Inverse Dynamic Model Calibration

Table of Contents

Installation Requirements

Full use of the software and files contained in the project will require:

- MATLAB, including the optimization and statistics toolboxes. We have tested the code from version 2012b to 2015b, and there have been no compatibility issues.
- OpenSim 3.3, which can be freely obtained at simtk.org. Other versions of OpenSim may work, however there have been some incompatibility issues so we recommend using version 3.3

No further program installations will be necessary, just extract the folders and all programs should run within Matlab/OpenSim.

Experimental Data

The files are named with the convention of:

PatientAlias_MotionType_Speed_TrialNumber

PatientAlias: The subject whose data is provided is called Patient4.
MotionType: All of the data provided is treadmill gait (TMGait)
Speed: Each speed is written as 0ptX instead of 0.X
TrialNumber: The data was recorded in 1 minute intervals, so each trail contains a minute of gait.

Three kinds of files are provided:

-.trc files:

Contain marker motion data according to the format used by MotionAnalysis systems. Sampled at 100 Hz.

-csv files:

Contain raw unprocessed EMG data as well as ground reaction forces and moments. Sampled at 1000 Hz.

Phase 1 - Parsing and Formatting

This section contains code for parsing TRC and CSV files into steps with a 20% of cycle pad at the beginning and end of the cycle. Data was resampled to 141 time points (heel strike occurs at frame 21 and 121). Furthermore, the GRF data is low pass filtered at 7/T Hertz, where T is the period of the gait cycle. The GRF data was formatted to be directly inputted into an OpenSim inverse dynamics analysis without further modification.

Files Provided:

Patient4_TMGait_Opt#_##.csv
Patient4_TMGait_Opt#_##.TRC
ParseAndFormatDataOpenSim.m

Usage steps:

1. Input lab specific and data collection specific information
   - Assign which leg is being studied
   - Assign new order for GRF and marker data
   - Assign force plate centers
   - Assign distance from force plate center to electrical center
   - Assign electric center
   - Assign corners of the force plate (force plate for left leg is assigned first)
2. Load all data
3. Filter data
   - GRF data is low pass filtered at 7/T Hertz, where T is the period of the gait cycle.
4. Locate heel strikes
   - Approximate locations of heel strikes and toe offs were found by finding instances where the vertical ground reaction forces either approach of move away from zero
   - Once heel strike locations were found, determine the beginning and endings of cycles
   - Add a 20% of cycle pad at the beginning and end of the cycle
5. Separate data into individual steps
6. Filter and Resample data
   - Resample the data to 141 points
7. Output: marker.trc, GRF.mot, and EMG.mot
   - Reorder the marker and GRF data before outputting files

Files Generated:

Patient4_TMGait_Opt##_TrialNumber_leg_StepNumber.trc
Patient4_TMGait_Opt##_TrialNumber_leg_StepNumber_GRF.mot
Patient4_TMGait_Opt##_TrialNumber_leg_StepNumber_EMG.mot

Phase 2 - Model Scaling

Reordering Marker Data

This section contains code to change the order of and scale columns of the experimental marker data trc file in order to be inputted to OpenSim.

Files Provided:

Patient4_Static01.TRC
ReorderMarkerData.m

Function call:

```
ReorderMarkerData('trialname’,x,y,z,scale factor)
```

Usage steps:

1. Load marker data
2. Scale data
   - The data is scaled according to scale factor chosen
3. Reorder data
   - The data should be reordered with the same assignment given in the Parsing and Formatting code
4. Output reordered and scaled marker data

Files Generated:

Patient4_Static01_reordered.trc

Scaling Model

Perform model scaling in OpenSim using the settings file (xml) provided.

Files Provided:

Scale.xml
BJ_ArnoldHamner_LegsTorsoArms_v2.1.osim

Generated Files Required:

Patient4_Static01_reordered.trc

Usage steps:

1. The generic Arnold Hammer model is modified to only contain 29 degrees of freedom (DOF) by removing toes, forearm, and wrist DOFs

Files Generated:

Patient4_scaled.osim

Add Marker Plates To Model

This section contains code to add marker plates to an existing osim model

Files Provided:

addMarkerPlatesToModel.m

Generated Files Required:

Patient4_scaled.osim

Function call:

```
addMarkerPlatesToModel('inModel.osim')
```

addMarkerPlatesToModel('inModel.osim','outModel.osim')

Usage steps:

1. Load osim model
2. Locate marker body names as well as position in each body segment
   — Markers that lie in the same body segment are grouped into a plate
3. Initialize plate location in parent bodies
4. Find origin locations for marker plate bodies
   — The average is found between all markers found in the same body segment. The average is then set as the new location of the plate
5. Create new marker locations in reference to newly define marker plate
6. Output plated osim model

Files Generated:

Patient4_plated.osim

Kinematic Calibration

Utility Function

This portion of the code contains all the utility functions used by the kinematic calibration routine.

