Manual SimTK optcntrlmuscle (v3.0)

Friedl De Groote, Maarten Afschrift, Tom Van Wouwe, Antoine Falisse

07/11/2018

Contents

1 Release notes

1.1 Release 3.0

- EMG-driven/EMG-constrained solutions of the muscle redundancy problem are enabled.
- Tracking of ultrasound (US) fiber length data is enabled.
- Muscle-tendon parameters can be additional variables in the muscle redundancy problem. This allows estimation of muscle-tendon parameters based on experimental data (kinematics and/or US and/or EMG).
- The estimation of the muscle-tendon parameters can be done using experimental data from different trials and motions all at once.
- Implementations using GPOPS-II and ADiGator are omitted. CasADi's 'opti stack' functionality is used to model the NLPs.
- Integration of the dynamic equations is done using Euler integration.
- We only provide implementations with muscle fiber length as state. Some changes in the formulation fixed some previous issues with the implementations with length as state compared to force as state.

- We provide plot functions to easily evaluate and compare simulations results.

2 Overview

2.1 Purpose of the software

The original intent of the provided MATLAB code was to solve the muscle redundancy problem using direct collocation as described in *De Groote F, Kinney AL, Rao AV, Fregly BJ. Evaluation of direct collocation optimal control problem formulations for solving the muscle redundancy problem. Annals of Biomedical Engineering (2016).* http://link.springer.com/article/10.1007%2Fs10439-016-1591-9.

From v3.0, there are possibilities to, concurrently with solvint the muscle redundancy problem, estimate parameters of the modelled muscle-tendon units by using collected EMG and ultrasound data. Optimal fiber length, tendon slack length and tendon stiffness can be set as free variables within the muscle redundancy problem. Experimentally measured fiber-lengths can be tracked (US-tracking), the tracking error is a part of the objective function. Collected EMG can either be tracked (EMG-tracking) or imposed exactly (EMG-driven). Another important feature is that the user can estimate muscle-tendon parameters over different trials of the same movement or from different movements. This allows to make estimation more reliable. We reckon that for solving the muscle redundancy problem OpenSim Moco might be a more user-friendly and straightforward alternative. However, our software allows the combination of different trials to estimate muscle-tendon parameters. Another difference is in that we use automatic differentiation, while this is not (yet) enabled in Moco. This software is still intended to solve inverse problems, for predictive simulations we refer to Moco or 'Antoine'.

From v2.1, CasADi can be used as an alternative to GPOPS-II and ADiGator. CasADi is an open-source tool for nonlinear optimization and algorithmic differentiation (https://web.casadi.org/). Results using CasADi and GPOPS-II are very similar (differences can be attributed to the different direct collocation formulations and scaling). We used CasADi's Opti stack, which is a collection of CasADi helper classes that provides a close correspondence between mathematical NLP notation and computer code (https://web.casadi.org/docs/#document-opti). CasADi is actively maintained and developed, and has an active forum (https://groups.google.com/forum/#!forum/casadi-users).

From v1.1, an implicit formulation of activation dynamics can be used to solve the muscle redundancy problem. Additionally, by using the activation dynamics model proposed by Raasch et al. (1997), we could introduce a nonlinear change of variables to exactly impose activation dynamics in a continuously differentiable form, omitting the need for a smooth approximation such as described in De Groote et al. (2016). A result of this change of variables is that muscle excitations are not directly accessible during the optimization. Therefore, we replaced muscle excitations by muscle activations in the objective function. This implicit formulation is described in *De Groote F, Pipeleers G, Jonkers I, Demeulenaere B, Patten C, Swevers J, De Schutter J. A physiology based inverse dynamic analysis of human gait: potential and perspectives F. Computer Methods in Biomechanics and Biomedical Engineering (2009).* http://www.tandfonline.com/doi/full/10.1080/10255840902788587. Results from both formulations are very similar (differences can be attributed to the slightly different activation dynamics models and cost functions). However, the formulation with implicit activation dynamics (De Groote et al., (2009)) is computationally faster. This can mainly be explained by

the omission of a tanh function in the constraint definition, whose evaluation is computationally expensive when solving the NLP.

2.2 Structure of the code

The code allows to solve three optimal control problems in the following order:

- Generic muscle redundancy problem: Solves the muscle redundancy problem using the provided musculoskeletal model, inverse kinematics and/or inverse dynamics data.
- Muscle parameter estimation: Solves the muscle redundancy problem where the variable space can be extended with user-specified muscle tendon parameters. Depending on availability the user can provide experimental muscle fiber length data to be tracked by simulated fiber lengths and/or EMG data to be tracked by simulated muscle excitations. The problem can be made EMG-driven as well, where the excitations will match the EMG signal up to a specified tolerance.
- Validation muscle redundancy problem: Solves the muscle redundancy problem using the optimized musculoskeletal model, inverse kinematics and/or inverse dynamics data. The idea of this simulation is to analyse whether the parameters (optimized by the estimation problem) predict musculoskeletal behaviour better than the generic model.

