The Journal of Open Source Software

MocoExtendProblem: Interface Between OpenSim and MATLAB for Rapidly Developing Direct Collocation Goals in Moco

Aravind Sundararajan [1¶], **Varun Joshi** [2], **Brian Umberger** [2], and **Matthew O'Neill** [1]

1 Department of Anatomy, Midwestern University, Glendale Arizona, USA **2** School of Kinesiology, University of Michigan, Ann Arbor, Michigan, USA ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

optimal control problems using OpenSim Moco (Dembia et al., 2020) and MATLAB (The Mathworks, Inc., Natick, MA, USA). MEP features several templates for testing and prototyping novel MocoGoals in lieu of rebuilding OpenSim or generating an .omoco file from C++ to load the problem into MATLAB. Instead, users structure custom goals, build them, and call custom goals from MATLAB scripts.

This repository features:

- A build.m script that compiles goals in the custom_goals directory and procedurally constructs the C++/MATLAB class implementations and compiles the MEX interface.
- Compatibility tested with OpenSim 4.2-4.5.
- OpenSim versions lower than 4.5 require unique modifications to the build pipeline since booleans for division by duration, distance and mass were migrated to the abstract MocoGoal.
- The ability to include MEP as a submodule, build, and use valid custom goals.
- Three example custom goals in the custom_goals and custom_goals_compat directories.

Statement of need

OpenSim is an open-source software platform for modeling musculoskeletal structures and creating dynamic simulations of movement (Seth et al., 2018). OpenSim enables researchers and clinicians to investigate how biological and non-biological structures respond to different loads, postures and activities in both static and dynamic situations. OpenSimhas been used to study a wide range of biomechanical problems, such as the mechanics of walking and running (e.g. Falisse et al., 2019), the impact of injury or disease on movement (e.g. Johnson et al., 2022), and the effectiveness of rehabilitation exercises (e.g. Spomer et al, 2023).

OpenSim Moco (Dembia et al., 2020) employs an optimization paradigm called direct collocation to solve trajectory optimization problems that range from solving for muscle forces, to tracking experimental data, and fully predictive simulations. Direct collocation is a numerical optimal control method (Kelly, 2017) that is computationally efficient and is used extensively in computational approaches to understanding biological movement . While direct collocation is powerful, Moco only provides a fixed set of optimization goals. It can be daunting for many users to develop custom goals in C++. We developed MEP so Moco users without experience compiling C++ can still write and test custom goals. The OpenSim interfaces are created with SWIG, as opposed to MEX, which can be daunting for even experienced biomechanists.

MocoExtendProblem was developed using MATLAB versions 2022a. Running build.m will compile MocoGoals in the custom_goals directory, or in the custom_goals_compat directory for OpenSim versions pre-4.5.

Direct collocation is a numerical optimization method used in dynamic systems and control engineering. It involves representing the system dynamics as a set of algebraic equations, which are then discretized over time, and solved as a nonlinear optimization problem to obtain the optimal control inputs. The method aims to find a numerical solution that satisfies the system constraints and optimizes a performance measure. Optimization paradigms like direct collocation have begun to play a critical role in expanding our understanding of biological locomotion through the in-silico testing of novel therapies and predictive capabilities.

Within OpenSim is the software toolkit Moco (Dembia, 2021), which employs direct collocation with the interior-point optimizer IPOPT (Wächter & Biegler, 2006) in order to solve trajectory optimization problems that could range from tracking experimental motion capture data for solving generalized coordinates, actuator controls, and kinetics to fully predictive simulations. Moco employs CasADi (Andersson et al., 2018) to transform MocoProblems consisting of control goals and constraints into sets of matrices for nonlinear optimization. While direct collocation is powerful and OpenSim can be used to generate a broad range of dynamically-consistent simulations, it can be daunting for some users to modify and rebuild novel direct collocation goals.

We developed `MEP` so researchers, clinicians, and students without experience compiling C++ can still write and test custom goals. By contrast, OpenSim's interfaces for MATLAB are developed using SWIG, as opposed to MEX, which can be daunting for even seasoned biomechanists. Running build.m will compile custom goals developed and placed in the custom_goals directory, or if using OpenSim 4.5, `MEP` will search the custom_goals45. This distinction for pre- and post- 4.5 `MEP` is to handle where scaling arguments are moved to the abstract MocoGoal.