Files Provided:

Patient4_HipIso.trc
Patient4_KneeIso.trc
Patient4_AnkleIso.trc
IKTasksSetupFile_Ankle.xml
IKTasksSetupFile_Hip.xml
IKTasksSetupFile_Knee.xml
IKTasksSetupFile.xml
ScaleSetSetupFile.xml
inputSetupFile.xml
utilFunctions.m
xmlRead.m

Generated Files Required:

Patient4_plated.osim
Patient4_TMGait_0pt5_02_left_25.trc

Functions Included in the Utility Function Code:

1. readInputSetupFile()
   — Reads the XML input setup file (using 3rd party code, xmlRead, written by Jarek Tuszynski) to collect user-specified joint parameter, marker plate parameter, and input settings information. The input argument to the

function is the name of the input setup file. The function returns three structure arrays containing information about joint parameters, marker plate parameters, and input settings, respectively. These arrays are used in the main kinematic calibration routine.

- Example function call: **[jS,mS,iS]=readInputSetupFile('inputSetupFile.xml')**

2. getParams()
   - Extracts model parameters, i.e., design variables, as well as user-specified maximum allowable movement change using both the joint and marker plate structures, derived by reading the input setup XML file, and the .osim model. The inputs to this function are a joint structure array, a marker plate structure array, and a .osim model. The function returns the design parameters and maximum allowable movement change vectors.
   - Example function call: **[v,vMax]=getParams(jointStruct,plateStruct,model);**
3. setParams()
   - Modifies the .osim model parameters. The input arguments to this function are the joint and marker plate structure arrays, the .osim model, the design parameters, and the maximum allowable movement change vectors. The function does not return an output argument.
   - Example function call: **setParams(jointStructOpt,plateStructOpt,model,v,dv);**
4. getCaseNumber()
   - Finds the appropriate case number associated with the selected model parameter(s) by referencing a lookup table (LUT). The input to this function is the vector containing the selected model parameters--in the form of 1s and 0s. The function returns the assigned case number vector.
   - Example function call: **caseNumber=getCaseNumber(jS.LocationInParent(index,:));**
5. modifyVariable()
   - Modifies appropriate variable(s) based on the derived case number obtained from getCaseNumber() routine. The input arguments to this function are the current variables, case number, design variable index location, design parameter(s), and vector of maximum allowable movement change. The function returns the vector of modified variable(s).
   - Example function call: **modVar=modifyVariable(currentVar(index,:), caseNumber, indexLocation,v,dv);**
6. calcMarkerErrors()
   - Computes the time-varying error of each marker in the model compared to the experimental data.The input arguments to this function are the .osim model, the inverse kinematics solver object, sampling time (sec), number of frames, number of markers in the model, and simulation start time (sec). The function returns the marker errors, virutal marker locations, and marker names.

— Example function call: **[markerErr,markerPos,markerNames]=calcMarkerErrors(model, ikSolver,dt, Nframes, Nmarkers, startTime);**

7. initOptStruct()

   — Initializes optimization specific parameters. The input arguments to this function are the joint and marker plate structures, the number of joints and number of marker plates, and the joints and marker plates that are included in the optimization (i.e., whichJoints and whichMarkerPlates). The function returns the optimization specific joint and marker plate structure arrays, respectively.

   — Example function call: **[jSOpt,mSOpt]=initOptStruct(jS,mS,numOfJoints, numOfMarkerPlates,whichJoints, whichMarkerPlates);**

8. buildSymConstraints()

   — Builds symmetry constraints and passes them to the main kinematic calibration function. The input arguments to this function are the optimization grouping structure array and the optimization specific joint structure array (not to be confused with the input joint structure array). The function returns the index pairs for the constrained values, for translations, orientations, or both.

   — Example function call for translation indices: **cnstrPairIndTrn=buildSymConstraints(optGroupStruct, jointStructOpt, numOfJointParams);**

9. readTrcFile()

   — Opens a .trc file and extracts marker names and locations for use in the main kinematic calibration routine. The input argument to the function is the name of the .trc file. The function returns the marker names (in cell array), a matrix containing marker locations, and a time vector.

