Getting Started with the Long Jump Challenge

Added by Jennifer Hicks, last edited by Thomas Uchida on May 05, 2014

Download the Long Jump Challenge files here. Updated 2014/04/28.

Provided files and other resources

There are three folders given in the package:

1. **Common** contains .h and .cpp files that have Probes, EventHandlers, and the MuscleLikeCoordinateActuator that are not in the OpenSim distribution. It also has files for the Covariance Matrix Adaptation (CMA) optimizer.
2. **InitialFiles** has a provided model file (*AshbyModelv8_twoConstraints.osim*) and a controls file for a nominal standing long jump (*InitialControllerParameters.sto*). Use this controls file as a template to make new controls files.
3. **nodal** contains a .cpp file that creates the main() program to load a model and either perform an optimization or run a forward simulation with various analyses attached.

This model is based on a model previously developed by Ashby and Delp. Ashby used the model to study the role of arms in a long jump in his 2006 paper, and a more detailed description is provided in his thesis. Differences in this implementation include

- Ligament forces are modeled using CoordinateLimitForce in OpenSim.
- Two point constraints are used at the toe and the heel, rather than just the toe.

Since this problem is a non-convex, non-linear optimization, the CMA optimizer provided and used as default. For more information about how this optimizer works, please refer to its Wikipedia page or tutorial slides. The CMA wrapper provided here also has an option to enable MPI to parallelize calculations for the optimization.

CMake Configuration Options

The root directory of the package has a CMakeLists.txt file that can be used to build the project.

- **ENABLE_MPI**: If checked, the project will be able to use the Message Passing Interface (MPI) standard to allow parallelization of the CMA optimizer.
- **MPI_INSTALL_DIR**: Used for Windows OS. Assumes that the Microsoft HPC Pack has been installed and will be used as the MPI implementation. (If using UNIX, this will be ignored and the MPICH libraries are assumed to be used).
- **NameSpace**: Leave blank if building from source. Use "OpenSim_" when linking to pre-built libraries.
- **OPENSIM_INSTALL_DIR**: Point to the install directory for OpenSim.

Building the project

Building the ALL_BUILD project will build two projects:

1. **Executables - StandingLongJump_nodal**: This will build *StandingLongJump_nodal.exe*. This executable is used for running both a single forward simulation or for performing an optimization.
2. **Libraries - sljCommon**: This will build a dynamic linked library (*sljCommon.dll* or *sljCommon.so*) that contains extra classes not included in the distribution of OpenSim. This includes Probes, EventHandlers, and the MuscleLikeCoordinateActuator needed for the model and optimization as well as an implementation of the CMA optimizer.

Running a single forward simulation with analyses

To run a single forward simulation with analyses, we need to have

1. The executable, *StandingLongJump_nodal.exe*
2. The common library, *sljCommon.dll*. This needs to either be in the same directory as the executable or on the system path for Windows.
3. Input model file without controls (*i.e. AshbyModelv8_twoConstraints.osim*)
4. Controls file (*i.e. InitialControllerParameters.sto*)

Options:

- -m <modelfile.osim>: Specify a model file to use. Default is *AshbyModelv8_twoConstraints.osim* if not specified.
- -cf <controlsfile.sto>: Specify a controls file to use. Default is *InitialControllerParameters.sto* if not specified.

The executable can be run from the command prompt. As an example, you can run the executable as:

```
StandingLongJump_nodal.exe -m newmodel.osim -cf optimizedcontrollerparameters.sto
```

This will output the following files:

- forceReporter.sto: Output from a ForceReporter.
- fwdLog.txt: Contains information about length of simulation, integrator tolerance, and values that are used to calculate the objective function.
- PointKinematics_HEEL_*.sto: PointKinematics analysis of the heel.
- PointKinematics_TOE_*.sto: PointKinematics analysis of the toe.

Performing an optimization

To perform an optimization, we need to have

1. The executable, *StandingLongJump_nodal.exe*
2. The common library, *sljCommon.dll*. This needs to either be in the same directory as the executable or on the system path for Windows.
3. Input model file without controls (*i.e. AshbyModelv8_twoConstraints.osim*)
4. Controls file with the initial guess for the optimizer (*i.e. InitialControllerParameters.sto*)

Options:

- -m <modelfile.osim>: Specify a model file to use. Default is *AshbyModelv8_twoConstraints.osim* if not specified.
- -cf <controlsfile.sto>: Specify a controls file to use. Default is *InitialControllerParameters.sto* if not specified.
- -opt: Flag that specifies that an optimization will be performed. This flag **must** be used when performing an optimization.
- -lambda <int>: CMA optimization parameter specifying number of samples per generation. Default is 100 if not specified.
- -sigma <positive real number>: CMA optimization parameter specifying constant scaling term of initial covariance matrix.
- -maxIters <int>: Optimization parameter limiting number of outer iterations (*a.k.a.* generations for CMA). Default is 1000 if not specified.
- -resume: If *resumecmaes.dat* file is present and the -opt flag is used, this flag specifies the CMA optimizer to resume from a previous optimization corresponding to the file. This will override any -sigma setting, but -lambda and -maxIters can still be specified. If *resumecmaes.dat* is not present, then the optimizer will begin a new optimization. If the -opt flag is not used, then the -resume flag will have no effect and a single forward simulation will be performed.