CMake and msbuild from Visual Studio 2019 or higher must be added to the system PATH. build.m will procedurally construct both extend_problem.m and ExtendProblem.cpp by parsing the header files of the discovered goals within the custom_goals directory. Both ExtendProblem.cpp and extend_problem.m generate bindings to instantiate custom goals placed in the custom_goals directory. Custom goals can be compiled with Visual Studio 2019 or higher and then MATLAB's MEX compiler is used to compile ExtendProblem. ExtendProblem.cpp leverages the C++ library mexplus (Yamaguchi, 2014) to gain access to MEX entry points through C++ macros.

```
MocoExtendProblem
├─bin
│ └─RelWithDebInfo
│     ├─ExtendProblem.cpp
│     ├─extend_problem.m
│     └─extendProblem.mexw64
├─build
├─custom_goals or custom_goals_compat
│ └─MocoExampleGoal
│     ├─MocoExampleGoal.cpp
│     ├─MocoExampleGoal.h
│     ├─osimMocoExampleGoalDLL.h
│     ├─RegisterTypes_osimMocoExampleGoal.cpp
│     └─RegisterTypes_osimMocoExampleGoal.h
├─test
├─utils
│ ├─build_extend_class.m
│ ├─construct_goal_tree.m
│ ├─cpp_end.m
│ ├─cpp_start.m
│ ├─generate_content.m
│ ├─generate_content_compat.m
│ ├─get_goal_names.m
│ ├─get_setter_functions.m
│ ├─mex_end.m
│ ├─mex_start.m
│ ├─wrap_end.m
│ └─wrap_start.m
└─build.m
```

To create a new goal with MEP: 1 OpenSim 4.5+ users should copy a goal in the custom_goals directory while 4.2-4.4 users should copy a goal in custom_goals_compat. 2 Replace mentions of the original goal name to that of your new custom goal name in each of the 5 files and file names, being careful to also modify the include guards in the dll and register types header files. 3 Reimplement constructProperties(), initializeOnModelImpl(), calcIntegrandImpl(), calcGoalImpl() such that they describe your custom goal.

To incorporate extend_problem goals into an existing MATLAB script, a C-style pointer to the instantiated MocoProblem is passed as a constructor argument to the extend_problem.m class that wraps the MEP MEX. Class methods of extend_problem.m (Figure 1; blue) are then used to add custom goals to the MocoProblem.

```
cptr = uint64(problem.getCPtr(problem));
ep = extend_problem(cptr);
ep.addMocoCustomGoal('custom_goal',weight,power,divide_by_distance);
```

This paradigm has implications for OpenSim and MATLAB developers beyond the scope of just incorporating novel MocoGoals; these same tools can be used to extend other classes and easily incorporate them into existing MATLAB-OpenSim scripts. We have posted all tools, instructions and simulation results related to this project on GitHub and SimTK.org (simtk.org/projects/moco-ep).

Requirements

- Download and install OpenSim from https://simtk.org and follow the documentation for setting up OpenSim's MATLAB scripting environment.
- Follow the instructions (OpenSim) to download necessary dependencies for both scripting in MATLAB and C++ development.
- In MATLAB, configure MEX with mex -setup C++ to use the MS VisualStudio 2019+.

Showcases

To demonstrate the utility of this framework, we generated a two-dimensional (2-D) walking simulation using the MATLAB-OpenSim API (Denton and Umberger, 2023). The base code uses the built-in MocoControlEffortGoal and MocoAverageSpeedGoal to generate tracking and predictive simulations of minimum effort walking at an average speed of 1.3 m s-1. Additionally, each objective function includes implicit acceleration and auxiliary derivative terms that are minimized to ensure smooth trajectories.

Since Moco lacks built-in gait stability goals, we developed three stability goals using MEP build.m to create an ExtendProblem class that adds these to an existing MocoProblem (Figure 1 blue). The first is a base of support (Equation 1 BOS) criterion in which the whole-body center of mass (COM) is optimized to lay between the two hindfeet COMs projected to the ground reference frame, the second is a zero-moment-point goal (Equation 2 ZMP) where the center of mass tracks the computed zero-tilting moment location, and the third is a marker acceleration minimization goal (Equation 3 ACCmarker) that minimizes the explicit accelerations of a marker placed on the head (marker location is arbitrary and can be set by the user).

