Author: Mohammad Kassas

Date: 2/1/2019

Overview:

We are designing a harness that can load and unload the body to study balance control. The harness will be attached to a balance robot that will simulate the standing balance control of humans. Because our experiments are on the order of hours, we require these harnesses to both safely and comfortably load and unload the body.

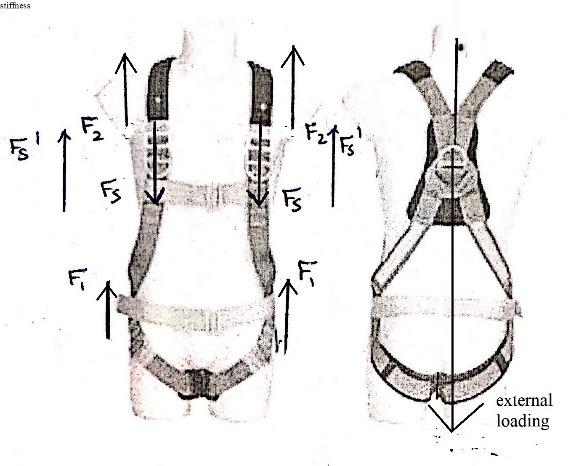
A picture containing indoor, wall

Description automatically generatedA picture containing wall, indoor, toy

Description automatically generated

*Fig1a&b: unloading and loading harness with the stiffness labels. Refer to “our model is parametrized by” to see what k1, k2, and ks represent.*

We hypothesize that a harness that evenly distributes loads between the attachment points to the body will be the most comfortable. To study this, we need to first optimize a harness design that achieves even load distributions over a range of body configurations of interest.

To model load distributions, we used an OpenSim musculoskeletal model of the full body with a simulated harness. We chose OpenSim because it is open source and easily extensible to additional scenarios outside the current scope of the study. Briefly, we used a full body OpenSim musculoskeletal model (Gait10dof18musc.osim) and attached additional rigid body and spring elements to represent the harness, with attachment points at the hips, thighs, and top of the shoulders. We additionally removed all the muscles and locked the joints to create a single rigid body model. We can unlock these joints (e.g. hip flexion) to increase the complexity of the model for future investigations. A picture containing linedrawing

Description automatically generated *Fig2&3*

We are modeling two scenarios: loading (left) and unloading (right). Loading involves the shoulders and the hip. Unloading involves with legs and hip. To model additional harness designs with other attachment points, users can add additional rigid body segments and/or spring elements attached to the body to model additional harness sections and body tissue interactions respectively.

The reason there the reaction forces of Fs are included is because we section the harness through that legs and the shoulder straps, getting two systems. F2 and Fs are one system; F1 and Fs’ are another. F1 represents the hip-body force. F2 is the shoulder-body force for the left image, while being the leg-body force for the right image. Fs is the strap force. (refer to fig2&3)

A close up of a logo

Description automatically generatedA picture containing weathervane

Description automatically generated

*Fig4 Fig5*

Our model is parameterized by:

1. Total body mass: m in mg (fig2&3)
2. Soft tissue material properties (Young’s modulus or stiffness) at the attachment points: k1 and k2 in fig1a&b
3. Harness material properties (Young’s modulus or stiffness) connecting harness sections (e.g. hip to shoulder or hip to legs): ks in fig1a&b.
4. Angle of shoulder-hip or leg-hip straps relative to the vertical.

In the fig4, we used blocks to reperesent the different parts of the harness. Fig5 shows the spring model of the harness. The springs that seem to float are the ones that take into account the forces between harness and body tissue.

For our simulations, we fixed

1. Total body mass (73.062 kg)
2. Soft tissue material properties
   1. hip-body strap stiffness k1: 1000 N/m
   2. shoulder-body strap stiffness k2 (fig1b): 3000 N/m
   3. leg-body strap stiffness k2 (fig1a): 3000 N/m

In order to find the optimal set of parameters, we used the scipy minimize function. We implemented a cost function minimizing , as this represents a balance of load forces on the body. The scipy minimize function can be paired with other cost functions in the future given design requirements.

required software:

1. OpenSim 3.3 <https://simtk.org/frs/index.php?group_id=91>

2) Python 3.6 (we recommend downloading the Anaconda package: <https://www.anaconda.com/distribution/>)

Downloading OpenSim:

1) setup a simTk account

2) https://simtk.org/frs/index.php?group\_id=91

3) go to previous releases and select 3.3

4) download the 64-bit version

Ensure that the Python environment and IDE are 64-bit to match the OpenSim 64-bit version

Linking libraries:

follow the instructions:

https://simtk-confluence.stanford.edu/display/OpenSim/Scripting+in+Python

To test whether or not the linking of libraries worked, in your command window type "import opensim as osim"

If there are not errors; great, it works!

In the case where you get an import error, follow the troubleshooting instructions found in the above link

If an import error occurs, that means that you either have an inconsistent bit version of either program or

you need to "tell" your Python Environment where to locate the OpenSim program on your computer.

