggle button for time-indication line
        self.toggle_vline_button = ttk.Button(
            self.root, text="Toggle V-Line", command=self.toggle_vline
        )
        self.toggle_vline_button.grid(row=0, column=3, padx=padxval, pady=padyval)

        # Populate subject/trial dropdown
        self.subject_trial_options = [
            f"Subject {s}, Trial {t}"
            for s, t, _, _ in self.loaded_data.keys()
        ]
        self.subject_trial_dropdown["values"] = self.subject_trial_options

```

```
        # Time Slider
        self.time_slider = ttk.Scale(
            self.root, from_=0, to=100, orient="horizontal", command=self.on_time_slide
        )
        self.time_slider.grid(row=1, column=0, columnspan=3, padx=padxval, pady=padyval, sticky="ew")

        # Frame Navigation Buttons
        self.prev_button = ttk.Button(self.root, text="Previous Frame", command=self.prev_frame)
        self.prev_button.grid(row=2, column=0, padx=padxval, pady=padyval)
        self.next_button = ttk.Button(self.root, text="Next Frame", command=self.next_frame)
        self.next_button.grid(row=2, column=2, padx=padxval, pady=padyval)

        # Video Block
        self.video_frame = ttk.LabelFrame(self.root, text="Video")
        self.video_frame.grid(row=3, column=0, padx=padxval, pady=padyval, sticky="nsew")
        self.root.grid_columnconfigure(0, weight=9)
        self.root.grid_rowconfigure(3, weight=1)
        self.video_canvas = tk.Canvas(self.video_frame, bg="black", width=480, height=853)
        self.video_canvas.pack()

        # Joints Block
        self.joints_frame = ttk.LabelFrame(self.root, text="Joints")
        self.joints_frame.grid(row=3, column=1, padx=padxval, pady=padyval, sticky="nsew")
        self.root.grid_columnconfigure(1, weight=1)
        self.joint_var = tk.StringVar()
        self.joint_dropdown = ttk.Combobox(self.joints_frame, textvariable=self.joint_var, state="readonly")
        self.joint_dropdown.pack(pady=5)
        self.joint_dropdown.bind("<<ComboboxSelected>>", self.update_plots)
        self.fig_joints, (self.ax_fIK, self.ax_yID_nID) = plt.subplots(2, 1, figsize=(6, 8))
        self.fig_joints.subplots_adjust(hspace=0.4)
        self.canvas_joints = FigureCanvasTkAgg(self.fig_joints, master=self.joints_frame)
        self.toolbar = NavigationToolbar2Tk(self.canvas_joints, self.joints_frame)
        self.toolbar.update()
        self.canvas_joints.get_tk_widget().pack()

        # Analysis Block
        self.analysis_frame = ttk.LabelFrame(self.root, text="Analysis")
        self.analysis_frame.grid(row=3, column=2, padx=padxval, pady=padyval, sticky="nsew")
        self.root.grid_columnconfigure(2, weight=0)
        self.time_range_var = tk.StringVar() # Have the option to limit the analysis range
        self.time_range_var.set("0,0")  # default: full range
        ttk.Label(self.analysis_frame, text="Time Range (s):").pack(anchor="nw", padx=5, pady=(5,0))
        self.time_range_entry = ttk.Entry(self.analysis_frame, textvariable=self.time_range_var)
        self.time_range_entry.pack(anchor="nw", fill="x", padx=5, pady=(0,5))
        self.analysis_text = tk.Label(self.analysis_frame, text="", justify="left", anchor="nw")
        self.analysis_text.pack(fill="both", expand=True, padx=5, pady=5)
        # Separator label
        ttk.Label(self.analysis_frame, text="\nEvent times:").pack(anchor="nw", padx=5, pady=(5,0))

