# Building a Dynamic Walker in Matlab

Note: This example is compatible with OpenSim version 4.1

## Objectives

Model development, reduction, and validation are key components in a meaningful simulation study. In this exercise, you will build a simple model for use in a passive dynamic simulation of walking. For the past few decades, passive dynamic systems have been used to study the fundamental principles of locomotion. By studying passive dynamic systems, researchers aim to understand how the geometric and inertial properties of a system influence the occurrence and stability of low-energy gaits. Proper selection of passive system dynamics can reduce the demand on a control system and allow locomotion to be achieved by modulating a small set of signals. The knowledge gained by studying passive dynamic walking can help improve the designs of autonomous legged robots and rehabilitation robots for locomotion.

In this exercise, you will create an un-actuated, four-link walker model using the OpenSim API through Matlab. This simple model could be useful for exploring control and optimization strategies in the OpenSim API. The objectives of this exercise are to learn how to instantiate and connect basic components of an OpenSim model (bodies and joints) and to add more complicated components like Forces and Contact Geometry.

Below are some useful (and necessary) resources that you should read before you begin and keep handy while you build your model;

- Read the Introduction to the OpenSim API section
- Setup your Matlab environment using instructions from the Scripting with Matlab page.
- Bookmark the Common Scripting Commands page for helpful tips and code snippets
- Read the Guide to Using Doxygen pages and bookmark the OpenSim Doxygen documentation
- Bookmark the API Guide on Doxygen.
- Setup a working folder where you will save and test your model.

## Model Description

The model you will create in this exercise consists of a rigid platform and a four-link walker. The angle of the platform is used to control the amount of potential energy converted to kinetic energy with each stride. To maintain a stable motion, energy must be added into the system to overcome losses from ground contact. The pelvis of the walker is connected to the platform by a planar joint, which allows three degrees of motion between the pelvis and the platform. Pin joints are used to connect the thighs to the pelvis and the shanks to the thighs. Contact Spheres and a Contact Half Space are attached—using a Hunt–Crossley contact model—to generate reaction forces.

## Setup

The script for this exercise is included in the OpenSim 4.1 distribution and can be found in your OpenSim resources directory; ***<Resources-Dir>/Code /Matlab/Dynamic_Walker_Builder/.***

You will complete the empty code blocks in the sections that follow, run the script, and inspect the generated OpenSim model. The default units for OpenSim are Newtons, kilograms, meters, and radians. However, certain functions do expect angles in degrees rather than radians, so you should view each class in the Doxygen Documentation to check the input type.

You will be opening and viewing the model in the OpenSim GUI. In the GUI, it will be useful to select to not see a floor so that you will be able to see the entire model motion.

```
%% Import OpenSim Libraries into Matlab
import org.opensim.modeling.*

%% Define key model variables
pelvisWidth = 0.20;
thighLength = 0.40;
shankLength = 0.435;

%% Instantiate an (empty) OpenSim Model
osimModel = Model();
osimModel.setName('DynamicWalkerModel');

% Get a reference to the ground object
ground = osimModel.getGroundBody();

% Define the acceleration of gravity
osimModel.setGravity(Vec3(0, -9.80665, 0));

% TODO: Construct Bodies and Joints Here
% ********** BEGIN CODE **********


% **********  END CODE  **********

% TODO: Construct ContactGeometry and HuntCrossleyForces Here
% ********** BEGIN CODE **********


% **********  END CODE  **********

% TODO: Construct CoordinateLimitForces Here
% ********** BEGIN CODE **********


% **********  END CODE  **********

%% Initialize the System (checks model consistency).
osimModel.initSystem();

% Save the model to a file
osimModel.print('DynamicWalkerModel.osim');
display(['DynamicWalkerModel.osim printed!']);
```

One of the advantages of building Models with API functions vs with XML is the ability to store model parameters in variables. Here, we create variables for the pelvis width and the lengths of the thighs and shanks. These three variables are used throughout the exercise, such as when defining the joints, the display geometry, and the contact elements.

A Model object called "osimModel" is created and the name of the model is set to "DynamicWalkerModel." In the final model file, this has the effect of creating a Model XML element and setting the "name" attribute to *DynamicWalkerModel* (i.e., <Model name="DynamicWalkerModel"> ... </Model>).

The setGravity function sets the direction and magnitude of the acceleration due to gravity using a SimTK::Vec3, and adds the XML element <gravity> 0 -9.80665 0 </gravity> to the model.

