SimTK Documents

OpenSim Advanced User & Developer Workshop

August 15-17, 2011, Stanford University

Websites: SimTK.org/home/opensim and opensim.stanford.edu

OpenSim Workshop Agenda

Day One – Monday, August 15, 2011
Li Ka Shing Center, Room 102, Stanford University

8:30 – 9:00am	Welcome, Workshop Goals, and Meet the OpenSim Team *Scott Delp and Jen Hicks*
9:00 – 10:15am	Participant Introduction and Goals *You: Each presenter will be limited to 2 min + 1 min Q&A*
10:15 – 10:30am	BREAK
10:30 – 12:00pm	Generating Forward Simulations with Residual Reduction and Computed Muscle Control: Theory, Best Practices, and Hands-On Example *Ajay Seth and Sam Hamner*
12:00 – 1:00pm	LUNCH
1:00 – 2:00pm	Components of an OpenSim Model with a Hands-On Example *Matt DeMers*
2:00 – 2:15pm	BREAK
2:15 – 2:30pm	Solidify Project Plans
2:30 – 5:00pm	Work on Projects
6:00pm	Informal Social Outing in Downtown Palo Alto

Day Two – Tuesday, August 16, 2011
Li Ka Shing Center, Room 102, Stanford University

8:30 – 9:00am	Batch Processing and Data Management *Edith Arnold*
9:00 – 10:00am	Breakout Session: Hands-On Introduction to the OpenSim API *Ajay Seth and Marjolein van der Krogt*
8:30 – 12:00pm	Work on Projects
12:00 – 1:00pm	LUNCH
1:00 – 1:30pm	Discussion of Common Issues *OpenSim Team and You*
1:30 – 5:00pm	Work on Projects

Day Three – Wednesday, August 17, 2011
Li Ka Shing Center, Room 102, Stanford University

8:30 – 12:00pm	Work on Projects
12:00 – 1:00pm	LUNCH
1:00 – 2:00pm	Prepare Project Presentations
2:00 – 3:45pm	Presentation of Progress, Hurdles, and Future Plans *You*
3:45 – 4:00pm	Closing Remarks *Scott Delp and Jen Hicks*
4:00 – 5:00pm	RECEPTION

Acknowledgments

OpenSim was developed as a part of SimTK and funded by the Simbios National Center for Biomedical Computing through the National Institutes of Health and the NIH Roadmap for Medical Research, Grant U54 GM072970. Information on the National Centers can be found at http://nihroadmap.nih.gov/bioinformatics.

Trademarks and Copyright and Permission Notice

Table of Contents

1 Introduction

1.1 Getting the Most Out of an OpenSim Workshop

The OpenSim team at Stanford puts many hours into preparing for workshops and developing the software, documentation, and examples. As participants, you've put many hours into collecting and analyzing your data and now have traveled from around the world to spend three days working on your projects. There are several guidelines we can follow to ensure that everyone gets a maximum benefit from the workshop:

- Use the didactic lectures, handouts, and online OpenSim resources we've provided as the first step for resolving problems.
- Work together! The participant sitting next to you might be able to answer your question just as well or better than a member of the Stanford team. Included with your handout materials is a list of all of the workshop attendees and their project topics.
- There will be many Stanford graduate students and staff researchers available to help answer questions during the workshop. Everyone has different areas of expertise. Please see the see the workshop leaders to find where best to direct your questions.
- Have fun and take breaks. We've purposely included breaks and time for social interaction and ask that you follow this part of the schedule. Taking the time to rest and recharge is essential for everyone.
- Set a clear and manageable project goal for the workshop. This is the purpose of preparing goals slides and the related pre-workshop interaction.
- Share your results. Create a project on SimTK.org to share your models and simulation results at the end of the workshop, if you haven't done so already. Documenting your work will allow other researchers to build on your findings and give you credit for the discoveries you've made in your research.
- Teach others. We hope you will share what you learn at the workshop with your students and colleagues. Please contact us if you are interested in starting an OpenSim user group or leading a workshop at your local institution.

- Fill out our online survey to give us feedback and help us improve OpenSim and future workshops.

1.2 Where to Find Additional Resources and Support

There are many resources available to help with troubleshooting, access models and simulation data, and interact with the rest of the OpenSim community.

1.2.1 The OpenSim GUI

The OpenSim Help menu provides the following resources:

- Direct links for filing a bug or requesting a new feature.
- Direct links to three tutorials for becoming familiar with the OpenSim GUI.
- A "Convert Files..." utility for converting older OpenSim model and setup file formats to the latest version.
- An "Available Objects..." option that opens a panel that lists all the model components, analyses, and tools that are available in OpenSim and lists their setup properties that specify the behavior of these objects.

1.2.2 OpenSim User's Guide

The User's Guide is an extensive resource for most of the features available in OpenSim. For assistance using the GUI and learning about setting up Tools in OpenSim, please refer to the OpenSim User's Guide that is available under "OpenSim Documentation" at https://simtk.org/docman/?group_id=91.

1.2.3 Model and Simulation Repository

You can create your own models of musculoskeletal structures and dynamic simulations of movement in OpenSim, as well as take advantage of computer models and dynamic simulations that other users have developed and shared. For example, you can use existing computer models of the human lower limb, upper limb, cervical spine, and whole body, which have already been developed and posted at https://simtk.org/home/nmblmodels. You can also use dynamic simulations of walking and other activities that have been developed, tested, and posted on SimTK.org. We encourage you to share your models and simulations with the research community by setting up a project on SimTK.org.

1.2.4 Additional Online Resources

There are many resources available online for support both during and after the workshop. This includes the User's Guide and model repository described above, as well as a user forum, tutorials, examples, webinars, and many other resources. We've recently launched a new website that serves as a portal to these resources. It can be found at http://opensim.stanford.edu. You can also find more about OpenSim and related projects by other researchers around that world at the SimTKorg project site at http://simtk.org/home/opensim.

1.2.5 Publications

You can find additional information in the following article: Delp, S.L., Anderson, F.C., Arnold, A. S., Loan, P., Habib, A., John, C., Guendelman, E.G., Thelen, D.G., OpenSim: Open-source software to create and analyze dynamic simulations of movement. *IEEE Transactions on Biomedical Engineering*, vol. 54, no. 11, pp. 1940-1950, 2007. Please cite this work in any of your own publications that use OpenSim.

2 The OpenSim Workflow

OpenSim has a broad range of capabilities for generating and analyzing musculoskeletal models and dynamic simulations. This chapter provides an overview of these capabilities and a list of resources to find more information about each component of the OpenSim workflow.

2.1 The OpenSim Model

One of the major goals of the OpenSim project is to provide a common platform for creating and sharing models of the musculoskeletal system. Thus the first component of any analysis is an OpenSim model. An OpenSim model represents the dynamics of a system of rigid bodies and joints that are acted upon by forces to produce motion. The OpenSim model file is made up of components corresponding to parts of the physical system. These parts include bodies, joints, forces, constraints, and controllers (Figure 2-1).

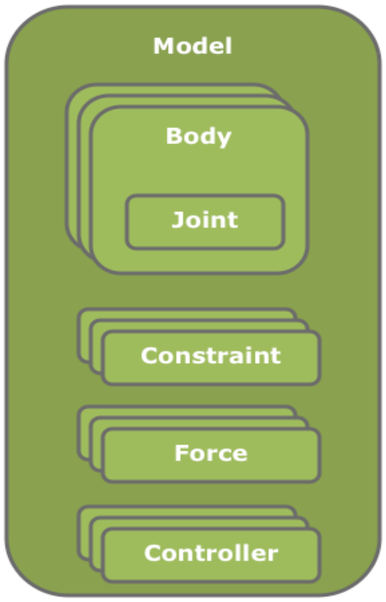


Figure 2-1 A conceptual representation of an OpenSim model and its components

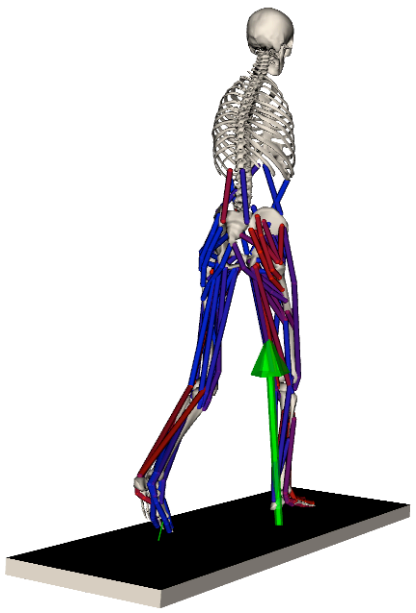


Figure 2-2 An OpenSim model for simulation of gait

For example, in a model used for simulation of human walking (Figure 2-2), the bodies represent the geometry and inertial properties of the body segments. The joints specify the articulations at the pelvis, hip, knee, and ankle joints, while a constraint could be used, for example, to couple the motion of the patella with the model's knee flexion angle. The forces in the model include both internal forces from muscles and ligaments and external forces from interaction with the ground. Finally, the model's controller determines the activation of muscles (e.g. computed muscle control).

Chapter 10 of this handout contains more information about the components of an OpenSim model and Chapter 11 describes an example of editing an existing model. Additional information is also available in the OpenSim User's Guide (Chapter 24). A large repository of existing models is available at SimTK.org (https://simtk.org/home/nmblmodels). This library includes models of the lower extremity, head and neck, spine, and wrist. We encourage you to contribute your own models to this library to enable other researchers to build on your work and further advance the field.

2.2 Importing Experimental Data

In many cases, you will use OpenSim to analyze experimental data that you have collected in your laboratory. This data typically includes:

- Marker trajectories or joint angles from motion capture
- Force data, typically ground reaction forces and moments and/or centers of pressure
- Electromyography

Chapter 23 of the User's Guide contains detailed information about preparing and importing your experimental data. Chapter 3 of this handout describes how to preview and verify your data.

2.3 Scaling

If you are using a generic model from the existing library of models, the next step is to scale the model to match the experimental data collected for your subject, functionality provided by the Scale tool in OpenSim. The purpose of scaling a generic musculoskeletal model is to modify the anthropometry, or physical dimensions, of the generic model so that it matches the anthropometry of a particular subject. Scaling is one of the most important steps in solving inverse kinematics and inverse dynamics problems because these solutions are sensitive to the accuracy of the scaling step. In OpenSim, the scaling step adjusts both the mass properties (mass and inertia tensor), as well as the dimensions of the body segments.

Chapter 4 of this handout provides best practices and trouble-shooting tips for using the Scale tool. In addition, there is detailed documentation of the tool in Chapter 14 of the OpenSim User's Guide. OpenSim Tutorial #3, available from the Downloads page of the OpenSim project site on SimTK.org includes an example using the Scale tool. This tutorial is also accessible from the OpenSim application Help menu.

2.4 The Inverse Problem

OpenSim enables researchers to solve the Inverse Dynamics problem, using experimental measured subject motion and forces to generate the kinematics and kinetics of a musculoskeletal model (Figure 2-3).

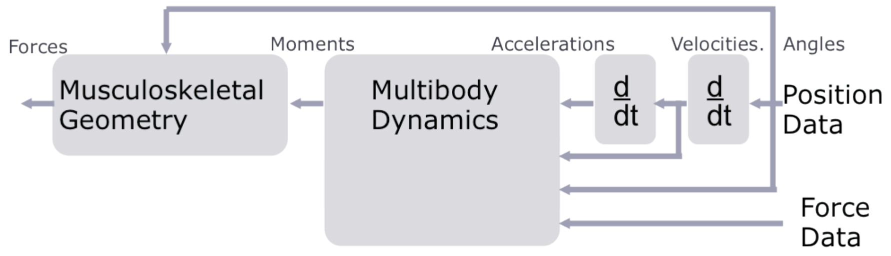


Figure 2-3 The Inverse Dynamics Problem. In inverse dynamics, experimentally measured marker trajectories and force data are use to estimate a model's kinematics and kinetics.

2.4.1 Inverse Kinematics

The Inverse Kinematics (IK) Tool in OpenSim finds the set of generalized coordinates (joint angles and positions) for the model that best match the experimental kinematics recorded for a particular subject (Figure 2-4). The experimental kinematics targeted by IK can include experimental marker positions, as well as experimental generalized coordinate values (joint angles). The IK tool goes through each time step of motion and computes generalized coordinate values which positions the model in a pose that "best matches" experimental marker and coordinate values for that time step. Mathematically, the "best match" is expressed as a weighted least squares problem, whose solution aims to minimize both marker and coordinate errors.