The executable can be run from the command prompt. As an example, you can run the executable as:

```
StandingLongJump_nodal.exe -opt -m newmodel.osim -cf initialcontrollerparameters.sto -lambda 50 -sigma 0.01
-maxIters 2000
```

If the ENABLE_MPI flag was set to true in CMake and the proper MPI libraries are linked, you can use more than one thread to parallelize calculations. An example way to run this (in this example using 8 threads) is:

```
mpiexec -n 8 StandingLongJump_nodal.exe -opt -m newmodel.osim -cf initialcontrollerparameters.sto -lambda 50
-sigma 0.01 -maxIters 2000
```

An example to resume a previous CMA optimization is to run the following in the command prompt:

```
mpiexec -n 8 StandingLongJump_nodal.exe -opt -resume -m newmodel.osim -cf controllerparameters.sto -lambda 50
-maxIters 2000
```

This will output the following files specific to this project:

- ControllerParameters_gen<int>.sto: This is the controller parameters corresponding to the best objective function value for the generation number. This is set to print the 1st and every 100th generations.
- optimizedControls.sto: Controller parameters corresponding to the best objective function found.
- optimizedModel.osim: Model with the controller and control parameters corresponding to optimizedControls.sto.
- optLog.txt: Lists the options and parameters used for the optimization.
- outputlog.txt: Tab-delimited file with information about the progression of the optimizer. The columns contain: 1) generation number, 2) best objective function value, 3) sigma for the current generation, and 4) corresponding controls for the best objective function. This is printed for the 1st and every 50th generations.
- resumecmaes.dat: Generated by the CMA optimizer to allow the user to resume an optimization.

Objective Function

Although you may make any changes to the objective function to help solve your specific optimization problem, final performance will be assessed using the provided objective function. This objective function rewards increased jump distance and penalizes use of ligaments, represented by CoordinateLimitForces, and slipping at the toe and heel point contacts.

$$f = -d + w_1(K_{landing_x} + K_{landing_y}) + w_2(K_{ligament}) + w_3(K_{slip})$$

- d : Distance from the initial horizontal toe position to the horizontal heel position at landing in meters
- $w_1 = 10.0, w_2 = 1.0 \times 10^{-4}, w_3 = 1.0$: Weights for 1) landing, 2) ligament, and 3) slip penalties.
- $K_{landing_x}$: Horizontal landing penalty if the horizontal position of the center of mass at landing is too far behind the horizontal position of the heel.
- $K_{landing_y}$: Vertical landing penalty if the verical position of the center of mass at landing is too far behind the vertical position of the heel.
- K_{slip} = : Slip penalty.

The landing penalty is broken up into two parts. The x-component penalizes a jump if the horizontal position of the center of mass is too far behind the horizontal position of the heel at the time of landing.

$$K_{landing_x} = \max(q_{heel_x} - q_{COM_x} - \delta_{COM_x}, 0)$$

- q_{heel_x} : Horizontal position of the heel at the time of landing.
- q_{COM_x} : Horizontal position of the center of mass at the time of landing.
- $\delta_{COM_x} = 0.17m$: If the horizontal distance to the heel from the center of mass is larger this value, a penalty is applied. This value is determined by experimental data.

The y-component penalizes a jump if the horizontal position of the center of mass is too low at the time of landing.

$$K_{landing_y} = \max(q_{heel_y} - q_{COM_y} + \delta_{COM_x}, 0)$$

- q_{heel_y} : Vertical position of the heel at the time of landing.
- q_{COM_y} : Vertical position of the center of mass at the time of landing.
- $\delta_{COM_y} = 0.62m$: If the center of mass is not at least this distance above the heel, a penalty is applied. This value is determined by experimental data.

The ligament penalty is used to determine if the model would injure itself by damaging the ligaments due to hyperextension or hyperflexion. This is quantified by the following equation:

$$K_{ligament} = \int_0^{t_f} \sum_{i=1}^{n} T_i^2 dt$$

- t_f : Final time of the simulation, when the feet make first contact at landing.
- $n = 4$: The number of CoordinateLimitForces in the model that represent ligaments.
- T_i : The *i*th CoordinateLimitForce.

The model's foot cannot slip with respect to the ground due to the point constraints, so a penalty is added by comparing the horizontal and vertical forces applied to foot from the ground. The slip penalty is represented by the following equation:

$K_{slip} = \sum_{i=1}^{m} \max(|F_x| - \mu F_y, 0)$

- $F_x^{(i)}$: Horizontal force applied to the foot from the ground at the *i*th constraint.
- $F_y^{(i)}$: Vertical force applied to the foot from the ground at the *i*th constraint.
- $m = 2$: Number of point constraints modeling contact.
- $\mu = 0.8$: Coefficient of static friction.

Qualification Round, Part 1: Optimization

The given controls file yields an objective function value of -2.22. To test that the optimization is working, run the optimization using the given controls file as the initial guess controls file with the following settings:

- lambda = 8
- sigma = 0.001
- maxIters = 100

On a single core, the optimization may take up to about 5 minutes. Although the optimization will not likely reach the original objective function value, you should see the objective function decreasing with more iterations. Also note that each time the optimizer is run, the optimizer will find a different objective function value because it is a stochastic optimizer. Expect to find a value of around -2.18 or better.

Hop to it!