The print command creates the resulting OpenSim model file. The print command is called after all of the Model Components (bodies, constraints, contact geometry, controllers, etc.) are added to the Model.

## Add Bodies to the Model

In this section, you will add the platform, pelvis, thigh, and shank segments to the model.

## Create the Platform Body

We will create our first Body, the platform. After initializing the Body object, we will attach visual Geometry to it so we can visualize it in the GUI.

To create a body, you must specify the following:

- Mass
- Location of the center of mass from the body's origin, expressed in the body frame
- The moments and products of inertia of the body about its center of mass, expressed in the body frame

The following API functions are used to create the body:

- OpenSim Body
- SimTK Vec3
- SimTK Inertia

The Geometry class allows you to specify the location (Frame), color, opacity, etc. of visual geometry displayed in the GUI and in the OpenSim API Visualizer. Visual Geometry can be added by using Geometry files, such as .vtp files, and using analytical geometry types for common shapes (Brick, Sphere, Ellipsoid, and Mesh).

In the following code block, we add a Platform body to the model and define a Brick Geometry for display. Copy the code into your source file, placed after the setGravity function and before the print function, as shown in the block of code above.

```
% TODO: Construct Bodies and Joints Here
% ********** BEGIN CODE **********
% Make and add a Platform Body
platform = Body();
platform.setName('Platform');
platform.setMass(1);
platform.setInertia( Inertia(1,1,1,0,0,0) );

% Add geometry to the body
platformGeometry = Brick( Vec3(10,0.05,1) );
platformGeometry.setColor( Vec3(0.8, 0.1, 0.1) );
platform.attachGeometry(platformGeometry);

% Add Body to the Model
osimModel.addBody(platform);

% TODO: Construct Joint Here
% ********** BEGIN CODE **********


% **********  END CODE  **********


% TODO: Construct the next body and joint
% ********** BEGIN CODE **********


% **********  END CODE  **********
```

Note: These functions are assuming that you are using units of kilograms, meters, and seconds.

The constructor for Brick takes a Vec3 that specifies the half-lengths of the brick in the x, y, and z directions. By specifying a small half-length in the Y-direction, this brick will look like a flat platform.

## Adding the Joint for the Platform Body

The figure below is Figure 2 is from Seth, A., Sherman, M., Eastman, P., Delp, S. Minimal formulation of joint motion for biomechanisms. Nonlinear Dyn 62: 291–303.

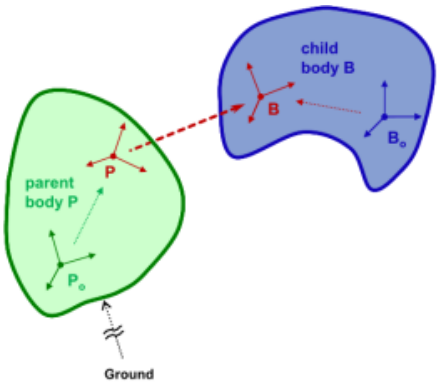


**Fig. 2** A mobilizer (*bold dashed arrow*) is the kinematic relationship between two bodies (a parent $P$ and a child body $B$) parameterized by 1 to 6 mobilities in Euclidian space. Equations of motion are recursively generated in terms of the derivatives of the mobilities and applied forces of each body

To connect your model's bodies together to form a multibody graph, you will need joints. Every model in OpenSim begins with a Ground body. A joint is a set of degrees of freedom that describes the kinematic relationship between two bodies, termed the parent and child bodies.

To allow flexibility in how joints are defined, OpenSim 4.0 introduced the **Frame** class hierarchy. There are three main types of Frames: **Ground** (each model starts with one), **Body**, and **PhysicalOffsetFrame**. All three of these frames are called **PhysicalFrames** because they either are a rigid body or are fixed to a rigid body. In OpenSim 4.x, a joint connects two PhysicalFrames (parent and child).

As an example, OpenSim's pin joint has one degree of freedom between its parent and child frames that allows rotation about the mutual Z-axis of the parent and child frames while restricting all other motion, including translation.