        # Dictionary to store the event entries
        self.event_entries = {}
        event_names = [("Neutral", "N"),
                       ("Bend-Pass", "BP"),
                       ("Neutral-weighted", "NW"),
                       ("Lift-Pass", "LP")]

        for name, short_label in event_names:
            frame = ttk.Frame(self.analysis_frame)
            frame.pack(fill="x", padx=5, pady=2)
            ttk.Label(frame, text=f"{name}:").pack(side="left")
            var = tk.StringVar()
            entry = ttk.Entry(frame, textvariable=var)
            entry.pack(side="left", fill="x", expand=True)
            self.event_entries[name] = (var, short_label)
            # Bind to update plots when changed
            entry.bind("<FocusOut>", lambda e: self.update_plots())

        if self.include_muscles:
            # Muscles Block
            self.muscles_frame = ttk.LabelFrame(self.root, text="Muscles")
            self.muscles_frame.grid(row=3, column=3, padx=padxval, pady=padyval, sticky="nsew")
            self.root.grid_columnconfigure(3, weight=1)
            self.muscle_var = tk.StringVar()
            self.muscle_dropdown = ttk.Combobox(self.muscles_frame, textvariable=self.muscle_var, state="readonly")
            self.muscle_dropdown.pack(pady=5)
            self.muscle_dropdown.bind("<<ComboboxSelected>>", self.update_plots)
            self.fig_muscles, (self.ax_muscles_l, self.ax_muscles_r) = plt.subplots(2, 1, figsize=(6, 8))
            self.fig_muscles.subplots_adjust(hspace=1)
            self.canvas_muscles = FigureCanvasTkAgg(self.fig_muscles, master=self.muscles_frame)
            self.toolbar = NavigationToolbar2Tk(self.canvas_muscles, self.muscles_frame)
            self.toolbar.update()
            self.canvas_muscles.get_tk_widget().pack()

    def on_subject_trial_select(self, event):
        """Handle subject/trial selection."""
        selection = self.subject_trial_var.get()
```

```
        # Remove "Subject " and "Trial " and split on ", "
        parts = selection.replace("Subject ", "").replace("Trial ", "").split(", ")
        subject = int(parts[0])
        trial = int(parts[1])
        self.current_subject_trial = next(
            k for k in self.loaded_data.keys()
            if k[0] == subject and k[1] == trial
        )

        s, t, m, e = self.current_subject_trial
        self.trial_info = (f"S{s}", f"T{t}", f"M{m}", f"E{e}")

        self.current_time = 0
        self.time_slider.config(to=len(self.loaded_data[self.current_subject_trial]['fIK']) - 1)
        self.load_video()
        self.update_joint_muscle_dropdowns()
        self.update_plots()

    def toggle_vline(self):
        """Toggle vertical line on/off."""
        self.show_vline = not self.show_vline
        self.update_plots()

    def load_video(self):
        """Load the video for the current subject/trial."""
        tVID_path = self.loaded_data[self.current_subject_trial]['tVID']
        self.video_clip = mp.VideoFileClip(tVID_path)

    def update_joint_muscle_dropdowns(self):
        """Update joint and muscle dropdowns based on current data."""
        fIK_data = self.loaded_data[self.current_subject_trial]['fIK']
        self.joint_names = [col for col in fIK_data.columns if col != "time"]
        self.joint_dropdown["values"] = self.joint_names
        self.joint_var.set(self.joint_names[0] if self.joint_names else "")

        if self.include_muscles:
            # Handle ySO data
            ySO_data = self.loaded_data[self.current_subject_trial].get('ySO', None)
            if ySO_data is not None:
                self.muscle_names = [col for col in ySO_data.columns if col.endswith(('_l', '_r'))]
                self.muscle_dropdown["values"] = sorted(set(m.replace('_l', '').replace('_r', '') for m in self.muscle_names))
                self.muscle_var.set(self.muscle_dropdown["values"][0] if self.muscle_dropdown["values"] else "")
            else:
                print("Warning: ySO data not found for this subject/trial.")
                self.muscle_dropdown["values"] = []
                self.muscle_var.set("")

    def on_time_slide(self, event):
        """Update current_time and plots when slider is moved."""
        self.current_time = int(float(event))
        self.update_plots()  # Update plots in real-time

    def prev_frame(self):
        """Move to previous frame."""
        self.current_time = max(0, self.current_time - 1)
        self.time_slider.set(self.current_time)
        self.update_plots()

    def next_frame(self):
        """Move to next frame."""
        max_time = len(self.loaded_data[self.current_subject_trial]['fIK']) - 1
        self.current_time = min(max_time, self.current_time + 1)
        self.time_slider.set(self.current_time)
        self.update_plots()


    def update_plots(self, event=None):
        """Update video frame and plots based on current_time."""
        if not self.current_subject_trial:
            return