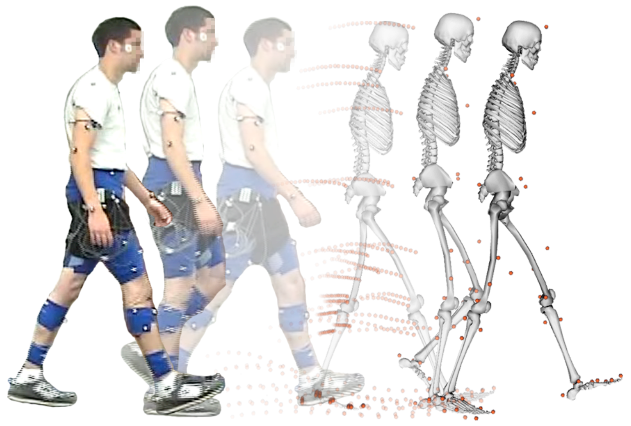


Figure 2-4: Inverse Kinematics Tool Overview. Experimental markers are matched by model markers throughout the motion by varying the generalized coordinates (e.g., joint angles) through time.

Some best practices and troubleshooting tips for using the IK tool are included in Chapter 5 of this handout. In addition, Chapter 15 of the User's Guide contains the full documentation for running IK in OpenSim. Tutorial #3, available on the downloads page of the OpenSim project site on SimTK.org and from the OpenSim application's Help menu walks through an example of using Inverse Kinematics for human walking.

2.4.2 Inverse Dynamics

Dynamics is the study of motion *and* the forces and moments that produce that motion. The Inverse Dynamics (ID) tool determines the generalized forces (e.g., net forces and torques) that cause a particular motion, and its results can be used to infer how muscles are utilized for that motion. To determine these internal forces and moments, the equations of motion for the system are solved with external forces (e.g., ground reactions forces) and accelerations given (estimated by differentiating angles and positions twice). The equations of motion are automatically formulated using the kinematic description and mass properties of a musculoskeletal model in Simbody™.

Some best practices and troubleshooting tips for using the ID tool are included in Chapter 6 of this handout. In addition, Chapter 16 of the User's Guide contains the full documentation for running ID in OpenSim. Tutorial #3, available on the downloads page of the OpenSim project site on SimTK.org and from the OpenSim application Help menu walks through an example of using ID for human walking.

2.4.3 Static Optimization

Static optimization is an extension of inverse dynamics that further resolves the net joint moments into individual muscle forces at each instant in time based on some performance criteria, like minimizing the sum of squared muscle forces. More information about the Static Optimization tool available in OpenSim can be found in Chapter 8 of this handout and Chapter 17 of the OpenSim User's Guide.

2.5 The Forward Problem

OpenSim is also capable of generating muscle-driven forward simulations of gait and other movements (Figure 2-5).

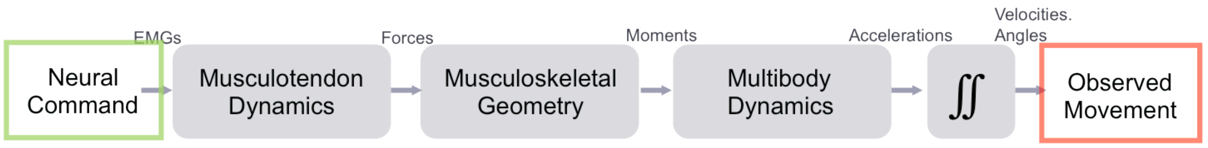


Figure 2-5 The forward dynamics problem. In a forward dynamic simulation of motion, simulated muscle excitations are used to drive the motion of a model to follow some observed movement.

The Forward Dynamics tool takes a set of controls (e.g., muscle excitations) to drive a model's motion by integrating forward in time. Typically, muscle excitations are generated using the Computed Muscle Control (CMC) tool. As a pre-cursor to running CMC, the Residual Reduction Algorithm (RRA) is used to minimize the effects of modeling and marker data processing errors that aggregate and lead to large nonphysical compensatory forces called residuals. Specifically, RRA alters the torso mass center of a subject-specific model and permits the kinematics of the model from inverse kinematics to vary in order to be more dynamically consistent with the ground reaction force data. Thus the typical workflow for generating a muscle-driven simulation after importing experimental data is Scale->IK->RRA->CMC->Forward Dynamics (Figure 2-6).

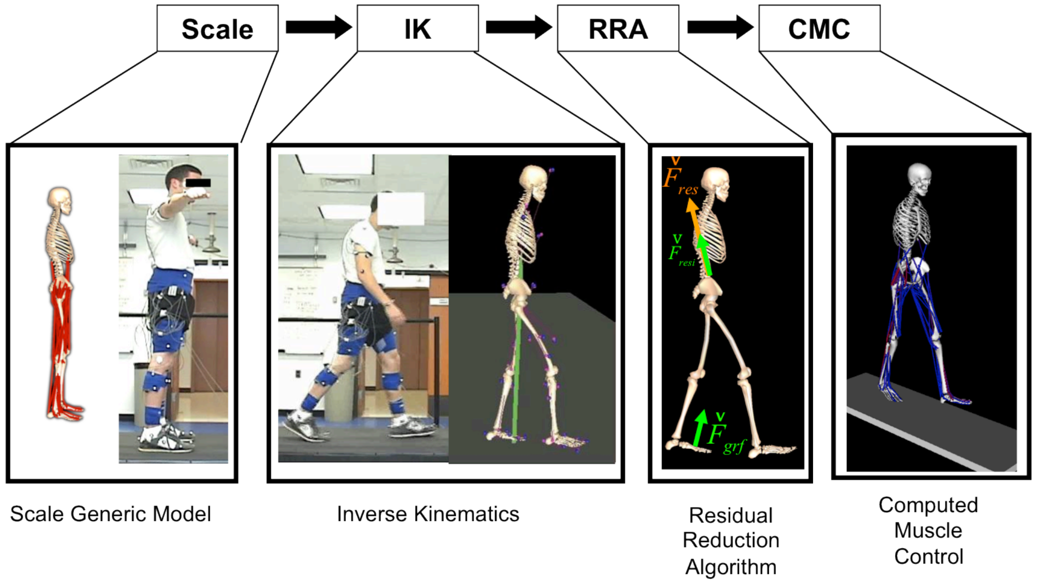


Figure 2-6 Overview of the workflow for generating a muscle-actuated simulation in OpenSim.

Best practices and troubleshooting tips for Forward, RRA, and CMC are included in Chapters 9, 10, and 11 of this handout. Full documentation of Forward Dynamics, RRA, and CMC is available in Chapters 18 through 20 of the User's Guide.

2.6 Analyzing Simulations

Often, answering your research questions requires delving deeper into the details of a simulation. Thus OpenSim includes an Analyze tool that allows you to estimate, for example, muscle fiber or tendon lengths during a motion or the loads on the knee joint. The Analyze Tool enables you to analyze a model or simulation based on a number of inputs that can include time histories of model states, controls, and external loads applied to the model. The following analyses are available in OpenSim:

1. **Body Kinematics**: Reports the spatial kinematics (position and orientation, linear and angular velocity, linear and angular acceleration) of specified bodies for the duration of the analysis.
2. **Point Kinematics**: Reports the global position, velocity and acceleration of a point defined local to a body during a simulation.
3. **Muscle Analysis**: Reports all attributes of all muscles. This includes: fiber length and velocity, normalized fiber length, pennation angle, active-fiber force, passive-fiber force, tendon force, and more.

4. **Joint Reactions**: Reports joint reaction forces. These are forces that enforce the motion of the joint. The force applied to either parent or child and expressed in ground, parent or child can be reported.
5. **Induced Acceleration:** Computes accelerations caused or "induced" by individual forces acting on a model, for example, the contribution of individual muscle forces to the mass center acceleration.
6. **Force Reporter:** Reports all forces acting in the model. For ligaments and muscles, the tension along the path is reported and for ideal actuators the scalar force or torque is reported. For all other forces, the resultant body forces (force and moment acting at the center of mass of the body) are reported. For example, contact forces from an ElasticFoundationForce element yields the resultant body force on the contacting bodies separately, expressed in ground. For constraints, the same is true, except the forces are expressed in the most distal common ancestor body. Whenever a constraint involves ground, this is the ground body; however, if for example a model of the arm has a hand with fingers touching via a point constraint, then the forces are expressed in the nearest common ancestor, which would be the palm (if modeled as a single body).

More details about the analyses available in OpenSim are available in Chapter 21 of the User's Guide. Chpater 22 contains information about conducting an Induced Acceleration Analysis in OpenSim.

3 Previewing Mocap Data

3.1 Tips and best practices for collecting experimental data

The first step for most researchers is collecting experimental data in the form of marker trajectories, ground reaction forces, and EMG. There are several key steps you can take to ensure that your motion capture data is as accurate as possible and represented correctly in OpenSim:

- Develop and document lab protocols and standards for your marker sets, camera positions, and force plate/camera system coordinate frames.
- Calibrate center-of-pressure measurements with marker positions using a calibration “T” to pinpoint where on the force-plate the point load (lowest tip of the “T”) is being applied. If the center-of-pressure calculated from the force-plate does not match the “T” location from markers (within marker resolution), you need align the force-plate and marker mo-cap reference frames.
- Digital cameras and camcorders are cheap! Take lots of photos/video during experiments so that you can verify marker placement and other factors for the data you collect.

3.2 Using the data previewer

After performing experiments, motion capture data can be previewed in the OpenSim GUI to verify that preprocessing was done correctly and that data is in agreement with the intended model. If users have multiple files representing different pieces of data, this tool allows users to synchronize data to verify that the data was transformed consistently. The data previewer handles two types of data: marker trajectories (contained in .trc files) and measured forces (e.g., ground reactions, contained in .mot files).

Select **Preview Motion Data…** from the **File** menu. Then browse to select the motion capture data file to be visualized. Once selected, OpenSim performs the following actions:

- Adds a new model to the Navigator with the default name of “ExperimentalData” and a unique number, such as “01”.
- Makes the loaded motion capture data file the *current* motion.

This enables the user to perform the following tasks:

- Use the motion slider in the toolbar to visualize different frames of data.
- Synchronize the motions from different motion files. These other motions could be either motion capture files or results from OpenSim tools. For example, experimentally measured marker trajectories and ground reaction forces can be synchronized and superimposed on the result of a forward simulation.

3.3 Visualizing marker trajectories in OpenSim

In OpenSim, marker trajectories are contained in a .trc file. When loaded with the data previewer, a model named "ExperimentalData" will contain the loaded motion file consisting of the marker data (e.g. "subject01_walk1.trc"). The "Markers" node lists each individual marker found in the file (Fig. 3-1). Nodes corresponding to individual markers have the following options (access options by *right clicking* the marker name):

- **Show**: Enabled only if a marker is hidden
- **Show Only**: Hides all other markers except for those selected.
- **Toggle Trail Display**: Toggles the display of a line representation of the trajectory of selected marker(s).

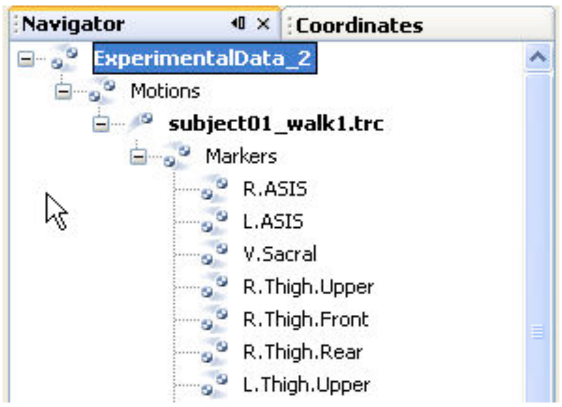


Figure 3-1. Navigator view of trajectory data file

3.4 Visualizing external forces in OpenSim

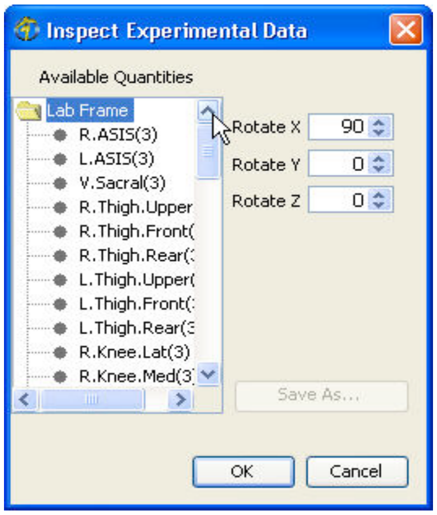
In OpenSim, external forces (e.g., ground reaction forces) are contained in a .mot file. The header of the selected force (.mot) file is the same as that expected by OpenSim tools (see Section 22.5 of the User's Guide for more detail). For the ground reaction forces used by the gait model, the column labels are shown below (for 2 external forces force1, force2, torque1, torque2):

ground_force1_vx, ground_force1_vy, ground_force1_vz,
ground_force1_px, ground_force1_py, ground_force1_pz,
ground_force2_vx, ground_force2_vy, ground_force2_vz,
ground_force2_px, ground_force2_py, ground_force2_pz,
ground_torque1_x, ground_torque1_y, ground_torque1_z,
ground_torque2_x, ground_torque2_y, ground_torque2_z

The data previewer expects groups of 6 columns for a force of the form "body"_"ForceName"_{vx, vy, vz, px, py, pz}, where *vi* corresponds to each component of the force and *pi* corresponds to each component of the location of the force (e.g., center of pressure) and 3 columns for a torque of the form "body"_"TorqueName"_{x, y, z}, corresponding to each component of the applied torque. Note that this naming convention is only necessary for previewing purposes. OpenSim tools employ a new user interface enabling you to specifiy any number of forces, along with points of application, or torques to a model during any simulation or analysis.