The OpenSim API provides two commonly-used constructors for Joints:

- Provide the parent and child *bodies* along with offsets to the parent and child frames for the joint:
- Provide the parent and child frames directly (new in 4.0):

When providing the parent and child bodies, as seen at the link above the signature above, the transformation for each of the offsets is represented by a pair of three-element "vectors" Vec3's. The first Vec3 holds the measures of the translation vector from the body origin to the joint frame origin. The second Vec3 holds the Euler angles describing the orientation of the joint frame with respect to body frame, using an X-Y-Z body-fixed rotation sequence. (Note: *pa rent* and *child* do not need to be bodies; they can be a PhysicalOffsetFrame or Ground.)

```
% Section: Create the Platform Joint
% Make and add a Pin joint for the Platform Body
locationInParent    = Vec3(0,0,0);
orientationInParent = Vec3(0,0,0);
locationInChild     = Vec3(0,0,0);
orientationInChild  = Vec3(0,0,0);
platformToGround    = PinJoint('PlatformToGround',...  % Joint
Name
                               ground,...             % Parent
Frame
                               locationInParent,...   %
Translation in Parent Frame
                               orientationInParent,...%
Orientation in Parent Frame
                               platform,...           % Child
Frame
                               locationInChild,...    %
Translation in Child Frame
                               orientationInChild);   %
Orientation in Child Frame


% TODO: Set the coordinate properties of the Pin Joint

% Add the PlatformToGround Joint to the Model
osimModel.addJoint(platformToGround);

% TODO: Construct the next body and joint
% ********** BEGIN CODE **********


% **********  END CODE  **********
```

Note: These functions are assuming that you are using units
of kilograms, meters, and seconds.

Check your model in the OpenSim GUI.

## Setting the Properties of the Coordinates in the Pin Joint

The Coordinate objects that define a joint are held in the Joint. Coordinates are ordered and can be accessed using an integer index value. Use the "get" variants if you do not want to edit the returned coordinate, and use the "upd" variants otherwise (e.g., if you want to set the name of a coordinate). The various OpenSim Joints have different numbers of degrees of freedom; the number of Coordinates in a Joint is equal to the number of degrees of freedom of the joint. For example, a PinJoint has only one Coordinate (Rotation about Z-axis).

```
% TODO: Set the coordinate properties of the Pin Joint
platform_rz = platformToGround.upd_coordinates(0);
platform_rz.setRange([deg2rad(-100), deg2rad(100)]);
platform_rz.setName('platform_rz');
platform_rz.setDefaultValue(deg2rad(-10));
platform_rz.setDefaultSpeedValue(0);
platform_rz.setDefaultLocked(true)
```

For one-coordinate joints like PinJoint, you can call
upd_Coordinate() or getCoordinate() without an argument.

Execute the code to create your model; it should be located in your build directory. Check your model in the OpenSim GUI.

## Create the Pelvis Body

In this section, you will create the Pelvis body and connect the platform to the pelvis via a PlanarJoint, which is a 3-degree-of-freedom joint that enables rotation about the Z-axis and translation in the X-Y plane. The starter code below sets up the construction of the Pelvis and Joint between the Pelvis and Platform. Copy the following block of code into your source file directly after the code creating the platform creating the platform and before the print command.

A modeling choice was made here to connect the pelvis to the platform so that pelvis height in relation to the contact surface does not change when the platform rotates. OpenSim can handle any tree topology for the bodies in the system. For example, an alternative model could be created where both the pelvis and platform are connected to Ground.

```
% Section: Create the Pelvis
% Make and add a Pelvis Body
pelvis = Body();
pelvis.setName('Pelvis');
pelvis.setMass(1);
pelvis.setInertia(Inertia(1,1,1,0,0,0));
% Add geometry for display
pelvis.attachGeometry(Sphere(pelvisWidth));
% Add Body to the Model
osimModel.addBody(pelvis);

% Make and add a Planar joint for the Pelvis Body
pelvisToPlatform = PlanarJoint('PelvisToPlatform', platform,
pelvis);
% Update the coordinates of the new joint
Pelvis_rz = pelvisToPlatform.upd_coordinates(0); % Rotation about
z
Pelvis_rz.setRange([-pi, pi]);
Pelvis_rz.setName('Pelvis_rz');
Pelvis_rz.setDefaultValue(0);

Pelvis_tx = pelvisToPlatform.upd_coordinates(1); % T about x
Pelvis_tx.setRange([0, 10]);
Pelvis_tx.setName('Pelvis_tx');
Pelvis_tx.setDefaultValue(0);
Pelvis_tx.setDefaultSpeedValue(0)

Pelvis_ty = pelvisToPlatform.upd_coordinates(2); % Translation
about y
Pelvis_ty.setRange([-5,5]);
Pelvis_ty.setName('Pelvis_ty');
Pelvis_ty.setDefaultValue(thighLength + shankLength);
Pelvis_ty.setDefaultSpeedValue(0)

% Add Joint to model
osimModel.addJoint(pelvisToPlatform)
```

The constructor for the Sphere Geometry takes a radius value.