        # Update video frame
        if self.video_clip:
            frame = self.video_clip.get_frame(self.current_time / self.video_clip.fps)
            # Convert the frame to a PIL Image
            pil_image = Image.fromarray(frame)
            # Convert the PIL Image to a Tkinter PhotoImage
            tk_image = ImageTk.PhotoImage(pil_image)
            # Update the canvas
            self.video_canvas.delete("all")
            self.video_canvas.create_image(0, 0, anchor="nw", image=tk_image)
            # Keep a reference to avoid garbage collection
            self.video_canvas.image = tk_image

        # Update joint plots
        IK_data  = self.loaded_data[self.current_subject_trial]['IK']
```

```
        fIK_data = self.loaded_data[self.current_subject_trial]['fIK']
        yID_data = self.loaded_data[self.current_subject_trial]['yID']
        nID_data = self.loaded_data[self.current_subject_trial]['nID']
        joint = self.joint_var.get()

        # Find matching columns in yID_data and nID_data
        yID_matching_cols = [col for col in yID_data.columns if col.startswith(joint)]
        nID_matching_cols = [col for col in nID_data.columns if col.startswith(joint)]

        # Plot fIK data
        if joint in fIK_data.columns:
            self.ax_fIK.clear()
            self.ax_fIK.plot(IK_data["time"], IK_data[joint], label=f"Raw {joint}", color='r')
            self.ax_fIK.plot(fIK_data["time"], fIK_data[joint], label=f"Filt {joint}", color='g')
            if self.show_vline:
                self.ax_fIK.axvline(x=fIK_data["time"][self.current_time], color="red", linestyle="--")
            self.ax_fIK.grid()
            self.ax_fIK.legend()
            self.ax_fIK.set_title(f"Joint '{joint}' angle over time for ({', '.join(self.trial_info)})",
                                  y=1.05)
            self.ax_fIK.set_xlabel('Time (s)')
            self.ax_fIK.set_ylabel('Angle (deg)')
        else:
            self.ax_fIK.clear()
            self.ax_fIK.set_title(f"Joint '{joint}' not found in fIK_data.")

        # Plot yID and nID data if matching columns are found
        if yID_matching_cols and nID_matching_cols:
            self.ax_yID_nID.clear()
            for col in yID_matching_cols:
                self.ax_yID_nID.plot(yID_data["time"], yID_data[col], label=f"yID {col}", color='g')
            for col in nID_matching_cols:
                self.ax_yID_nID.plot(nID_data["time"], nID_data[col], label=f"nID {col}", color='r', ls=':')

            if self.show_vline:
                self.ax_yID_nID.axvline(x=yID_data["time"][self.current_time], color="red", linestyle="--")
            self.ax_yID_nID.grid()
            self.ax_yID_nID.legend()
            self.ax_yID_nID.set_title(f"Joint '{joint}' over time for ({', '.join(self.trial_info)})",
                                      y=1.05)
            self.ax_yID_nID.set_xlabel('Time (s)')
            if any(substring in joint for substring in ['tx', 'ty', 'tz', 'beta']):
                self.ax_yID_nID.set_ylabel('Force (N)')
            else:
                self.ax_yID_nID.set_ylabel('Moment (Nm)')

        else:
            self.ax_yID_nID.clear()
            self.ax_yID_nID.set_title(f"No matching columns for '{joint}' in yID/nID data.")