3.5 Previewing transformed data

For any kind of previewed data (e.g., marker trajectories or ground reaction forces), you can visualize the effect of a rigid-body-transform applied to the data and save the data as a new file. However, the visualization is transformed to enable users to adapt preprocessing tools accordingly. The visual transformation dialog is available as a context menu (Fig. 3-2) by right clicking on the node with the data file name.



Rotate X 90
Rotate Y 0
Rotate Z 0

Figure 3-2. Transform data previewer

Arrow buttons change the angle for “Rotate X”, “Rotate Y”, and “Rotate Z” that transform the data in the viewer. Angles are always positive (0-360°). Translations can be previewed by modifying the display offset. The data file is NOT modified by OpenSim, but provides a preview so that the appropriate transformation(s) may be determined and applied during preprocessing.

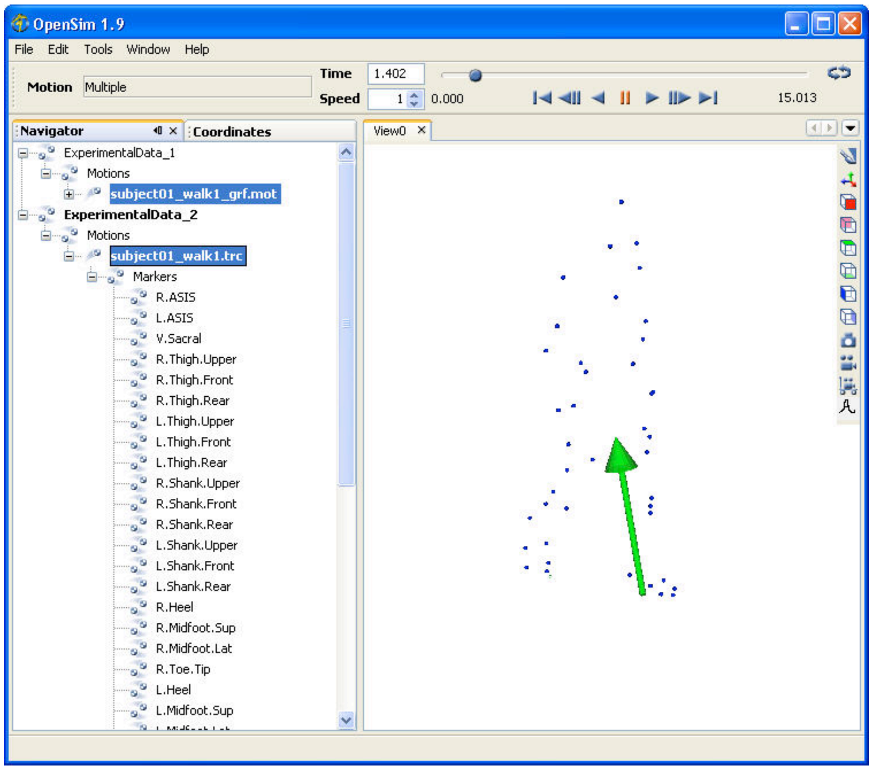


Figure 3-3. Preview of motion capture data in OpenSim

4 Scaling

4.1 How it Works

The Scale Tool alters the anthropometry of a model so that it matches a particular subject as closely as possible. Scaling is typically performed based on a comparison of experimental marker data with virtual markers placed on a model. In addition to scaling a model, the scale tool can be used to adjust the locations of virtual markers so that they better match the experimental data.

In this chapter, we provide a conceptual review of the inputs and outputs of the Scale tool and a set of troubleshooting tips and best practices for scaling. The OpenSim User's Guide provides detailed information and step-by-step instructions on scaling a model (Chapter 14). Carefully scaling your model to match your subject is essential for getting good results from later tools, like Inverse Kinematics and Inverse Dynamics.

4.2 Scale Tool

The Scale Tool is accessed by selecting **Tools → Scale Model…** from the OpenSim main menu bar. Like all tools, the operations performed by the Scale Tool apply to the current model.

4.2.1 Input

The modelName_Setup_Scale.xml file is the setup file for the Scale Tool. Note that all filenames given are examples. You may use different naming conventions, if desired. The setup file contains settings, which help describe the model, data, and parameters for scaling. These include:

1. modelName.osim: A generic, unscaled model.
2. modelName_Scale_MarkerSet.xml: A virtual marker set that corresponds to your experimental marker set.
3. subject_static.trc: The experimental marker data of your subject in a static pose and the time range when the static pose was collected. The static pose should include the subject wearing the full marker set. The marker trajectories are specified in global frame.

4. modelName_Scale_MeasurementSet.xml: The measurement set for the Scale Tool, which contains pairs of experimental markers, the distance between which are used to scale the generic musculoskeletal model.
   AND/OR
   modelName_Scale_ScaleSet.xml: Scale set for the Scale Tool. Alternately, you can use manual scale factors to scale the generic musculoskeletal model.
5. modelName_Scale_Tasks.xml: In addition to scaling the model, the Scale Tool moves the virtual markers on the model so that their positions match the experimental marker locations. To do this, the Scale Tool must position the model so that it best matches the position of the subject, which requires an inverse kinematics problem to be solved. This file contains the inverse kinematics tasks describing which virtual and experimental markers should be matched up during the inverse kinematics phase. The file also contains marker weights, which are relative and determine how "well" the virtual markers track experimental markers (i.e., a larger weight will mean less error between virtual and experimental marker positions).
6. subject_static.mot (optional): Experimental generalized coordinate values (joint angles) for a trial obtained from alternative motion capture devices or other specialized algorithms. You can specify coordinate weights in the Tasks file, if joint angles are know a priori. Coordinate weights are also relative and determine how "well" a joint angle will track the specified angle.

4.2.2 Output

1. subject.osim: OpenSim musculoskeletal model scaled to the dimensions of the subject.
2. subject_static_ik.mot: A motion file containing the joint angles for the static pose.

4.3 Best Practices and Troubleshooting

1. When collecting data, take pictures of your subjects in the static pose.
2. Have your subjects perform movements to calculate functional joint centers at the hip, knee, ankle, and/or shoulders and append the joint centers to your static trial data.
3. Measure subject specifics, like height, mass, body segment lengths, mass distribution (if DXA is available), and strength (if a Biodex is available).

4. Do not use all markers from motion-capture to position and scale the model. Markers that match anatomical landmarks and functional joint centers are the only markers that can be relied on for scaling. Avoid adjusting their model positions to match experimental positions, and use the marker position errors to assess the quality of your scaling results.
5. Some segments, like the pelvis are best scaled non-uniformly.
6. In general, maximum marker errors for bony landmarks should be <2 cm.
7. Use what you know about your subject's static pose when looking at the results of Scale. For example, in a typical static posture ankle angle is generally less than 5^{o} and hip flexion angle is less than 10^{o}.
8. Use the "preview static pose" option in the GUI to visualize the scaled model's anatomical marker positions relative to the corresponding experimental markers to see how well the model "fits" the data before adjusting all the markers to match the experimental data.
9. If the results of scale look incorrect, you can either change the location of virtual markers or alter marker weightings to calculate static pose.
10. Use coordinate tasks (Static Pose Weights) to set joint angles for troublesome joints (commonly the ankle joint and lumbar joint) that are very sensitive to how the markers are placed. For example if it is known that the foot is flat, an ankle angle can be provided and then the markers adjusted in order to match the known pose.
11. It is common to iterate through Scale and Inverse Kinematics to fine-tune segment dimensions and marker positions that yield low marker errors for the task of interest.
12. If using coordinates from a motion capture system make sure that the joint/coordinate definitions match otherwise you may cause more harm than good.
13. The model has a built in assumption that the global Y axis is up. If your data doesn't fit this, then consider transforming it using the Preview Motion Data option in the GUI.

5 Inverse Kinematics

5.1 How It Works

The Inverse Kinematics Tool steps through each time frame of experimental data and positions the model in a pose that “best matches” experimental marker and coordinate data for that time step. This “best match” is the pose that minimizes a sum of weighted squared errors of markers and/or coordinates.

In this chapter, we provide a conceptual review of the inputs and outputs of the Inverse Kinematics (IK) tool and a set of troubleshooting tips and best practices. The OpenSim User’s Guide provides detailed information and step-by-step instructions (Chapter 15). Getting accurate results from the IK tool is essential for using later tools like Static Optimization, Residual Reduction Algorithm, and Computed Muscle Control.

5.2 Inverse Kinematics Tool

To launch the IK Tool, select **Tools → Inverse Kinematics** from the OpenSim main menu bar.

5.2.1 Input

The primary inputs to IK are the following files:

1. subject01.osim: A subject-specific OpenSim model generated by scaling a generic model with the Scale Tool or by other means, along with an associated marker set containing adjusted virtual markers.
2. motionTrial.trc: Experimental marker trajectories for a trial obtained from a motion capture system, along with the time range of interest
3. marker_tasks.xml: A file containing marker weightings. As in the scale tool, marker weights are relative and determine how "well" the virtual markers track experimental markers (i.e., a larger weight will mean less error between virtual and experimental marker positions).
4. subject01_coords.mot (optional): Experimental generalized coordinate values (joint angles) for a trial obtained from alternative motion capture devices or other

specialized algorithms. You can optionally specify relative coordinate weights in the Tasks file, if joint angles are known a priori.

5.2.2 Output

1. subject01_ik.mot: A motion file containing the generalized coordinate trajectories (joint angles and/or translations) computed by IK.

5.3 Best Practices and Troubleshooting

1. When collecting experimental data, place three non-collinear markers per body segment that you want to track. You need at least three markers to track the 6 DOF motion (position and orientation) of a body segment.
2. Place markers on anatomical locations with minimum skin/muscle motion.
3. Weight "motion" segment markers, for example from a triad placed on the thigh segment, more heavily than anatomical markers affixed to landmarks like the greater trochanter and the acromion, which can be helpful for scaling, but are influenced by muscle and other soft tissue movements during motion.
4. Relative marker weightings are more important than their absolute values. Therefore, a weighting of 10 vs. 1 is 10 times more important whereas 20 vs. 10 is only twice as important. Markers are not necessarily tracked better because they both have higher weightings.
5. Total RMS and max marker errors are reported in the messages window. Use these values to guide changes in weightings, or if necessary to redo marker placement and possibly scaling. Maximum marker error should generally be less than 2-4cm and RMS under 2cm is achievable.
6. Compare your results to similar data reported in the literature. Your results from an unimpaired average adult should generally be within one standard deviation.

6 Inverse Dynamics

6.1 How it Works

The Inverse Dynamics Tool steps through each time frame of a motion and computes the net forces and/or torques at each joint in the model based upon/due to the experimental kinematics. The equations of motion relate the model accelerations to the forces and/or joint torques applied to the model.

In this chapter, we provide a conceptual review of the inputs and outputs of the Inverse Dynamics tool and a set of troubleshooting tips and best practices. The OpenSim User's Guide provides detailed information and step-by-step instructions (Chapter 16).