MEP's build.m was used to generate an ExtendProblem class that adds these new stability cost terms:

$$J_{BOS} = W_1 EFF^2 + W_2 ACC_{smoothing} + W_3 BOS \tag{1}$$

$$J_{zmp} = W_1 EFF^2 + W_2 ACC_{smoothing} + W_3 ZMP \tag{2}$$

$$J_{acc} = W_1 EFF^2 + W_2 ACC_{smoothing} + W_3 ACC_{marker} \tag{3}$$

The results of each multi-objective predictive simulation, in which the stability criterion was compiled using MEP, is shown against the results from a tracking simulation (Figure 2, Table 1) that closely-matched experimental data (Denton and Umberger, 2023). As the purpose was to demonstrate the utility of MEP, we did not tune the stability term weights to match the tracking result as closely as possible.

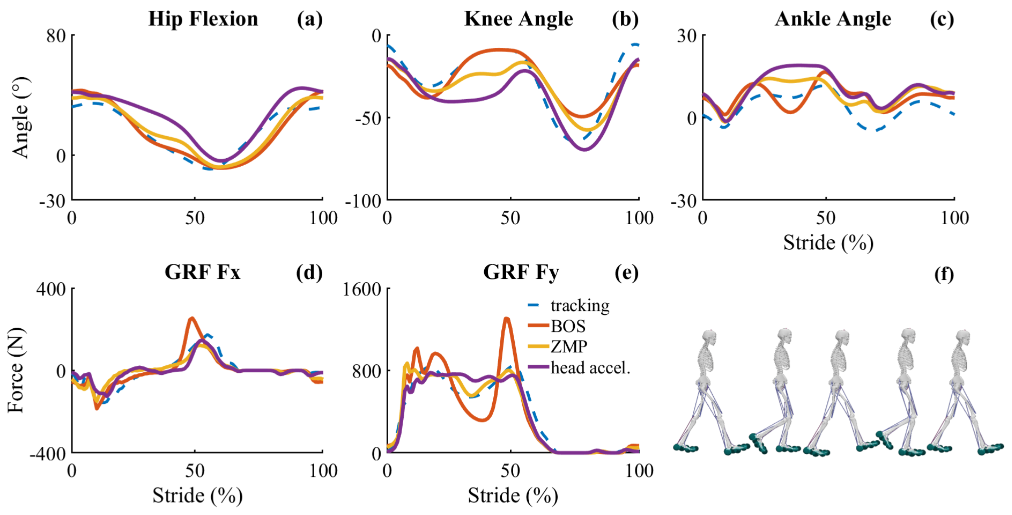

Figure 1: Sagittal plane hip, knee and ankle angles (a-c), vertical and A-P ground reaction forces (d-e), the 11 degree-of-freedom, 18 muscle sagittal plane human walking model used for tracking and predictive simulations (f)

Table 1: Objective cost and term breakdown for three predictive simulations using MEP.

	Objective cost	Effort cost	Smoothing cost	Stability cost
J_{BOS}	3.759046	2.270912	0.683608	0.794155
J_{ZMP}	4.184254	2.751212	0.725837	0.686290
J_{accel}	4.774932	3.797785	0.793123	0.174308

While these examples used planar gait simulations, MEP is agnostic to model complexity or task, and is being used successfully in our ongoing research (e.g. Joshi et al. 2022, Sundararajan et al. 2023) of locomotor performance in humans and other animals. GNU Octave support would require minimal syntactical modification. An additional benefit of sequestering novel goals into ExtendProblem is being able to back-port goals from a newer OpenSim version to an older version (i.e. taking a goal from OpenSim 4.4 and bringing that functionality to 4.2). Ultimately, MEP offers a modular framework to rapidly develop, test and compare novel MocoGoals for features beyond OpenSim Moco's current scope.

#Funding

This work was supported by the National Science Foundation (BCS 2018436 and BCS 2018523)

References

Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B., & Diehl, M. (2018). CasADi: A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, *11*, 1–36. https://api.semanticscholar.org/CorpusID:53559074

Dembia, N. A. A. F., C. L. AND Bianco. (2021). OpenSim moco: Musculoskeletal optimal control. *PLOS Computational Biology*, *16*(12), 1–21. https://doi.org/10.1371/journal.pcbi.1008493

Wächter, A., & Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, *106*, 25–57. https://api.semanticscholar.org/CorpusID:14183894