        if self.include_muscles:
            # Update muscle plots
            ySO_data = self.loaded_data[self.current_subject_trial]['ySO']
            nSO_data = self.loaded_data[self.current_subject_trial]['nSO']
            muscle_base = self.muscle_var.get()

            self.ax_muscles_l.clear()
            self.ax_muscles_l.plot(ySO_data["time"], ySO_data[f"{muscle_base}_l"], label=f"ySO {muscle_base}_l", color='g')
            self.ax_muscles_l.plot(nSO_data["time"], nSO_data[f"{muscle_base}_l"], label=f"nSO {muscle_base}_l", color='r', ls=':')
            if self.show_vline:
                self.ax_muscles_l.axvline(x=ySO_data["time"][self.current_time], color="red", linestyle="--")
            self.ax_muscles_l.grid()
            self.ax_muscles_l.legend()
            self.ax_muscles_l.set_title(f"Muscle '{muscle_base}'_l activation over time for ({', '.join(self.trial_info)})")
            self.ax_muscles_l.set_xlabel('Time (s)')
            self.ax_muscles_l.set_ylabel('Activation')

            self.ax_muscles_r.clear()
            self.ax_muscles_r.plot(ySO_data["time"], ySO_data[f"{muscle_base}_r"], label=f"ySO {muscle_base}_r", color='g')
            self.ax_muscles_r.plot(nSO_data["time"], nSO_data[f"{muscle_base}_r"], label=f"nSO {muscle_base}_r", color='r', ls=':')
            self.ax_muscles_r.axvline(x=ySO_data["time"][self.current_time], color="red", linestyle="--")
            self.ax_muscles_r.grid()
            self.ax_muscles_r.legend()
            self.ax_muscles_r.set_title(f"Muscle '{muscle_base}'_r activation over time for ({', '.join(self.trial_info)})",
                                        y=1.05)
            self.ax_muscles_r.set_xlabel('Time (s)')
            self.ax_muscles_r.set_ylabel('Activation')



        # Plot event lines
        for name, (var, label_short) in self.event_entries.items():
            times_str = var.get().split(",")  # allow multiple times separated by commas
            for t_str in times_str:
                t_str = t_str.strip()
```

```
                if t_str == "":
                    continue
                try:
                    t_val = float(t_str)
                    # Add vertical line to all relevant axes (fIK, yID/nID, muscles)
                    axes_to_draw = [self.ax_fIK, self.ax_yID_nID, self.ax_muscles_l, self.ax_muscles_r]
                    for ax in axes_to_draw:
                        ax.axvline(x=t_val, color="dimgray", linestyle="--", linewidth=1)
                        # Add text label above the line
                        ylim = ax.get_ylim()
                        ax.text(t_val, ylim[1], label_short, color="dimgray", fontsize=8,
                                verticalalignment="bottom", horizontalalignment="center")
                except ValueError:
                    pass  # ignore invalid entries

        self.canvas_joints.draw()

        if self.include_muscles:
            self.canvas_muscles.draw()

        # Stat analysis for yID and nID
        if yID_matching_cols and nID_matching_cols:
            # Use the first matching column
            # Parse time range from entry
            try:
                t_start, t_end = [float(x.strip()) for x in self.time_range_var.get().split(",")]

                if t_end == 0:
                    t_end = yID_data["time"].iloc[-1]
            except Exception:
                # fallback if invalid entry
                t_start, t_end = yID_data["time"].iloc[0], yID_data["time"].iloc[-1]

            # Filter data based on time range
            mask_y = (yID_data["time"] >= t_start) & (yID_data["time"] <= t_end)
            mask_n = (nID_data["time"] >= t_start) & (nID_data["time"] <= t_end)

            y_vals = yID_data[yID_matching_cols[0]][mask_y].dropna()
            n_vals = nID_data[nID_matching_cols[0]][mask_n].dropna()

            # Ensure equal lengths
            min_len = min(len(y_vals), len(n_vals))
            y_vals = y_vals[:min_len]
            n_vals = n_vals[:min_len]

            #take absolute values
            y_vals = np.abs(y_vals)
            n_vals = np.abs(n_vals)

            # Stats
            y_mean, y_std = np.mean(y_vals), np.std(y_vals)
            n_mean, n_std = np.mean(n_vals), np.std(n_vals)

            y_cum = np.trapz(np.abs(y_vals), yID_data["time"][:min_len])
            n_cum = np.trapz(np.abs(n_vals), nID_data["time"][:min_len])