Functions:

1. Optimizer\_callBack: runs a simulation and returns the result of the cost function
2. External\_tension: call back function to set the stiffness of the external cable carrying the model during unloading.
3. Shoulder\_hip\_right: call back function to set the stiffness of the right shoulder-hip strap
4. Shoulder\_left\_right: call back function to set the stiffness of the left shoulder-hip strap
5. Hip\_body\_right: call back function to set the stiffness of the right-side strap representing the reaction of hip and tissue
6. Hip\_body\_left: call back function to set the stiffness of the left-side strap representing the reaction of hip and tissue.
7. Shoulder\_body\_right: call back function to set the stiffness of the right-side strap representing the reaction of shoulder and tissue.
8. Shoulder\_body\_left: call back function to set the stiffness of the left-side strap representing the reaction of shoulder and tissue.
9. Angle\_shifter: sets the angle of the straps
10. Length\_calc: calculates the length of the straps
11. Normalize: returns an array that is set in terms of a new reference frame
12. Refactor: returns an array that is set back to the original reference frame

More details on optimizer\_callBack():

The optimizer takes in the parameter array as an input. For our callback, the parameter array is a 2-array with the strap stiffness (ks) and strap angle (alpha) (in order). The callback sets the angles and stiffnesses of the OpenSim model to the current parameter set. It then runs a forward dynamics tool with a force analysis for 3 seconds, which we found was sufficient to reach steady-state. From the force analysis, we extract the forces on the body segments (shoulder or legs and the torso), in the straps, and on the external loading cable. Because our cost function is , we simply use the body segment forces to compute and return the cost.

In order to perform one iteration of the optimizer call back in the OpenSim GUI, follow these instructions:

1. Load the hangingHarnessModel.osim or equivalent from the file tab
2. Only the pelvis\_ty, hip\_ty, legR\_ty, and legL\_ty or pelvis\_ty, hip\_ty coordinates should be unlocked.
   1. In the case of loading, only the pelvis\_ty, hip\_ty, shR\_ty, and shL\_ty coordinates should be unlocked.
3. In the Tool tab, select Forward Dynamics.
4. In the pop window, check “solve for equilibrium states”
5. Go to analyses tab, and select add “Force analysis”
6. Press “run”
7. After running the forward dynamics simulation, the force reporter file will be produced in the same directory as your OpenSim model. The file name starts with the name of your “modelName\_ Force\_Reporter\_forces.sto”
   1. Other outputs:
      1. “modelName\_controls.sto” will be empty in our case
      2. “modelName\_states.sto” will contain the initial states of the model
      3. “model\_Name\_states\_degrees.mot” is a motion file that has the angles of certain bodies of the model.

\*note: the “modelName” prefix is only added to the name of the file if the simulation is run manually through OpenSim. In the case of running simulation through Python for example, it will be “\_ForceReporter\_forces.sto”.

OpenSim components/methods:

For the adept user, included are some notes on the most relevant OpenSim methods and classes. For more information on interfacing with OpenSim, visit https://simtk.org/api\_docs/opensim/api\_docs20b/main.html

a) import opensim as osim: imports all the necessary OpenSim methods

b) osim.Model("..."): loads in the OpenSim model that will be simulated

c) model.getForceSet(): gets all the added external forces that are acting on model

d) model.getForceSet().get(0): gets the base of the first force from the forceSet()

e) PathSpring is a type of spring in OpenSim that only produces force during tension (similar to a string)

f) osim.PathSpring.safeDownCast(model.getForceSet().get(0)): this down casts the force to a PathSpring --> having access to all the PathSpring functions and attributes.

g) <PathSpring Object>.setStiffness(double): sets the stiffness of the spring

h) model.initSystem(): sets up the initial states of the model before simulation

i) osim.ForceReporter(<Model Object>): tells the forward tool to produce a force file after running

j) model.addAnalysis(reporter): adds the force reporter to the list of analyses performed during the simulation

k) osim.ForwardTool(): specifies the type of simulation to be a forward dynamics simulation

l) setModel(<Model Objecy>): sets the model which will be simulated

m) setFinalTime(double): specifies the end time of the simulation

n) osim.Vec3(double, double, double): produces a 3-D vector

o) <PathSpring Object>.setRestingLength(double): sets the resting length of the spring

p) upd\_GeometryPath(): returns a GeometryPath Object

q) <GeometryPath Object>.updPathPointSet(): returns the set of pathpoints that define the geometry of the spring

r) <PathPoint Object>.setLocation(SimTK init State, Vec3): changes the coordinates of the PathPoint

important information:

PathSprings only produce tension forces (no compression)

PathPoint coordinates are specified in reference to parent bodies. In order to find lengths or manipulate

different springs and points, one must normalize everything to a single origin (parent body)

**Release v2:**

Folders:

1. hangingHarnessModel: unloading model with optimizer and assistive functions
2. harnessOptimizers: more complex models that take into account pelvis tilt and/or horizontal forces from the strap on the body during tilting
3. loadingHarnessModel: loading model and assistive functions

The first and the third folder are identical to release v1.

For harnessOptimizers, the final draft of the optimizer lies within the folder horiz\_optimizer. The other files in harnessOptimizers are previous versions of the code which is less modular. Optimizer\_pelvis\_tilt only considers the pelvis tilt without any horizontal forces. However, optimizer\_pelvis\_tilt\_horiz accounts for the horizontal forces associated with pelvis tilt.

In “horiz optimizer”, there are four modular codes:

1. coordinates: respsonsible for housing, normalizing, and refactoring all the coordinates of the bodies and forces
2. geometry\_func: contains sub-programs that change and retrieve the geometry of the straps and springs (angle, length, and GeometryPath)
3. model\_returner: returns an OpenSim model, a copy of that model, a file path, the indices of the forces, the forceSet, and whether it is loading or unloading. Moreover, this file performs all the file I/O to create a copy of the model, so that any alterations would not affect the original model. If you are changing parameters, ensure the paths are consistent and the indices returned align properly to the column indices of the force reporter file (f1Index and f2Index). If you are adding additional complexity to the model or changing the entire model, make sure to have the correct forceSet indices be returned (frcSet). The frcSet returns the indices of force 1 and force 2 (hip-strap and shoulder-strap reaction resp.)
4. vector\_operations: normalizes and refactors the coordinates of the springs to any specified coordinate.