6.2 Inverse Dynamics Tool

To launch the Inverse Dynamics Tool select **Inverse Dynamics…** from the **Tools** menu. The **Inverse Dynamics Tool** dialog, like all other OpenSim tools, operates on the Current Model open and selected in OpenSim.

6.2.1 Input

1. subject01.osim: A subject-specific OpenSim model generated by scaling a generic model with the Scale Tool or by other means, along with an associated marker set containing adjusted virtual markers. The model must include inertial parameters. Note that forces like contact, ligaments, bushings, and even muscles will be applied to the model based on the kinematic state of the model and defaults for the muscle states, unless these forces are specifically excluded in the calculation.
2. subject01_ik.mot: Coordinate motion data (i.e., joint angles or translation for a motion) and time range for the motion of interest).
3. subject01_grf.xml: External load data (i.e., ground reaction forces, moments, and center of pressure location). Note that it is necessary to measure and apply or model all external forces acting on a subject during the motion to calculate accurate joint torques and forces. This file includes the name of the ground reaction force-data file (e.g. subject01_grf.mot) as well as the names of the bodies they are applied to. Options to specify the forces, point of application, and torques in a global or body

local frame (relative to the body to which the force is being applied) are also defined here.

6.2.2 Output

1. subject01_id.sto: Joint torques and forces, acting along the coordinate axes that produce the accelerations estimated (via double differentiation) from your measured experimental motion and modeled and external forces applied.

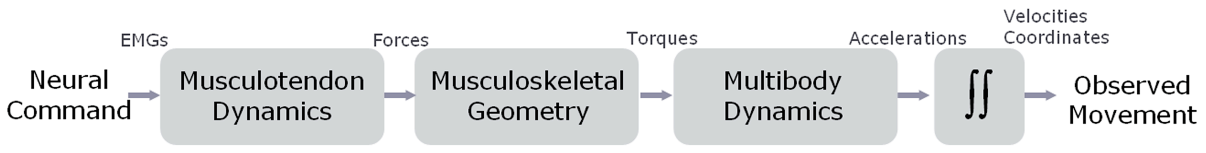
6.3 Best Practices and Troubleshooting

1. Filter your raw coordinate data, since noise is amplified by differentiation. Without filtering, the calculated forces and torques will be very noisy.
2. Compare your results to data reported in the literature. Your results should be within one s.d. of reported values.
3. Inspect results from Inverse Dynamics to check if ground reaction forces were applied correctly or not. For gait, applying ground reaction forces should help reduce the forces computed by Inverse Dynamics at the pelvis.

7 Forward Dynamics

7.1 How it Works

The Forward Dynamics Tool uses the model together with the initial states and controls to run a muscle-driven forward dynamics simulation. A forward dynamics simulation is the solution (integration) of the differential equations that define the dynamics of a musculoskeletal model.



7.1.1 Musculoskeletal Model Dynamics

In contrast to inverse dynamics where the motion of the model was known and we wanted to determine the forces and torques that generated the motion, in forward dynamics, a mathematical model describes how coordinates and their velocities change due to applied forces and torques (moments).

From Newton's second law, we can describe the accelerations (rate of change of velocities) of the coordinates in terms of the inertia and forces applied on the skeleton as a set of rigid-bodies:

$$\ddot{\boldsymbol{q}} = \left[\boldsymbol{M}(\boldsymbol{q})\right]^{-1}\left\{\boldsymbol{\tau} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{F}\right\} \qquad \text{Multibody dynamics}$$

where $\ddot{\boldsymbol{q}}$ is the coordinate accelerations due to joint torques, $\boldsymbol{\tau}$, Coriolis and centrifugal forces, $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$, as a function of coordinates, $\boldsymbol{q}$, and their velocities, $\dot{\boldsymbol{q}}$, gravity, $\boldsymbol{G}(\boldsymbol{q})$, and other forces applied to the model, $\boldsymbol{F}$, and $\left[\boldsymbol{M}(\boldsymbol{q})\right]^{-1}$ is the inverse of the mass matrix.

$$\boldsymbol{\tau}_m = \left[\boldsymbol{R}(\boldsymbol{q})\right]\boldsymbol{f}(\boldsymbol{a},\boldsymbol{l},\dot{\boldsymbol{l}}) \qquad \text{Moments due to muscle forces}$$

$$\dot{\boldsymbol{l}} = \Lambda(\boldsymbol{a},\boldsymbol{l},\boldsymbol{q},\dot{\boldsymbol{q}}) \qquad \text{Muscle contraction dynamics}$$

$$\dot{\boldsymbol{a}} = \mathrm{A}(\boldsymbol{a}, \boldsymbol{x})$$ Muscle activation dynamics

The net muscle moments, $\boldsymbol{\tau}_m$, in turn, are a result of the moment arms, $\boldsymbol{R}(\boldsymbol{q})$, multiplied by muscle forces, $\boldsymbol{f}$, which are a function of muscle activations, $\boldsymbol{a}$, and muscle fiber lengths, $\boldsymbol{l}$, and velocities, $\dot{\boldsymbol{l}}$. Muscle fiber velocities are governed by muscle contraction dynamics, Λ, which is dependent on the current muscle activations and fiber lengths as well as the coordinates and their velocities. Activation dynamics, A, describes how the activation rates, $\dot{\boldsymbol{a}}$, of the muscles respond to input neural excitations, $\boldsymbol{x}$, generally termed the model's controls. These form a set of differential equations that model *musculoskeletal dynamics*.

7.1.2 States of a Musculoskeletal Model

The *state* of a model is the collection of all model variables defined at a given instant in time that are governed by dynamics. The model dynamics describe how the model will advance from a given state to another through time. In a *musculoskeletal model* the states are the coordinates and their velocities and muscle activations and muscle fiber lengths. The dynamics of a model require the state to be known in order to calculate the rate of change of the model states (joint accelerations, activation rates, and fiber velocities) in response to forces and controls.

7.1.3 Controlling a Musculoskeletal Model

The forces (e.g., muscles) in a *musculoskeletal model* are governed by dynamics and have inputs that affect their behavior. In OpenSim, these inputs are called the *controls* of a model, which can be excitations for muscles or torque generators. Ultimately, controls determine the forces and/or torques applied to the model and therefore determine the resultant motion.

7.1.4 Numerical Integration of Dynamical Equations

A simulation is the integration of the musculoskeletal model's dynamical equations starting from a user-specified initial state. After applying the controls, the activation rates, muscle fiber velocities, and coordinate accelerations are computed. Then, new states at small time interval in the future are determined by numerical integration. A 5th-order Runge-Kutta-Feldberg integrator is used to solve (numerically integrate) the dynamical equations for the trajectories of the musculoskeletal model states over a definite interval in time. The Forward Dynamics Tool is an open-loop system that applies muscle/actuator controls with no feedback, or correction mechanism, therefore the states are not required to follow a desired trajectory.

7.2 Forward Dynamics Tool

To launch the Forward Dynamics Tool select **Forward Dynamics…** from the **Tools** menu. The **Forward Dynamics Tool** dialog like all other OpenSim tools operates on the *Current Model* open and selected in OpenSim.

7.2.1 Inputs

Three items are required by the Forward Dynamics Tool:

1. subject01.osim: A subject-specific OpenSim model generated by scaling a generic model with the Scale Tool or by other means, along with an associated marker set containing adjusted virtual markers. The model must include inertial parameters (segment masses, etc.).
2. subject01_controls.xml: XML file containing the time histories of the model controls (e.g., muscle excitations) to the muscles and/or joint torques. This file may be generated by the user, Static Optimization Tool, or Computed Muscle Control Tool. If no controls are provided they are assumed to be zero for any actuators in the model.
3. subject01_states.sto: Storage file containing the initial states of the musculoskeletal model that includes coordinates and their velocities and muscle activations and muscle fiber lengths. Alternately, the simulation can begin from the pose that mode is in without providing initial states. Muscle states are estimated by solving for muscle-fiber and tendon force equilibrium.

A few *additional* items may be provided to the Forward Dynamics Tool:

1. subject01_grf.xml: External load data (i.e., ground reaction forces, moments, and center of pressure location).
2. Settings for numerical integration
3. Additionally, analyses can be added so that they are executed during a forward simulation.

7.2.2 Outputs

The Forward Dynamics Tool generates storage files containing the time histories of the model's controls and states that result from integration of the model's dynamical equations in the output directory specified.

7.3 Best Practices and Troubleshooting

1. Forward dynamics simulations are sensitive to initial conditions and it is good practice to double check that they are appropriate for the desired simulation.

2. If the Forward Tool fails gracefully (i.e., without crashing OpenSim) or the output of the Forward Tool drifts too much (i.e., the model goes crazy), shorten the interval over which the Forward Tool runs (i.e., make initial_time and final_time closer to each other in the Forward Tool setup dialog box or setup file). Open-loop forward dynamics tends to drift over time due to the accumulation of numerical errors during integration.

8 Static Optimization

8.1 How it Works

As described in the previous chapter, the motion of the model is completely defined by the generalized positions, velocities, and accelerations. The Static Optimization Tool uses the known motion of the model to solve the equations of motion for the unknown generalized forces (e.g., joint torques) subject to one of the following muscle activation-to-force conditions:

$$\underbrace{\sum_{m=1}^{nm}\left(a_m F_m^0\right) r_{m,j} = \tau_j}_{\text{ideal force generators}} \quad \text{or} \quad \underbrace{\sum_{m=1}^{nm}\left[a_m f\left(F_m^0, l_m, v_m\right)\right] r_{m,j} = \tau_j}_{\text{constrained by force-length-velocity properties}}$$

while minimizing the objective function:

$$J = \sum_{m=1}^{nm}\left(a_m\right)^p$$

where *nm* 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; l_m is its length; v_m is its shortening velocity; $f\left(F_m^0, l_m, v_m\right)$ is its force-length-velocity surface; $r_{m,j}$ is its moment arm about the j th joint axis; τ_j is the generalized force acting about the j th joint axis; and p is a user defined constant.

8.2 Static Optimization Tool

To launch the Static Optimization Tool, select **Static Optimization...** from the **Tools** menu. The *Static Optimization Tool* dialog window, like all other OpenSim tools, operates on the current model open and selected in OpenSim

8.2.1 Input

1. subject01.osim: A subject-specific OpenSim model generated by scaling a generic model with the Scale Tool or by other means, along with an associated marker set containing adjusted virtual markers. The model must include inertial parameters (segment masses, etc.).
2. subject01_ik.mot: Kinematic data (i.e., joint angles) from IK or states (i.e., joint angles AND velocities) from RRA and the time range of interest.
3. x: The exponent for the activation-based cost function, $\sum_{actuators} a^x$ to be minimized (i.e., the criteria used to solve muscle force distribution problem).
4. subject01_grf.xml: External load data (i.e., ground reaction forces, moments, and center of pressure location). Note that you must measure or model all external forces acting on a subject during the motion to calculate accurate muscle forces.

8.2.2 Output

1. subject01_actuator_forces.sto: A set of muscle and other actuator forces that produce the necessary generalized force (joint torque) for your measured motion.

8.3 Best Practices and Troubleshooting

1. You can use IK or RRA results as input kinematics. If using IK results, you usually need to filter them, either externally or using the OpenSim analyze/static optimization field. If using RRA results, you usually do not have to filter.
2. Add residual actuators to the first free joint in the model (typically the ground-pelvis joint).
3. If the actuators/muscles are weak, the optimization will take a long time to converge or never converge at all. This takes a long time. If troubleshooting a weak model and each time, optimization is slow, try reducing the parameter that defines the max number of iterations.
4. Increase the maximum control value of a residual or reserve actuator, while lowering its maximum force. This allows the optimizer to generate a large force (if necessary) to match accelerations but large control values are penalized more heavily. In static optimization, ideal actuator excitations are treated as activations in the cost function.

9 Residual Reduction Algorithm

9.1 How it Works

The purpose of Residual Reduction is to minimize the effects of errors in modeling and marker kinematics that lead to significant nonphysical forces called residuals. Specifically, residual reduction slightly adjusts the mass properties of a subject-specific model and the joint kinematics from inverse kinematics to improve dynamic consistency with respect to the ground reaction force data.

In this chapter, we provide a conceptual review of the inputs and outputs of the Residual Reduction Algorithm (RRA) along with a set of troubleshooting tips and best practices for completing the tool. RRA is covered in detail in Chapter 19 of the OpenSim User's Guide.

9.2 Residual Reduction Algorithm Tool

The residual reduction algorithm tool is accessed by selecting **Tools** → **Residual Reduction Algorithm…** from the OpenSim main menu bar. Like all tools, the operations performed by the computed muscle control tool apply to the current model.