Execute the code to create your model. Check your model in the OpenSim GUI.

### Create the Thigh and Shank Bodies

Assemble the remaining six segments (right and left thigh, shank, and foot segments) and joints (right and left hip, knee, and ankle joints) with the properties found below.

Make sure to set the joints' "locationInParent" and "locationInChild" so that the mass center for the thigh is at the midpoint between the hip and knee joints, and the mass center for the shank is between the center of the shank. Also, consider that, in the default pose, the model should resemble the screenshot on the home page for this example.

For each of the new segments, attach an Ellipsoid Geometry with dimensions such that the length along the long axis is the segment length, and the length of the other two axes is one-tenth the segment length.

When you are finished, execute the code, produce a model, and verify the model in the OpenSim GUI.

Desired Properties for the Remaining Bodies:

- **Right Thigh**: Name: RightThigh, CoM Location: (0,0,0) m, Mass: 1 kg, BodyInertia: (2,2,0.02,0,0,0) kg-m$^2$
  - **Right Thigh Geometry**: Ellipsoid where X & Z radii = thighLength/10, Y radii = thighLength/2
- **Right Hip Joint**: JointName: RightThighToPelvis, CoordinateName: RHip_rz, DefaultValue: 30 deg, Range: -100 deg to 100 deg
- **Left Thigh**: Name: LeftThigh, CoM Location: (0,0,0) m, Mass: 1 kg, BodyInertia: (2,2,0.02,0,0,0) kg-m$^2$
  - **Left Thigh Geometry**: Ellipsoid where X & Z radii = thighLength/10, Y radii = thighLength/2
- **Left Hip Joint**: JointName: LeftThighToPelvis, CoordinateName: LHip_rz, DefaultValue: -10 deg, Range: -100 deg to 100 deg
- **Right Shank**: Name: RightShank, CoM Location: (0,0,0) m, Mass: 1 kg, BodyInertia: (1,1,1,0,0,0) kg-m$^2$
  - **Right Shank Geometry**: Ellipsoid where X & Z radii = thighLength/10, Y radii = thighLength/2
- **Right Knee Joint**: JointName: RightShankToThigh, CoordinateName: RKnee_rz, DefaultValue: -30 deg, Range: -100 deg to 0 deg

- **Left Shank**: Name: LeftShank, CoM Location: (0,0,0) m, Mass: 1 kg, BodyInertia: (1,1,1,0,0,0) kg-$m^2$
    - **Left Shank Geometry**: Ellipsoid where X & Z radii = thighLength/10, Y radii = thighLength/2
- **Left Knee Joint**: JointName: LeftShankToThigh, CoordinateName: LKnee_rz, DefaultValue: -30 deg, Range: -100 deg to 0 deg

## Add ContactGeometry and Forces to the Model

Forces can be applied between bodies in OpenSim by adding ContactGeometry and Force components to the model. When two contact objects come into contact, an OpenSim Force is responsible for calculating the magnitude and location of the resulting contact force and applying it to the appropriate bodies. For our model, we will model the platform as an infinite half-space for contact and use spheres positioned along the body to avoid penetration of the segments with the platform. For more complex geometries, there is a ContactMesh object that can create a triangular mesh contact surface on a body.

The following API functions can be used to create ContactGeometry objects:

- OpenSim ContactHalfSpace
- OpenSim ContactSphere

### Add Contact Geometry to the Platform and the Right Hip

```
% TODO: Construct ContactGeometry and HuntCrossleyForces Here
% ********** BEGIN CODE **********
contactSphereRadius = 0.05;

% Make a Contact Half Space
groundContactLocation = Vec3(0,0.025,0);
groundContactOrientation = Vec3(0,0,-1.57);
groundContactSpace = ContactHalfSpace(groundContactLocation,...

groundContactOrientation,...
                                    platform);
groundContactSpace.setName('PlatformContact');
osimModel.addContactGeometry(groundContactSpace);

% Make a Right Hip Contact
RightHipContactSphere = ContactSphere();
RightHipContactSphere.setRadius(contactSphereRadius);
RightHipContactSphere.setLocation( Vec3(0,0,pelvisWidth) );
RightHipContactSphere.setFrame(pelvis)
RightHipContactSphere.setName('RHipContact');
osimModel.addContactGeometry(RightHipContactSphere);
```

### Add ContactSpheres to the Left Hip, Knees, and Feet

Finish the previous block of code by adding a contact sphere for the left hip, left and right knees, and left and right feet. Assign the knee and feet contact spheres to the shank bodies and the hip contact spheres to the pelvis.

