Creating a Controller Part Two

The steps covered in part two are:

Writing the main()

To run a forward dynamics simulation using our controller, we can write a main program (as in Chapter 2). Our main program below initializes the model, attaches a controller to the model, and runs a forward dynamics simulation.

```
int main()
{
  try {
    // Create an OpenSim model from the model file provided.
    Model osimModel( "tugOfWar_model_ThelenOnly.osim" );
```

Creating the controller and attaching it to the model

Next in our main program, we create an instance of the TugOfWarPDController, passing it the model we read in as well as values we choose for the position and velocity gains:

```
  // Create the controller.
  // Create the controller.
  TugOfWarController *controller = new TugOfWarController(kp, kv);

  // Add the controller to the Model.
  osimModel.addController( controller );
```

Initializing the system

Next, we set the initial states of the system, which include the position and speed values for all the coordinates in the model, as well as the muscle activations and fiber lengths:

```
  // Initialize the system and get the state representing the
  // system.
  SimTK::State& si = osimModel.initSystem();

  // Define non-zero (defaults are 0) states for the free joint.
  CoordinateSet& modelCoordinateSet = osimModel.updCoordinateSet();

  // Get the z translation coordinate.
  Coordinate& zCoord = modelCoordinateSet.get( "blockToGround_zTranslation" );

  // Set z translation speed value.
  zCoord.setSpeedValue( si, 0.15 * Pi );

  // Define the initial muscle states.
  const Set<Actuator>& actuatorSet = osimModel.getActuators();
  Muscle* muscle1 = dynamic_cast<Muscle*>( &actuatorSet.get(0) );
  Muscle* muscle2 = dynamic_cast<Muscle*>( &actuatorSet.get(1) );
  if((muscle1 == NULL) || (muscle2 == NULL)){
  throw OpenSim::Exception("ControllerExample: muscle1 or muscle2 is not an ActivationFiberLengthMuscle and
example cannot proceed.");
  }
  muscle1->setDefaultActivation( 0.01 ); // muscle1 activation
  muscle1->setDefaultFiberLength( 0.2 ); // muscle1 fiber length
  muscle2->setDefaultActivation( 0.01 ); // muscle2 activation
  muscle2->setDefaultFiberLength( 0.2 ); // muscle2 fiber length
```

Creating the integrator

We create the integrator for the simulation in order to perform the numerical integration of the system equations of motion during the forward dynamics simulation.

```
// Create the integrator and manager for the simulation.
SimTK::RungeKuttaMersonIntegrator integrator( osimModel.getMultibodySystem() );
integrator.setAccuracy( 1.0e-4 );
Manager manager( osimModel, osimModel.getMultibodySystem(),integrator );
```

Running the simulation

We run the simulation by setting the initial and final times for the integrator and then instructing the manager to integrate starting from the initial state of the system.

```
// Integrate from initial time to final time.
manager.setInitialTime( initialTime );
manager.setFinalTime( finalTime );
std::cout << "\n\nIntegrating from " << initialTime << " to " << finalTime << std::endl;
manager.integrate( si );
```

At this point, you can run the main program. We can then visualize the result of the forward dynamics simulation to see how well the controller was able to make the model follow the desired trajectory.

Next: Creating an Optimization

Previous: Creating a Controller Part One

Home: Scripting and Development | Developer's Guide | Adding New Functionality