9.2.1 Input

1. subject01.osim: A subject-specific OpenSim model generated by scaling a generic model with the Scale Tool or by other means, along with an associated marker set containing adjusted virtual markers. The model must include inertial parameters.
2. subject01_ik.mot: Kinematic data (i.e., joint angles) from IK and time range.
3. subject01_RRA_tasks.xml: A tracking tasks file specifying which coordinates to track and the corresponding tracking weight (weights are relative and determine how "well" a joint angle will track the specified joint angle from IK).
4. subject01_RRA_actuators.xml: The Actuator Set specifies the residual and reserve actuators to be applied and their parameters, like maximum/minimum force and body or joint, or location, depending on the actuator type. Each degree of freedom in the model should have an ideal torque or force (reserve) actuator. This includes the 6 DOFs of the

model's base segment, which are called the residual actuators. In most circumstances, these Ideal joint actuators used to replace the muscles in the model (by checking "Replace model actuators" in the Actuators tab.

5. subject01_RRA_ControlConstraints.xml: The actuator constraints file specifying the maximum and minimum "excitation" (i.e., control signal) for each actuator. Note that the maximum/minimum force or torque generated by an ideal actuator is the product of the max/min force and max/min excitation.
6. subject01_grf.xml: External load data (i.e., ground reaction forces, moments, and center of pressure location). See Inverse Dynamics (Handout Chapter 6) for more details.
7. Integrator settings (i.e., max number of steps, max step size, min step size, and error tolerance)

9.2.2 Output

1. subject01_RRA_states.sto: Adjusted kinematics (i.e., joint angles) and corresponding model states of the simulated motion (i.e., joint angles AND velocities).
2. subject01_adjusted.osim (optional): A model with adjusted mass properties.
3. subject01_RRA_forces.sto: Actuator forces and torques (i.e., joint torques corresponding to adjusted kinematics).
4. subjct01_RRA_controls.xml: Actuator excitations (i.e., control signals needed to generate actuator forces and torques)

9.3 Best Practices and Troubleshooting

1. RMS difference in joint angle during the movement should be less than 2-5° (or less than 2 cm for translations).
2. Peak Residual Forces should be less than 10-20 N.
3. Compare the residual moments from RRA to the moments from Inverse Dynamics. You should see a 30-50% reduction in peak residual moments.
4. Compare the joint torques/forces to established literature (if available). Try to find data with multiple subjects. Your results should be within one standard deviation of the literature.
5. Optimal forces for residuals should be low to prevent the optimizer from "wanting" to use residual actuators (an actuator with large optimal force and low excitation is "cheap" in the optimizer cost).

6. To help minimize residuals, make an initial pass with default inputs, then check residuals and coordinate errors. To reduce residuals further, decrease tracking weights on coordinates with low error. You can also try decreasing the maximum excitation on residuals or the actuator optimal force.
7. If RRA is failing, try increasing the max excitation for residuals by 10x until the simulation runs. Then try working your way back down while also "relaxing" tracking weights on coordinates.
8. If residuals are very large (typically, this is greater than 2-3x BW, depending on the motion), there is probably something wrong with either (i) the scaled model, (ii) the IK solution, or (iii) the applied GRFs. To double check that forces are being applied properly, visualize GRFs with IK data (you can use the “Preview motion data” function in the GUI).

10 Computed Muscle Control

10.1 How it Works

The purpose of Computed Muscle Control (CMC) is to compute a set of muscle excitations (or more generally actuator controls) that will drive a dynamic musculoskeletal model to track a set of desired kinematics in the presence of applied external forces (if applicable).

In this chapter, we provide a conceptual review of the inputs and outputs of the Computed Muscle Control (CMC) tool, along with a set of troubleshooting tips and best practices. CMC is a tool for estimating the muscle excitations that drive a subject's motion and is covered in detail in Chapter 20 of the OpenSim User's Guide.

10.2 Computed Muscle Control Tool

The computed muscle control tool is accessed by selecting **Tools → Computed Muscle Control...** from the OpenSim main menu bar. Like all tools, the operations performed by the computed muscle control tool apply to the current model.

10.2.1 Input

1. subject01.osim: A subject-specific OpenSim model generated by scaling a generic model with the Scale Tool or by other means, along with an associated marker set containing adjusted virtual markers. The model must include inertial parameters.
2. subject01_RRA_states.sto: Kinematic data (i.e., joint angles) from RRA and the time range of interest.
3. subject01_CMC_tasks.xml: A tracking tasks file specifying which coordinates to track and the corresponding tracking weight (weights are relative and determine how "well" a joint angle will track the specified joint angle from RRA).
4. subject01_CMC_actuators.xml: This includes reserve and residual actuators, as in RRA.
5. subject01_CMC_ControlConstraints.xml: The control constraints file specifying the maximum and minimum "excitation" (i.e., control signal) for each actuator. Control constraints can also be used to enforce when certain actuators are "on" or "off" and the range in which they can operate.
6. subject01_grf.xml: External load data (i.e., ground reaction forces, moments, and center of pressure location). See Inverse Dynamics (Handout Chapter 6) for more details.

7. Integrator settings (i.e., max number of steps, max step size, min step size, and error tolerance)

10.2.2 Output

1. subject01_CMC_states.sto: Model and muscle states of the simulated motion (i.e., joint angles AND velocities, muscle fiber lengths AND activations).
2. subjct01_CMC_controls.xml: Actuator (e.g., muscle) excitations (i.e., control signals needed to generate muscle forces and reserve/residual forces and torques).
3. subject01_CMC_forces.sto: Muscle forces and reserve/residual forces and torques .

10.3 Best Practices and Troubleshooting

1. Compare the simulated activations to experimental EMG data (either recorded from your subject or from the literature). Activations should exhibit similar timing and magnitude to EMG data.
2. Peak reserve actuators torques should typically be less than 10% of the peak joint torque.
3. If performing Perturbation or Induced Acceleration Analysis, you should verify that reserves and residuals contribute less than 5% to the net acceleration of interest.
4. Compare your results (muscle activations or forces) to other simulations, if available.
5. To help minimize reserve torques, make an initial pass with default inputs, and then check reserves, residuals, and joint angle errors. To reduce reserves further, decrease tracking weights on coordinates with low error.
6. Optimal forces for reserves should be low to prevent optimizer from "wanting" to use reserve actuators (an actuator with large optimal force and low excitation is "cheap" in the optimizer cost). Increase the maximum control value of residuals so they can generate sufficient force, but are penalized for doing so.
7. If CMC is failing, try increasing the max excitation for reserves and residuals by 10x until the simulation runs. Then try working your way back down while also "relaxing" tracking weights on coordinates.

11 Analyzing a Simulation: Estimating Joint Loads

11.1 Overview

JointReaction is an OpenSim Analysis for calculating the joint forces and moments transferred between consecutive bodies in a model. These forces and moments correspond to the internal loads carried by the joint structure. These loads represent the contributions of all un-modeled joint structures that would produce the desired joint kinematics, such as cartilage contact and any omitted ligaments. The reaction load acts at the joint center (mobilizer frame) of both the parent and child bodies. The loads can be reported and expressed in either the child, parent, or ground frames. The default behavior is to express the force on the child in the ground frame. For more detailed description of the method implementation see *Tibiofemoral Contact Force during Crouch Gait* (Steele et al, in review).

In this chapter, we provide a conceptual review of the inputs and outputs for a JointReaction Analysis along with a set of troubleshooting tips and best practices. Running an Analysis like JointReaction is covered in detail in Chapter 21 of the OpenSim User's Guide.

11.2 Performing a JointReaction Analysis

The Analyze Tool is accessed by selecting **Tools → Analyze...** from the OpenSim main menu bar. Like all tools, the operations performed by the Analyze Tool apply to the current model.

11.2.1 Input

Inputs Specific to JointReaction:

1. joint_names: List of the names of the joints of interest. JointReaction reports loads for only listed joints that exist in the model. Joint names may be repeated any number of times to allow reporting on different bodies or with respect to different reference frames. Using the keyword 'all' reports the loads for all joints in the model. Default is 'all'.

2. apply_on_bodies: List of the body (parent or child) on which the corresponding reaction occurred. If the array has only one entry, that selection is applied to all joints specified in joint_names. The default is 'child'.
3. express_in_frame: List of the frames (ground, parent, or child) in which the corresponding reaction is expressed. If the array has only one entry, that selection is applied to all joints specified in joint_names.
4. forces_file: The name of a file containing forces storage. If a file name is provided, the applied forces for all actuators will be constructed from the forces_file instead of from the states. This option should be used to calculate joint loads from static optimization results.

11.2.2 Output

JointReaction prints results to one storage file with the suffix "_ReactionLoads.sto". This file contains rows of time-stamped data containing the 3 force and 3 moment vector components of the reaction load at each joint specified. Each data column label includes all information about how the load is applied and expressed. Specifically, the form is

< joint name >_on_<body>_in_<frame>_<component>.

For example, the column containing the z-component of hip force occurring on the femur and expressed in the femur frame would be labeled "hip_on_femur_in_femur_FX" while the y-component of the knee moment occurring on the tibia and expressed in the ground frame would have the label "knee_on_tibia_in_ground_MY".

11.3 Best Practices and Troubleshooting

Input error: If JointReaction doesn't recognize a specified joint, body, or frame name, it will perform its default action. The overall default action is to report the loads on all joints as applied to the joint's child body and expressed in the ground reference frame.

Consistency of modeling inputs to the Analyze tool: The validity of the joint loads depends on modeling assumptions and correct modeling practices. JointReaction is very sensitive to the consistency of all inputs that define a dynamic trial, including the following inputs to the Analyze Tool: <model_file>, <replace_force_set>, <force_set_files>, <states_file>, <coordinates_file>, <speeds_file>, <lowpass_cutoff_frequency_for_coordinates>, <external_loads_file>, <external_loads_model_kinematics_file>, <lowpass_cuttoff_frequency_for_load_kinematics>

If any of these files are not associated with the same dynamic trial, the system of accelerations and forces will not be consistent. Therefore, JointReaction will calculate incorrect joint loads.

Special Types of Forces: Certain types of forces and actuators, called SpringGeneralizedForce, CoordinateLimitForce, and CoordinateActuator, are associated with specific degrees of freedom and treated as part of the joint structure. This means that any contribution from these components will be treated as part of the resultant loads reported by JointReaction instead of body forces. As an example, consider a reserve actuator on the hip joint of a gait model. If this reserve actuator is defined as a CoordinateActuator, its contribution is treated as a residual force provided by the joint and therefore will be added to the resultant load at the hip. However, if the hip reserve is defined as a TorqueActuator, its torque will be treated as a motor external to the joint and attached between the pelvis and femur. Therefore, its torque will not be added to the reported resultant load at the hip.

12 Elements of a Model

12.1 What is a musculoskeletal model in OpenSim?

In OpenSim, the skeletal part of a model is represented by rigid bodies interconnected by joints. Joints define how a body (e.g., bone segment) can move with respect to its parent body. In OpenSim, all bodies have a parent and are connected to its parent via a joint, except for ground. Constraints can also be applied to limit the motion of bodies.

Muscles are modeled as specialized force elements that act at muscle points (e.g., insertion and origin points) connected to rigid bodies. The force of a muscle is typically dependent on the path through muscle points comprised of muscle fiber and tendon lengths, the rate of change of the fiber lengths, and the level of muscle activation. OpenSim also has a variety of other forces, which represent externally applied forces (e.g. ground reaction forces), passive spring-dampers (e.g., ligaments), and controlled linear and torsional actuators.