Simply adding ContactGeometry to a model is not sufficient to generate contact forces between bodies; we must also add a Force object that describes the forces that are generated when two ContactGeometry objects collide. You will do this in the next section.

### Add ContactForces to the Model

The following code is used to create HuntCrossleyForce objects for the contact spheres.

```
% Section: Add HuntCrossleyForces
stiffness           = 1000000;
dissipation         = 2.0;
staticFriction      = 0.8;
dynamicFriction     = 0.4;
viscousFriction     = 0.4;
transitionVelocity  = 0.2;

% Make a Hunt Crossley Force for Right Hip and update parameters
HuntCrossleyRightHip = HuntCrossleyForce();
HuntCrossleyRightHip.setName('RHipForce');
HuntCrossleyRightHip.addGeometry('RHipContact');
HuntCrossleyRightHip.addGeometry('PlatformContact');
HuntCrossleyRightHip.setStiffness(stiffness);
HuntCrossleyRightHip.setDissipation(dissipation);
HuntCrossleyRightHip.setStaticFriction(staticFriction);
HuntCrossleyRightHip.setDynamicFriction(dynamicFriction);
HuntCrossleyRightHip.setViscousFriction(viscousFriction);
HuntCrossleyRightHip.setTransitionVelocity(transitionVelocity);
osimModel.addForce(HuntCrossleyRightHip);
```

A common bug is misspelling the name of the contact element in the call to addGeometry(). Matlab does not know if you have named the correct element, so the program will execute without an error in Matlab, but the resulting model will generate an error when it is loaded into OpenSim.

### Add HuntCrossleyForces Between the Remaining ContactSpheres and the Platform

For the remaining five spheres, create a new set of HuntCrossleyForce objects (using the same parameter values as above) and add them to the model.

Execute the code to create your model. Check your model in the OpenSim GUI.

## Add Coordinate Limit Forces

In order to enforce the coordinate limits set in the joint definitions, we will create a CoordinateLimitForce for the hip and knee joints that will apply a force similar to a spring and damper to enforce the joint limits.

```
% TODO: Construct CoordinateLimitForces Here
% ********** BEGIN CODE **********
% Define Coordinate Limit Force Parameters
upperStiffness = 0.5;
lowerStiffness = 0.5;
kneeUpperLimit = 0;
kneeLowerLimit = -140;
hipUpperLimit = 100;
hipLowerLimit = -100;
damping = 0.025;
transition = 5;

% Make a Right Hip Coordinate Limit Force
RHipLimitTorque = CoordinateLimitForce();
RHipLimitTorque.setName('RHipLimitTorque')
RHipLimitTorque.set_coordinate('RHip_rz');
RHipLimitTorque.setUpperStiffness(upperStiffness);
RHipLimitTorque.setLowerStiffness(lowerStiffness);
RHipLimitTorque.setUpperLimit(hipUpperLimit);
RHipLimitTorque.setLowerLimit(hipLowerLimit);
RHipLimitTorque.setDamping(damping);
RHipLimitTorque.setTransition(transition)
osimModel.addForce(RHipLimitTorque);
```

Search for CoordinateLimitForce in the OpenSim API documentation and find the method for constructing the CoordinateLimitForce with parameters. For the remaining three Leg coordinates, create a new set of CoordinateLimitForce objects (using the same parameter values as above) and add them to the model.

Execute the code to create your model. Check your model in the OpenSim GUI.

## Run a Simulation

We have finished writing the code to build the model. Once you have executed your code and printed your model. Load the model in the OpenSim GUI and run a forward simulation from 0 to 2 seconds. Make sure your model is in the default pose before running the forward simulation.

The model will simulate forward, but will likely fall immediately. Passive Dynamic walkers are sensitive to initial conditions; you can change the initial pose in the GUI and re-run the forward simulation. In the next example, we will augment the model to improve its stability.

## References


1. Sherman, M.A., Seth, A., Delp, S.L. (2011). Simbody: multibody dynamics for biomedical research. Procedia IUTAM 2:241–261.
2. Seth, A., Sherman, M.A., Eastman, P., Delp, S.L. (2010). Minimal formulation of joint motion for biomechanisms. Nonlinear Dynamics 62:291–303.