The user is off course free to select any of the three described problems.
Any questions? Please contact us: maarten.afschrift@kuleuven.be and tom.vanwouwe@kuleuven.be
for questions on the parameter optimization algorithm; friedl.degroote@kuleuven.be, antoine.falisse@kuleuven

3 Installation Instruction

Add the main folder and subfolder to your MATLAB path

```
addpath(genpath('C/...../SimTK_optcntrlmuscle'))).
```

Several software packages are needed to run the program

- The OpenSim MATLAB interface is used to generate the inputs to the optimal control problem based on a scaled OpenSim model and the solution of inverse kinematics (providing the solution of inverse dynamics is optional). To this aim, install OpenSim and set up the OpenSim MATLAB interface (OpenSim: https://simtk.org/frs/?group_id=91, OpenSim API: http://simtk-confluence.stanford.edu:8080/display/OpenSim/Scripting+with+Matlab).
- CasADi
  - CasADi is used for nonlinear optimization and algorithmic differentiation (https://web.casadi.org/).

4 Main Function

SolveMuscleRedundancy is the main function of this program and is used to solve up to three optimal control problems.

4.1 Input Arguments

Required input arguments for SolveMuscleRedundancy

1. **model_path**: directory and filename of the scaled OpenSim model (.osim file). The code should work with any OpenSim model with valid muscle-tendon parameters for which OpenSim's Inverse Dynamics and Muscle Analysis Tools generate reliable results. Note that only the muscle-tendon parameters and not the muscle model specified in the osim-file are used (for details see Muscle model).

2. **time**: 1 x 2 MATLAB array with the initial and final time of the analysis in seconds. Initial and final states influence the optimal controls over a period of about 50 ms at the beginning and end of the time interval over which the optimal control problem is solved. Since in practice the initial and final states are generally unknown, problems should be solved for a time interval containing five additional data points (considering a 100Hz sampling frequency) at the beginning and end of the motion cycle. Those additional data points should not be considered in further analyses. The user should thus not be surprised to observe unrealistically high muscle activation at the beginning of the motion (more details in companion paper).

3. **Out_path**: directory where you want to store the results from the muscle analysis.

4. **Misc**: miscellaneous input arguments

    - *DofNames_Input* is a cell array specifying for which degrees of freedom you want to solve the muscle redundancy problem. Typically the muscle redundancy problem is solved for one leg at a time (there are no muscles spanning both legs).
    - *MuscleNames_Input* is a cell array that specifies the muscles to be included when solving the muscle redundancy problem. All muscles that actuate (i.e. have a moment arm with respect to) the degrees of freedom specified in *DofNames_Input* will be selected by default if this array is left empty.
    - *IKfile*: array of filenames of the inverse kinematics solution of different motion trials (.mot file).
    - *IDfile*: array of filenames of the inverse dynamics solution of different motion trials (.sto file). If left empty, the inverse dynamics solution will be computed from the external loads (see Optional input arguments).
    - *EMGfile*: array of filenames containing EMG data of different motion trials (.mot file).
    - *USfile*: array of filenames containing fiberlength data of different motion trials (.mot file). The fiber length data is usually measured using ultra-sound (US).
    - *UStracking*: boolean to select whether you want to track provided muscle fiber lengths.
    - *EMGconstr*: boolean to select whether you want to track provided EMG signals.
    - *MRSbool*: boolean to select whether you want to solve the generic muslce redundancy problem.
    - *ValidationBool*: boolean to select whether you want to solve the validation muslce redundancy problem.

Optional input arguments for SolveMuscleRedundancy

1. **Misc**.*Loads_path*: directory and filename of the external loads (.xml file). The program will use the OpenSim libraries to solve the inverse dynamics problem when the required input argument ID_path is empty and Misc.*Loads_path* points to an external loads file.

2. **Misc**.*Estimate_TendonStiffness*: array with names of muscle from which tendon stiffness will be estimated.

3. **Misc**.*Coupled_TendonStiffness*: array with names of muscle from which tendon stiffness will be coupled. This means that the generic tendon stiffnesses of these muscles will be scaled with same variable.

4. **Misc**.*lb_kT_scaling*: lower bound of the scaling factor that will scale the generic tendon stiffness into the optimized tendon stiffness.

5. **Misc**.*ub_kT_scaling*: lower bound of the scaling factor that will scale the generic tendon stiffness into the optimized tendon stiffness.

6. **Misc**.*Estimate_OptFL*: array with names of muscle from which optimal fiber length will be estimated.

7. **Misc**.*Coupled_fiber_length*: array with names of muscle from which optimal fiber length will be coupled. This means that the generic fiber lengths of these muscles will be scaled with same variable.