12.2 Organization of the OpenSim model file

In formulating the equations-of-motion (i.e., the system dynamics), OpenSim employs Simbody which is an open-source multibody dynamics solver. In Simbody and OpenSim, the body is the primary building block of the model. Each body in turn owns a joint and that joint defines the coordinates and kinematic transforms that govern the motion of that body with respect to its parent body. Within the model all bodies are contained in a BodySet. The ConstraintSet contains all the kinematic constraints that act on bodies (and/or their coordinates) in the model. User forces acting on the model are all included in a ForceSet.

```
<Model name="Bouncing Ball">
    <!—Default values for properties that are not specified -->
    <defaults> ...
    <credits> John Doe </credits>
    <publications> Fantastic Journal of  ... </publications>
    <length_units> m </length_units>
    <force_units> N </force_units>
    <!--Acceleration due to gravity.-->
    <gravity> 0.00000000      -9.80650000       0.00000000 </gravity>
    <!--Bodies in the model.-->
    <BodySet name=""> ...
    <!--Constraints in the model.-->
    <ConstraintSet name=""> ...
    <!--Forces in the model.-->
    <ForceSet name=""> ...
    <!--Markers in the model.-->
    <MarkerSet name=""> ...
    <!—ContactGeometry associated which contact forces that are in the model.-->
    <ContactGeometrySet name=""> ...
</Model>
```

Figure 7-1: Top level organization of an OpenSim model file

12.3 Specifying a Body and its Joint

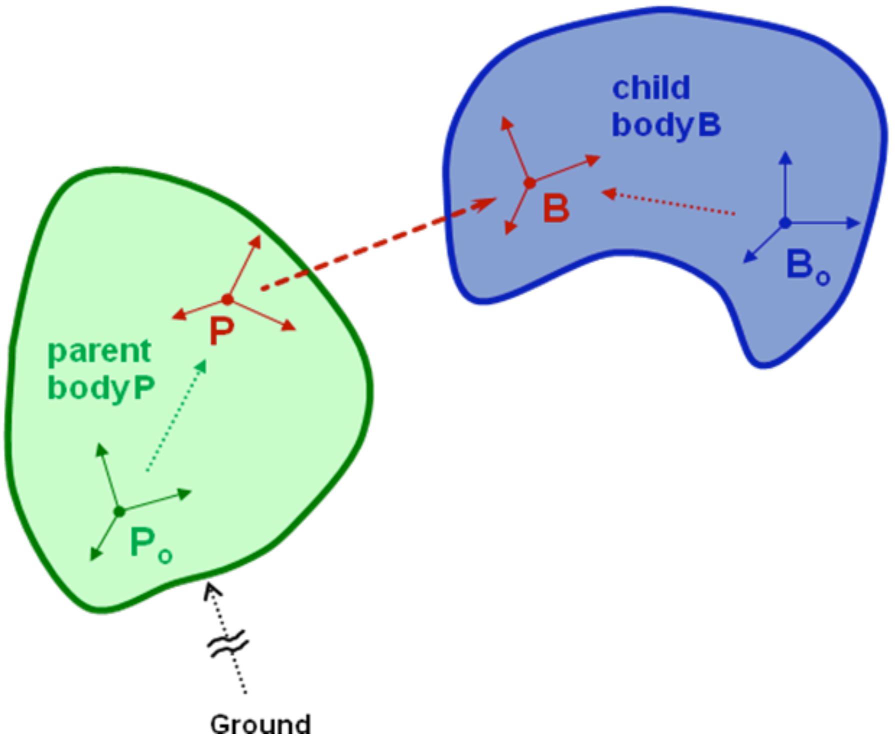


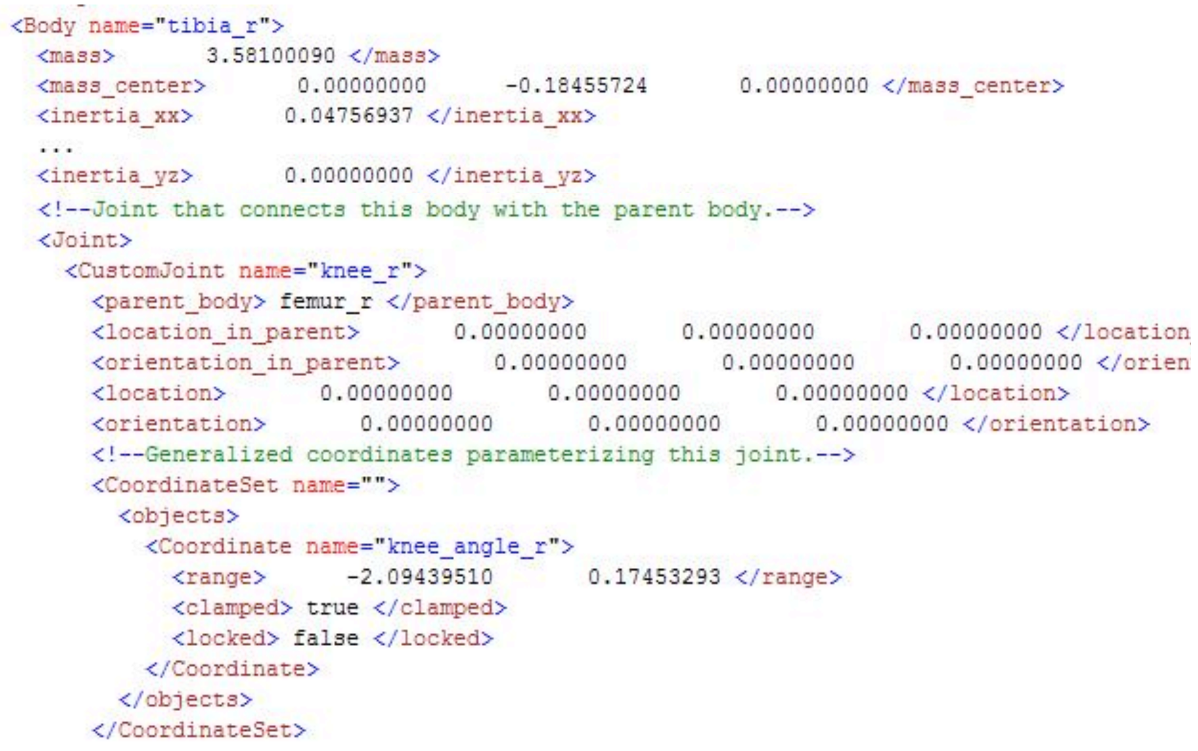
Figure 8: A joint (in red) defines the kinematic relationship between two frames (B and P) each affixed to a rigid-body (the parent, P_o, and the body being added, B_o) parameterized by joint coordinates

A body is a moving reference frame (B_o) in which its center-of-mass and inertia are defined, and the location of a joint frame (B) fixed to the body can be specified (Fig. 10-2). Similarly, the joint frame (P) in the parent body frame (P_o) can also be specified. Flexibility in specifying the joint is achieved by permitting joint frames that are not coincident with the body frame.

12.3.1 Available Joint Types

1. WeldJoint: introduces no coordinates (degrees of freedom) and fuses bodies together
2. PinJoint: one coordinate about the common Z-axis of parent and child joint frames

3. SliderJoint: one coordinate along common X-axis of parent and child joint frames
4. BallJoint: three rotational coordinates that are about X, Y, Z of B in P
5. EllipsoidJoint: three rotational coordinates that are about X, Y, Z of B in P with coupled translations such that B traces and ellipsoid centered at P
6. FreeJoint: six coordinates with 3 rotational (like the ball) and 3 translations of B in P
7. CustomJoint: user specified 1-6 coordinates and user defined spatial transform to locate B with respect to P



```
<Body name="tibia_r">
  <mass>       3.58100090 </mass>
  <mass_center>       0.00000000      -0.18455724       0.00000000 </mass_center>
  <inertia_xx>       0.04756937 </inertia_xx>
  ...
  <inertia_yz>       0.00000000 </inertia_yz>
  <!--Joint that connects this body with the parent body.-->
  <Joint>
    <CustomJoint name="knee_r">
      <parent_body> femur_r </parent_body>
      <location_in_parent>       0.00000000       0.00000000       0.00000000 </location
      <orientation_in_parent>       0.00000000       0.00000000       0.00000000 </orien
      <location>       0.00000000       0.00000000       0.00000000 </location>
      <orientation>       0.00000000       0.00000000       0.00000000 </orientation>
      <!--Generalized coordinates parameterizing this joint.-->
      <CoordinateSet name="">
        <objects>
          <Coordinate name="knee_angle_r">
            <range>      -2.09439510       0.17453293 </range>
            <clamped> true </clamped>
            <locked> false </locked>
          </Coordinate>
        </objects>
      </CoordinateSet>
```

Figure 10-3: Sample Body and Joint Definitions in OpenSim. The right knee joint is governed by one coordinate, the knee_angle_r.

12.3.2 The CustomJoint Transform

Most joints in an OpenSim model are custom joints since this is the most generic joint representation, which can be used to model both conventional (pins, slider, universal, etc...) as well as more complex biomechanical joints. The user must define the transform (rotation and translation) of the child in the parent (B and P, Fig. 10-2) as a function of the generalized coordinates listed in the Joint's CoordinateSet (Fig. 10-3). Consider the spatial transform ${}^{P}\mathbf{X}^{B}$:

$$
{}^{P}\mathbf{X}(q)^{B} = \begin{bmatrix} & x_4 \\ {}^{P}\mathbf{R}^{B}(x_1, x_2, x_3) & x_5 \\ & x_6 \end{bmatrix},
$$

where

$$
x(q) = \begin{Bmatrix} f_1(q_1, q_2, \ldots, q_m) \\ f_2(q_1, q_2, \ldots, q_m) \\ \vdots \\ f_6(q_1, q_2, \ldots, q_m) \end{Bmatrix},
$$

q are the joint coordinates, and x are the spatial coordinates for the rotations (x_1, x_2, x_3) and translations (x_4, x_5, x_6) along user-defined axes that specify a spatial transform (**X**) according to functions f_i. The behavior of a CustomJoint is specified by its SpatialTransform. A SpatialTransform is comprised of 6 TransformAxes (3 rotations and 3 translations) that define the spatial position of B in P as a function of coordinates. Each transform axis enables a function of joint coordinates to operate about or along its axis. The function of q is used to determine the displacement for that axis. The order of the spatial transform is fixed with rotations first followed by translations. Subsequently, coupled motion (i.e., describing motion of two degrees of freedom as a function of one coordinate) is easily handled. The example below (from the gait2354.osim model) describes coupled motion of the knee, with both tibial translation and knee flexion described as a function of knee angle, by continuing the CustomJoint definition from Fig. 10-3.

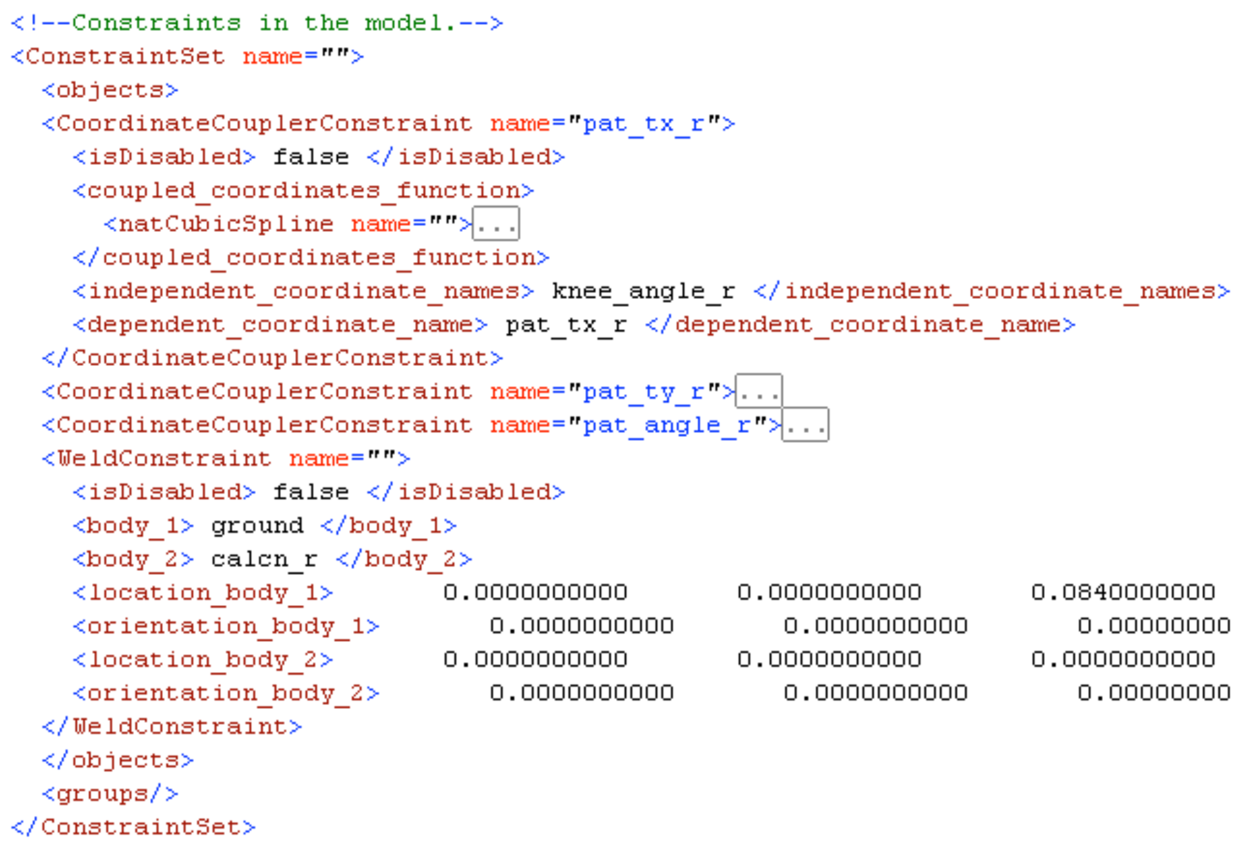
```
<SpatialTransform name="">
   <!--3 Axes for rotations are listed first.-->
   <TransformAxis name="rotation1">
      <function>
         <LinearFunction name="">
            <coefficients> 1.00000000       0.0000000 </coefficients>
         </LinearFunction>
      </function>
      <coordinates> knee_angle_r </coordinates>
      <axis>       0.00000000       0.00000000       1.00000000 </axis>
   <TransformAxis>
   <TransformAxis name="rotation2">
      <function>
         <Constant name="">
            <value>      0.00000000 </value>
         </Constant>
      </function>
      <coordinates> </coordinates>
      <axis>       0.00000000       1.00000000       0.00000000 </axis>
   </TransformAxis>
   <TransformAxis name="rotation3"> ...
   <!--3 Axes for translations are listed next.-->
   <TransformAxis name="translation1">
```

```
            <function>
               <NaturalCubicSpline name=""> ...
            </function>
            <coordinates> knee_angle_r </coordinates>
            <axis>      1.00000000      0.00000000      0.00000000 </axis>
         </TransformAxis>
         <TransformAxis name="translation2">
               <function>
               <NaturalCubicSpline name=""> ...
               </function>
            <coordinates> knee_angle_r </coordinates>
            <axis>      0.00000000      1.00000000      0.00000000 </axis>
         </TransformAxis>
         <TransformAxis name="translation3"> ...
      </SpatialTransform>
   </CustomJoint>
</Joint>
```