   — Example function call: **[markerNames,markerLocations,time]=readTrcFile(markerFileName);**

10. rescaleModel()

    — Rescales the optimized model for cosmetic purposes. The input argument to this function is a structure array containing the following variables:

    - v: design parameters
    - dv: maximum allowable change
    - vJoint: joint design parameters
    - model: .osim model
    - numOfJoints: number of joints
    - allJointNames: joint names
    - jointStructOpt: optimization specific joint structure
    - optGroup: optimization group structure
    - numOfJointParams: number of joint parameters
    - optSettings: optimization settings
    - markerPlateStructOpt: marker plate structure

- numOfModelJoints: number of model joints

— The function returns a scaled model.

— Example function call: **rescaleModel(inputArgStruct)**

Usage steps:

1. Assign the variables in the following lines of the InputSetupFile.xml

Line	Variable	Example
358	`<input_model_file>`	`Patient4_plated.osim`
359	`<accuracy>`	`1e-5`
360	`<output_model_file>`	`Patient4_optModel.osim`
361	`<scale_set_file>`	`ScaleSetSetupFile.xml`
368	`<marker_file>`	`Patient4_TMGait_0pt5_02_left_25.trc`
369	`<time_range>`	`31.860000000000 33.730000000000`
373	`<ik_set_file>`	`IKTasksSetupFile.xml`
388	`<marker_file>`	`Patient4_HipIso.trc`
389	`<time_range>`	`0 26.9500`
393	`<ik_set_file>`	`IKTasksSetupFile_Hip.xml`
419	`<marker_file>`	`Patient4_KneeIso.trc`
420	`<time_range>`	`0 18.6200`
424	`<ik_set_file>`	`IKTasksSetupFile_Knee.xml`
441	`<marker_file>`	`Patient4_AnkleIso.trc`
442	`<time_range>`	`0 15.6600`
446	`<ik_set_file>`	`IKTasksSetupFile_Ankle.xml`
461	`<marker_file>`	`Patient4_TMGait_0pt5_02_left_25.trc`
462	`<time_range>`	`31.860000000000 33.730000000000`
466	`<ik_set_file>`	`IKTasksSetupFile.xml`

Kinematic Calibration

This portion of the code run the kinematic calibration

Files Provided:

kinCal.m

Generated Files Required:

costFunction.m

utilFunctions.m

Usage steps:

1. The *readInputSetupFile* is first called upon to load the joint parameters, marker plate parameters, and optimization settings

   — The parent and child location/orientation parameters stated in the input setup file aren't the actual model parameters. The parent location of a joint stating <0 0 1> indicates that the joint will only be optimized in the z direction.

2. The optimization loop begins by setting up the inverse kinematic solver

	Experimental Data	Joints	Marker Plates
1	Walking Trial	N/A	Torso and Pelvis
2	Hip Isolation	Left and Right Hip Joints	Pelvis, Left and Right Femur
3	Knee Isolation	Left and Right Knee Joints	Left and Right Tibia
4	Ankle Isolation	Left and Right Ankle and Subtalar Joints	N/A
5	Walking Trial	Left and Right Hip Joints, Left and Right Knee Joints, Left and Right Ankle and Subtalar Joints	Torso, Pelvis, Left and Right Femur, and Left and Right Tibia

3. The *calcMarkerErrors* is then called to calculate the marker errors prior to optimization
4. Next, *readTrcFile* is called to extract the marker names and locations of the experimental data file for the corresponding optimization order.
5. The function *initOptStruct* is then called to initialize the optimization structure array of the specific joints and marker plates for the indicated optimization order.
6. Next, the function *getParams* is ran twice to determine the initial model parameter values for optimized parameters. The first time its ran its to obtain the optimization parameters for the joints and the second time its ran its to obtain the optimization parameters for the marker plates
7. The function *buildSymConstraints* is ran to identify the index pairs of the symmetry formed constraints for either rotation or translations (or both, and in one, two, or all three directions).
8. Matlab's least squares nonlinear function (LSQNONLIN) is ran using the *costFunction*.
    - All of the joint and marker plate parameters are varied by small amounts using the *getCaseNumber* and *modifyVariable* functions such that the marker errors from repeated OpenSim inverse kinematic analysis were minimized
        - These two functions are only called for the *setParams* function
    - The case number corresponds to the directions (x,y,z) for which the location or orientation parameters are to be optimized
    - The cost function included penalty terms that prevented large changes in joint and marker plate positions and orientations that would produce only small improvements in marker tracking
9. The new optimal parameters calculated are than changed in the model using the *setParams* function
10. The model is then rescaled using the *rescaleModel* function

- The *setParams* function is called once again within the *rescaleModel* function in order to reset the joints towards their optimal locations since scaling tends to move them in an undesired manner

11. The *calcMarkerErrors* function is lastly called in order to calculate the marker errors after the optimization run

Files Generated:

Patient4_optModel.osim