            #Paired t-test
            t_stat, p_val = sp.ttest_rel(y_vals, n_vals)
            signif = "YES" if p_val < 0.05 else "NO"

            try:
                w_stat, w_pval = sp.wilcoxon(y_vals, n_vals)
                w_signif = "YES" if w_pval < 0.05 else "NO"
            except ValueError:
                # Wilcoxon requires non-zero differences; handle edge case
                w_stat, w_pval, w_signif = np.nan, np.nan, "N/A"

            # Cohens d, difference
            diff = y_vals - n_vals
            cohens_d = np.mean(diff) / np.std(diff, ddof=1)

            analysis_str = (
                f"Joint: {joint}\n\n"
                f"yID: Mean={y_mean:.2f}, Std={y_std:.2f}, Cumulative={y_cum:.2f}\n"
                f"nID: Mean={n_mean:.2f}, Std={n_std:.2f}, Cumulative={n_cum:.2f}\n\n"
                f"T-test (=0.05): p={p_val:.4f} → Significant: {signif}\n"
                f"Wilcoxon test: p={w_pval:.4f} → Significant: {w_signif}\n"
                f"Cohen's d (effect size): {cohens_d:.3f}"
            )
        else:
            analysis_str = "No matching joint data available."

        self.analysis_text.config(text=analysis_str)


def save_joint_statistics_to_csv(file_groups, loaded_data):
```

```
    # Define the joints of interest
    joints_of_interest = ["lumbar_ext",
        "lumbar_rota", "lumbar_bend",
        "shoulder_flexion_l", "shoulder_flexion_r",
        "shoulder_adduction_l", "shoulder_adduction_r",
        "elbow_flexion_l", "elbow_flexion_r"
    ]

    # Prepare a list to store results
    results = []

    for group_key, files in file_groups.items():
        subject, trial, material, ergonomic = group_key
        trial_info = f"S{subject}_T{trial}_M{material}_E{ergonomic}"

        # Skip if required data is missing
        if not all(data_type in loaded_data[group_key] for data_type in ['yID', 'nID']):
            continue

        yID_data = loaded_data[group_key]['yID']
        nID_data = loaded_data[group_key]['nID']

        row = {"subject_trial": trial_info}

        for joint in joints_of_interest:
            # Find matching columns in yID and nID data
            yID_matching_cols = [col for col in yID_data.columns if col.startswith(joint)]
            nID_matching_cols = [col for col in nID_data.columns if col.startswith(joint)]

            if yID_matching_cols and nID_matching_cols:
                # Use the first matching column
                y_vals = yID_data[yID_matching_cols[0]].dropna()
                n_vals = nID_data[nID_matching_cols[0]].dropna()

                # Ensure equal lengths
                min_len = min(len(y_vals), len(n_vals))
                y_vals = y_vals[:min_len]
                n_vals = n_vals[:min_len]

                # Take absolute values
                y_vals = np.abs(y_vals)
                n_vals = np.abs(n_vals)

                # Compute statistics
                y_mean, y_std = np.mean(y_vals), np.std(y_vals)
                n_mean, n_std = np.mean(n_vals), np.std(n_vals)

                # Add to row
                row[f"n_{joint}"] = f"{n_mean:.2f}±{n_std:.2f}"
                row[f"y_{joint}"] = f"{y_mean:.2f}±{y_std:.2f}"
            else:
                # If no matching columns, fill with NaN
                row[f"n_{joint}"] = "NaN"
                row[f"y_{joint}"] = "NaN"

        results.append(row)

    # Convert results to DataFrame and save as CSV
    df = pd.DataFrame(results)
    df.to_csv("joint_statistics.csv", index=False)
    print("Saved joint statistics to joint_statistics.csv")



#%%

#set legend size
plt.rc('legend',fontsize=8) # using a size in points

root = tk.Tk()
#root.geometry("3440x1440") #widescreen
root.geometry("2880x1800")
app = TrialGUI(root, file_groups=file_groups_plot)
save_joint_statistics_to_csv(file_groups_plot, app.loaded_data)
root.mainloop()

```