Figure 9: Spatial transform of a custom joint that implements a translating knee joint

12.3.3 Kinematic Constraints in OpenSim

OpenSim currently supports three types of built-in constraints: PointConstraint WeldConstraint and CoordinateCouplerConstraint. A point constraint fixes a point defined with respect to two bodies (i.e., no relative translations). A weld constraint fixes the relative location and orientation of two bodies (i.e., no translations or rotations). A coordinate coupler relates the generalized coordinate of a given joint (the dependent coordinate) to any other coordinates in the model (independent coordinates). The user must supply a function that returns a dependent value based on independent values. The following example implements coordinate coupler constraint for the motion of the patella as a function of the knee ankle and also welds the foot to ground.



```
<!--Constraints in the model.-->
<ConstraintSet name="">
  <objects>
  <CoordinateCouplerConstraint name="pat_tx_r">
    <isDisabled> false </isDisabled>
    <coupled_coordinates_function>
      <natCubicSpline name="">...
    </coupled_coordinates_function>
    <independent_coordinate_names> knee_angle_r </independent_coordinate_names>
    <dependent_coordinate_name> pat_tx_r </dependent_coordinate_name>
  </CoordinateCouplerConstraint>
  <CoordinateCouplerConstraint name="pat_ty_r">...
  <CoordinateCouplerConstraint name="pat_angle_r">...
  <WeldConstraint name="">
    <isDisabled> false </isDisabled>
    <body_1> ground </body_1>
    <body_2> calcn_r </body_2>
    <location_body_1>      0.0000000000       0.0000000000       0.0840000000
    <orientation_body_1>      0.0000000000       0.0000000000       0.00000000
    <location_body_2>      0.0000000000       0.0000000000       0.0000000000
    <orientation_body_2>      0.0000000000       0.0000000000       0.00000000
  </WeldConstraint>
  </objects>
  <groups/>
</ConstraintSet>
```

Figure 10: Example of constraints in OpenSim

12.4 Forces in OpenSim

Forces in OpenSim are contained in a model's ForceSet. Forces come in two varieties: *passive* forces like springs, dampers, and contact and *active* forces like motors and muscles. Active forces that require input (controls) supplied by the user or by a controller are called Actuators and are a subset of the ForceSet.

12.4.1Available Forces

OpenSim has several built-in forces that include: PrescribedForce, SpringGeneralizedForce, BushingForce, as well as HuntCrossleyForce and ElasticFoundationForce to model forces due to contact (Note: contact forces also require defining contact geometry). Below is an example of a bushing force used to model passive structures surrounding a single lumbar joint that connects a torso body to a pelvis body.

```
<!— Generate a force proportional to the separation of two frames in terms of both
relative rotational and translational displacement (stiffness) and velocity and
velocity (damping) -->
<BushingForce name="BackJointBushing">
   <body_1> pelvis </body_1>
   <body_2> torso </body_2>
   <location_body_1> -0.1007 0.0815 0.0000 </location_body_1>
   <orientation_body_1> 0.0000  0.0000  0.0000 </orientation_body_1>
   <location_body_2> 0.0000  0.0000   0.0000 </location_body_2>
   <orientation_body_2>  0.0000  0.0000  0.0000 </orientation_body_2>
   <rotational_stiffness> 10.000  10.000 10.000 </rotational_stiffness>
   <translational_stiffness> 0.0000 0.000 0.000 </translational_stiffness>
   <rotational_damping>  0.0000  0.0000  0.0000 </rotational_damping>
   <translational_damping> 0.0000 0.000 0.000 </translational_damping>
</BushingForce>
```

Figure 10-6: Example of a passive BushingForce

12.4.2 Common Actuators

OpenSim also includes "ideal" actuators which apply pure forces or torques that are directly proportional to the input control (i.e., excitation) via its optimal force (i.e., a gain). Forces and torques are applied between bodies, while generalized forces are applied along the axis of a generalized coordinate (i.e., a joint axis).

```
<!—Apply an equal and opposite force at points on two bodies along the line
that connects the two points -->
<PointActuator name="FY_residual">
   <optimal_force> 8.0 </optimal_force>
   <body> pelvis </body>
   <point> -0.0724376 0.00000000 0.00000000 </point>
   <direction> 0.0 1.0 0.0 </direction>
</PointActuator>

<!—Apply an equal and opposite torque on two bodies about the axis defined
in the the first body -->
<TorqueActuator name="MZ_residual">
   <optimal_force> 1000.0 </optimal_force>
   <bodyA> ground </bodyA>
   <axis> 0.000 0.000 -1.000 </axis>
   <bodyB> pelvis </bodyB>
</TorqueActuator >

<!—Apply a generalized force along (force) or about (torque) the axis of a
generalized coordinate. Positive force increases the coordinate -->
<CoordinateActuator name="knee_reserve">
   <optimal_force> 300.0 </optimal_force>
   <coordinate> knee_angle_r  </coordinate>
</CoordinateActuator>
```

Figure 11: Sample of linear and torque actuators in a model's ForceSet

12.4.3 The Muscle Actuator

There are several muscle models in OpenSim. All muscles include a set of muscle points where the muscle is connected to bones (bodies) and provide utilities for calculating muscle-actuator lengths and velocities. Internally muscle models may differ in the number and type of parameters. Muscles typically include muscle activation and contraction dynamics and their own states (for example activation and muscle fiber length). The control values are typically bounded excitations (ranging from 0 to 1) which lead to a change in activation and then force. Below is an example of a muscle model, as described by Thelen (2003), from an OpenSim model.

```
<Thelen2003Muscle name="soleus_r">
   <GeometryPath name="">
      <!-- points on bodies that define the path of the muscle -->
      <PathPointSet name="">
         <objects>
            <PathPoint name="soleus_r-P1">
               <location> -0.00240000 -0.15330000 0.00710000 </location>
               <body> tibia_r </body>
            </PathPoint>
            <PathPoint name="soleus_r-P2">
               <location> 0.00000000  0.03100000 -0.00530000 </location>
               <body> calcn_r </body>
            </PathPoint>
         </objects>
         <groups/>
      </PathPointSet>
      <PathWrapSet name=""> ...
   </GeometryPath>
   <!--maximum isometric force of the muscle fibers-->
   <max_isometric_force> 4000.00000000 </max_isometric_force>
   <!--optimal length of the muscle fibers-->
   <optimal_fiber_length> 0.08000000 </optimal_fiber_length>
   <!--resting length of the tendon-->
   <tendon_slack_length> 0.22000000 </tendon_slack_length>
   <!--angle between tendon and fibers at optimal fiber length-->
   <pennation_angle> 0.43633231 </pennation_angle>
   <!--time constant for ramping up of muscle activation-->
   <activation_time_constant> 0.01000000 </activation_time_constant>
   <!--time constant for ramping down of muscle activation-->
   <deactivation_time_constant> 0.04000000 </deactivation_time_constant>
   <!--maximum contraction velocity at full activation (fiber length/s)-->
   <Vmax> 10.00000000 </Vmax>
   <!--maximum contraction velocity at low activation (fiber lengths/s)-->
   <Vmax0> 5.00000000 </Vmax0>
   <!--tendon strain due to maximum isometric muscle force-->
   <FmaxTendonStrain> 0.03300000 </FmaxTendonStrain>
   <!--passive muscle strain due to maximum isometric muscle force-->
   <FmaxMuscleStrain> 0.60000000 </FmaxMuscleStrain>
   <!--shape factor for Gaussian active muscle force-length relationship-->
   <KshapeActive>  0.50000000 </KshapeActive>
   <!--exponential shape factor for passive force-length relationship-->
   <KshapePassive>  4.00000000 </KshapePassive>
   <!--passive damping in the force-velocity relationship-->
   <damping> 0.05000000 </damping>
   <!--force-velocity shape factor-->
   <Af> 0.30000000 </Af>
   <!--maximum normalized lengthening force-->
```

```
    <Flen>  1.80000000 </Flen>
</Thelen2003Muscle>
```

Figure 10-8: Sample muscle actuator from a model's ForceSet

13 Extending OpenSim's Capabilities

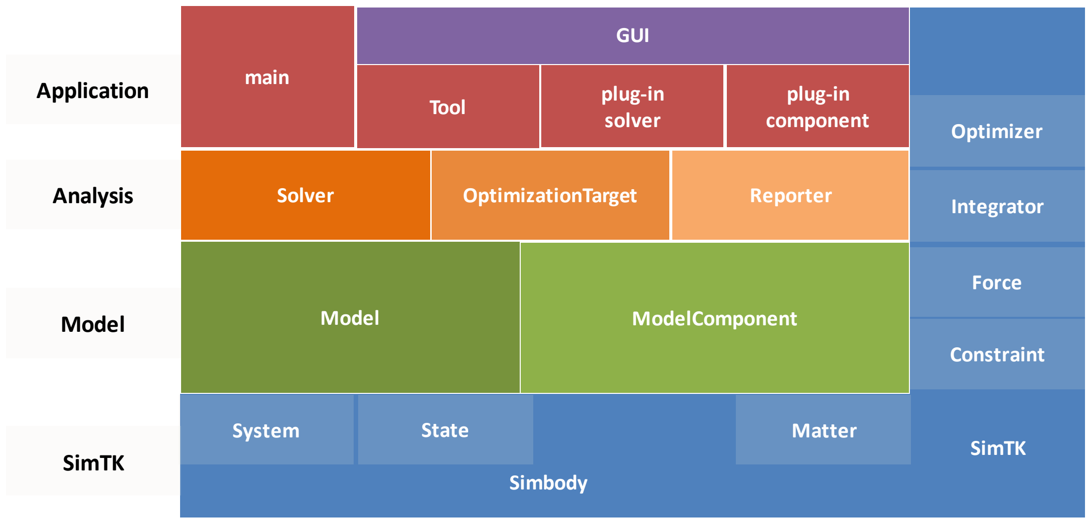
13.1 Overview

OpenSim provides several mechanisms for extending its existing capabilities either by adding new model elements, computing new quantities, or computing existing quantities in a new way. For example, you may want to model the drag acting on bodies moving through a fluid, which OpenSim does not provide. Another example is being able to extract the linear and angular momentum of the model during a simulation. In order to extend to OpenSim, it is important to know what functionality exists and to have a sense of where to add new functionality.