8. **Misc**.*lb_lMo_scaling*: lower bound of the scaling factor that will scale the generic optimal fiber length into the optimized optimal fiber length.

9. **Misc**.*ub_lMo_scaling*: lower bound of the scaling factor that will scale the generic optimal fiber length into the optimized optimal fiber length.

10. **Misc**.*Coupled_slack_length*: For muscles from which the optimal fiber length is optimized, the tendon slack length will be optimized as well. Here you acn define an array with names of muscle from which tendon slack length will be coupled. This means that the generic fiber lengths of these muscles will be scaled with same variable.

11. **Misc**.*lb_lTs_scaling*: lower bound of the scaling factor that will scale the generic tendon slack length into the optimized tendon slack length.

12. **Misc**.*ub_lTs_scaling*: lower bound of the scaling factor that will scale the generic tendon slack length into the optimized tendon slack length.

13. **Misc**.*wlM*: cost function weighting factor for 'tracking fiber' lenghts term.

14. **Misc**.*wEMG*: cost function weighting factor for 'tracking EMG' term.

15. **Misc**.*wA*: cost function weighting factor for minimizing effort.

16. **Misc**.*wTres*: cost function weighting factor for minimizing reserve actuator contribution.

17. **Misc**.*wVm*: cost function weighting factor for minimizing muscle fiber velocities (term mainly for regularization of the optimization).

18. **Misc**.*Loads_path*: directory and filename of the external loads (.xml file). The program will use the OpenSim libraries to solve the inverse dynamics problem when the required input argument ID_path is empty and Misc.*Loads_path* points to an external loads file.

19. **Misc**.*ID_ResultsPath*: directory where the inverse dynamics results will be saved when the input argument *ID_path* is left empty.

20. **Misc**.*f_cutoff_ID*: cutoff frequency for the butterworth recursive low pass filter applied to the inverse dynamics data (default is 6 Hz).

21. **Misc**.*f_order_ID*: order of the butterworth recursive low pass filter applied to the inverse dynamics data (default is 6).

22. **Misc**.*f_cutoff_LMT*: cutoff frequency for the butterworth recursive low pass filter applied to the muscle tendon lengths from the muscle analysis (default is 6 Hz).

23. **Misc**.*f_order_LMT*: order of the butterworth recursive low pass filter applied to the muscle tendon lengths from the muscle analysis (default is 6).

24. **Misc**.*f_cutoff_dM*: cutoff frequency for the butterworth recursive low pass filter applied to the muscle moment arms from the muscle analysis (default is 6 Hz).

25. **Misc**.*f_order_dM*: order of the butterworth recursive low pass filter applied to the muscle moment arms from the muscle analysis (default is 6).

26. **Misc**.*f_cutoff_IK*: cutoff frequency for the butterworth recursive low pass filter applied to the inverse kinematics data (default is 6 Hz) when performing the muscle analysis to compute muscle-tendon lengths and moment arms.

27. **Misc**.*f_order_IK*: order of the butterworth recursive low pass filter applied to the inverse kinematics data (default is 6).

28. **Misc**.*Mesh_Frequency*: number of mesh interval per second (default is 100, but a denser mesh might be required to obtain the desired accuracy especially for faster motions).

29. **Misc**.*Atendon*: vector with tendon stiffness for the selected muscles. The order should correspond to *MuscleNames_Input*. The default value is 35 and a lower value corresponds to a more compliant tendon. The default value will be used when left empty. An example is provided in section **??** to set a different stiffness to the Achilles tendon.

4.2 Output arguments

We provide all state and control trajectories for the different trials and optimal control problems in one Results structure. Trajectories for different trials and optimal control problems are divided in substructures. All trajectories are interpolated on the mesh points.

1. Time: time vector

2. MExcitation: muscle excitation

3. MActivation: muscle activation

4. RActivation.meshPoints: activation of the reserve actuators

5. TForcetilde: normalized tendon force

6. TForce: tendon force

7. Fpe: passive muscle force

8. lMTinterp: muscle tendon length
9. lMtildeopt: normalized muscle fiber length
10. lm: muscle fiber length
11. MvM: normalized muscle fiber velocity
12. FMltilde: force-length multiplier
13. FMltilde: force-velocity multiplier
14. Param: scaling factors for the different optimized parameters
15. Misc: for reference of the settings of the simulation we add the Misc structure to the results.

5 Muscle model

The musculotendon properties are fully described in the supplementary materials of the aforementioned publication. Importantly, only the tendon slack length, optimal muscle fiber length, maximal isometric muscle force, optimal pennation angle and maximal muscle fiber contraction velocity are extracted from the referred OpenSim model. Other properties are defined in the code and can be changed if desired. By default, the activation and deactivation time constants are 15 and 60 ms respectively (see tau_act and tau_deact in SolveMuscleRedundancy_$<$ *state* $>$.m).

6 Examples