13.2 Organization of OpenSim

OpenSim is built on the computational and simulation core provided by SimTK. This includes low-level, efficient math and matrix algebra libraries such as LAPACK as well as the infrastructure for defining a dynamical system and its state. One can think of the system as the set of differential equations and the state comprised of its variables.

Empowering the computational layer is Simbody™, an efficient multibody dynamics solver, which provides an extensible multibody system and state. The OpenSim modeling layer maps biomechanical structures (bones, muscles, tendons, etc.) into bodies and forces so that the dynamics of the system can be computed by Simbody.



13-1: The three interface layers of OpenSim built on SimTK

OpenSim is essentially a set of modeling libraries for building complex actuators (e.g. muscles) and other forces (e.g. contact) and enabling the motion (kinematics) of highly articulated bodies (bones). Actuators can then be controlled by model controllers (e.g. Computed Muscle Control) to estimate the neural control and muscle forces required to reproduce human movement. An analysis layer is equipped with solvers and optimization resources for performing calculations with the model and to report results. At the highest level these blocks are assembled into specialized applications (ik.exe, forward.exe, analyze.exe) to simulate and analyze model movement and internal dynamics. The OpenSim application is a Java based program that calls Tools, Models, and underlying computations in SimTK to provide an interactive graphical user-interface (GUI).

13.3 OpenSim Model and ModelComponents

The job of an OpenSim::Model is to organize (hierarchically) the pieces (components) of a musculoskeletal system and to create a representative computational (mathematical) system that can be solved accurately and efficiently using Simbody and the flat SimTK::System framework.

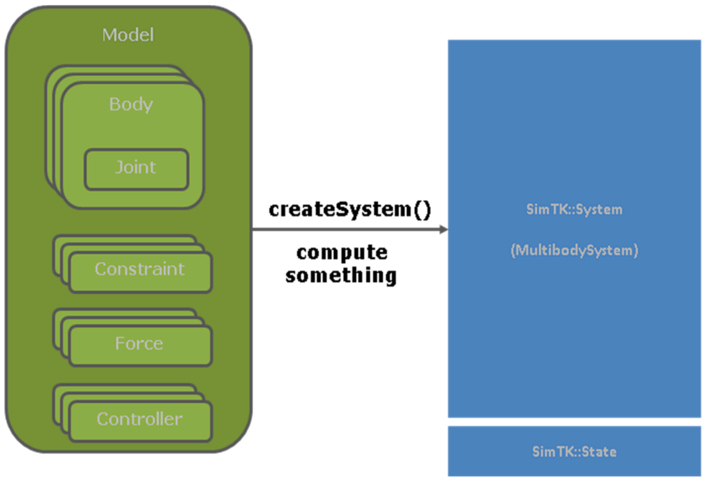


Figure 13-2: Organizational Context vs. Computation

By separating the contextual organization of a model from its computational representation, OpenSim can exploit the conceptual benefits of hierarchically organized models and software without sacrificing computational efficiency. One can then think of the system as the set of system equations while the state is a coherent set of system variable values that satisfies the system equations. Model components know about the parts they add to the multibody system (for example, another rigid body, a force, or a constraint) and are free to mix and match. For example, a Coordinate component knows how to access its underlying degree-of-freedom value, velocity and even its acceleration, given the system has been "solved" for accelerations. A Coordinate also adds different constraints to the underlying system, in the case that Coordinate is locked or if its motion is prescribed. It provides context to organize locking constraints with the Coordinates being locked, but computationally it is just another constraint equation. The Coordinate therefore acts to manage the bookkeeping (which DOF, constraint, etc.) and provide an interface that has context.

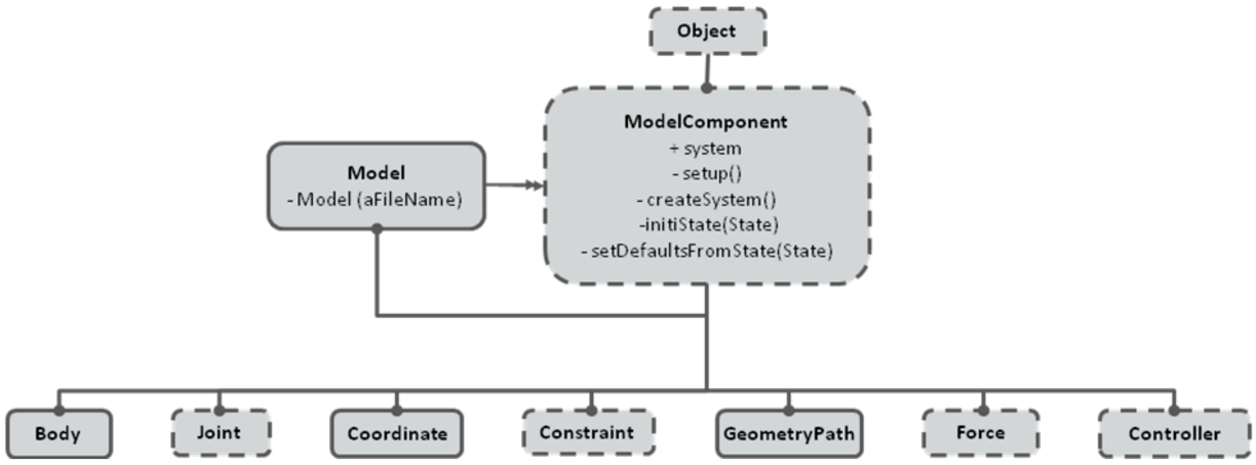


Figure 13-3: OpenSim Model and its ModelComponents

All model components in OpenSim have a similar responsibility to create their underlying system representation (createSystem()). A setup() method ensures that a model is appropriately defined (for example, a Body is being connected to a parent that exists) before creating the system. Two additional methods allow the ModelComponent to initialize the state of the system (from default properties) and also to hold the existing state in the ModelComponent's defaults. For example, a Coordinate's default may indicate that it should be locked, in which case its initState would set the state of its underlying constraint as "enabled". Similarly, after performing an analysis to find the coordinates to satisfy a static pose, calling setDefaultsFromState(state) will update the Coordinate's default values for the coordinate value from the desired state. Next time the model is initialized, it will be in the desired pose.

13.4 OpenSim Application Programming Interface (API)

In order to build custom components, it is necessary to have a general understanding of which objects (classes) are responsible for what actions/behaviors. The functions (methods) that OpenSim's public classes provide (that other applications/programs can call) define its Application Programming Interface or API.

We have already seen four methods that a model component must implement to behave as a ModelComponent in OpenSim. This defines the ModelComponent interface. Each type of ModelComponent, in turn, specifies additional methods in order to satisfy that type of component. For example, a Force in OpenSim must implement a computeForce() method (in addition to the ModelComponent methods), a Controller must implement computeControls(), etc. The set of all Classes and their interfaces defines the OpenSim API.

The OpenSim API is undergoing rapid development and improvement. We therefore rely on Doxygen to automatically generate html documentation of the latest source, which describe the classes that are available and the accessible methods. The Doxygen pages can be viewed using a web browser and are available with your OpenSim installation in: <OpenSim_Install_Dir>/sdk/doc/index.html. This provides the latest organization of the available classes where one can see the list of available controllers, for example.

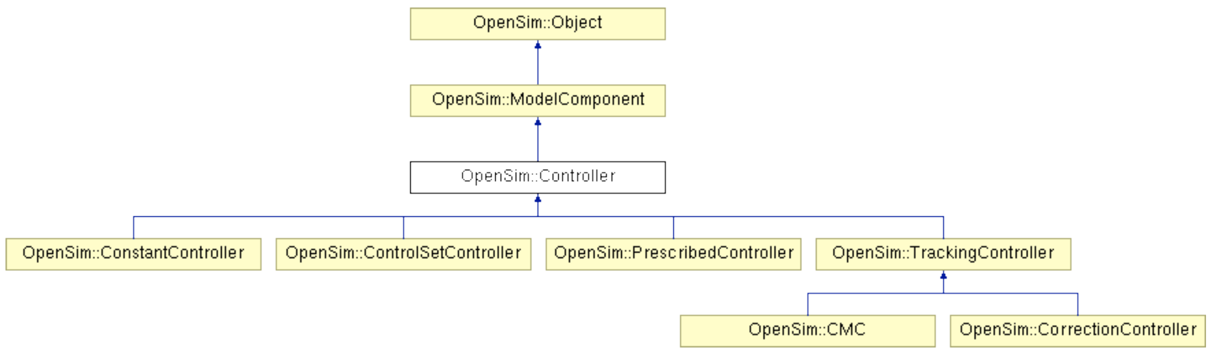

OpenSim::Object
OpenSim::ModelComponent
OpenSim::Controller
OpenSim::ConstantController
OpenSim::ControlSetController
OpenSim::PrescribedController
OpenSim::TrackingController
OpenSim::CMC
OpenSim::CorrectionController

Figure 13-4: Example Doxygen documentation for available controllers

13.5 What is an OpenSim plug-in?

When creating a new component (like a force, controller) or a new analysis, you may want to include it in an existing model, run it with existing tools, and/or share your contribution with colleagues. An OpenSim plug-in is a way of packaging of your code in a dynamically linked library so that an existing OpenSim application can recognize it, load it, and make your code “runnable”. For an example of creating an analysis as a plug-in please see <OpenSim_Install_Dir>/sdk/examples/plugin.

13.6 What is an OpenSim “main” program?

A main program in C/C++ results in a standalone executable that you can run from a command prompt or by double clicking in Windows. All C/C++ programs have a main() function, which can be as simple as printing “Hello World” or it can invoke several libraries to produce complex applications, like Word and Excel. By including the OpenSim libraries, your main program can call the OpenSim API, and you may also include any other (C++)

libraries that provide additional computational and/or visualization resources. Main programs are extremely flexible, but they are particularly useful for streamlining/automating processes independent of the GUI. For example, ik.exe, id.exe, and cmc.exe (available with the OpenSim distribution) are main programs that take setup files and perform tasks related to the OpenSim workflow. Alternatively, users have created their own main programs to systematically scale strengths of all muscles in a model, run forward simulations with their own controllers, perform design optimizations, etc. An advantage of a main program (compared to a plug-in) is that any classes you define in the project are immediately useable by your program. This can make prototyping and testing of your new component or analysis faster and easier without having to wrap, load, and call your plug-in from the GUI.

13.7 OpenSim Developer's Guide

The developer's guide provides a step-by-step example of calling the OpenSim API to build a model, including muscles and contact forces, and to perform a simulation in a main program. Please refer to the OpenSim Developer's Guide, available at https://simtk.org/docman/?group_id=91 for more details.

13.8 Command Line Utilities

All of the OpenSim Tools are available as command-line utilities that take as input the same setup (or settings) file loaded into or saved from the OpenSim GUI application. For example, to perform Inverse Kinematics from the command line (the Command Prompt in Windows) one can execute the following command:

```
ik -S arm26_Setup_InverseKinematics.xml
```

Similarly, this command line arguments work for CMC or any other tool, with the complete set of command line executables available in <OpenSim_Install_Dir>/bin. In addition to the `-`***`S`***`etup` option, there are `-`***`H`***`elp` , `-`***`P`***`rint`***`S`***`etup` and `-`***`P`***`roperty`***`I`***`nfo` options. Help provides this list of options. Print Setup prints a default setup file for that Tool with all available properties (XML tags) for Tool settings.

The `-PropertyInfo` option can be a very handy resource to obtain information about existing settings for Tools and ModelComponents including the XML tags needed in the model and/or setup file. This is the same information listed in the "Available Objects..." panel under the Help menu in the OpenSim GUI. Executing `ik -PI` lists all the available classes (components, analyses, utilities and tools) available in OpenSim. For more information about a particular object, such as adding a point constraint to the model, executing
`ik -PI PointConstraint` yields:

```
PROPERTIES FOR PointConstraint (5)
1. isDisabled
2. body_1
3. body_2
4. location_body_1
5. location_body_2
```

The information returned lists the properties for defining a point constraint in OpenSim.

13.9 MATLAB Utilities for Data Import

There are several MATLAB scripts for reading .trc, .c3d, .mot, and .sto files into MATLAB and writing out the data file formats required by OpenSim. Scripts provided by the Neuromuscular Biomechanics Lab at Stanford are available on the OpenSim Utilities project on SimTK.org: https://simtk.org/home/opensim-utils. Additional utilities by OpenSim users are posted on SimTK.org and can be found using the search tool on SimTK.org