How to compute muscle moment arm with Simbody

Michael Sherman, 12 Oct 2010

OpenSim provides to its biomechanics users a calculation called "muscle moment arm" which captures to some degree the leverage of a particular muscle with respect to a particular joint, while in a particular configuration. For simple cases, this is the same as the conventional moment arm calculation in mechanical engineering. A straight-line muscle whose origin and insertion points connect two adjacent bodies connected by a pin joint is the simplest case. In practice, however, muscle may span several joints (wrist, neck, spine) and will follow contorted paths crossing over various curved bone surfaces. For these situations we need a careful definition of "muscle moment arm" that is analogous to the mechanical engineering concept and of use to biomedical researchers.

There are other related quantities that can easily be confused with moment arm, such as the amount of joint acceleration that a muscle can produce. That will be equivalent to moment arm in simple cases but confusingly different in more complex situations. Here we will give a precise definition of what we mean by "muscle moment arm" and, given that definition, how that quantity may be efficiently calculated. We will not attempt to address any deeper questions such as when you should be interested in moment arm or how you should use it.

Assumptions

A muscle is considered to connect body A to body B via a minimum-length curved path connecting a point on body A to a point on body B, possibly passing frictionlessly over wrapping surfaces on these or other bodies. No particular connectivity is assumed for these bodies in the multibody model of the skeleton. For example, there might be a patella with wrap surface hanging off one of the bodies and driven from an arbitrary mobility.

Assumption 1: The path includes both muscle and tendon segments whose relative lengths vary dynamically, but we are assuming that the total length l is just a kinematic quantity $l(q)$ (typically representing the shortest path) that can be calculated once the poses of all the bodies are known via specification of the generalized coordinates q. So for any given muscle we assume

$$l = l(q) \tag{1}$$

Assumption 2: Force generation by the muscle is completely characterized by a scalar tension $s \geq 0$ acting uniformly along the path, such that the set of spatial forces* F applied by the muscle in a given configuration is just a linear function of s:

$$F(s) = Ts \tag{2}$$

where $T=T(q)$ is the muscle's instantaneous "force transmission matrix". We expect that a muscle element can efficiently compute F given s, although T is not explicitly available.

Definition of "moment arm"

Moment arm is a property of a given muscle about a particular angular quantity of interest we'll call θ, measured at a particular configuration q of the multibody system. It is a scalar geometric quantity $r=r(q)$, with units of length. r relates an infinitesimal change in muscle length l to the corresponding unitless change in angle θ, via the definition

$$r = \frac{dl}{d\theta} \tag{3}$$

Moment arm is defined only for angles θ which determine muscle length l kinematically, that is, we expect a virtual displacement $d\theta$ to produce a virtual length change dl that depends only on q, not on velocities, forces, or masses. All generalized coordinates q that can affect θ are thus assumed to be coupled; any that are not coupled explicitly by a constraint will be held constant during the moment arm calculation (that is, they are "coupled" with a coupling factor of 0).

An OpenSim user requesting a moment arm calculation will specify a muscle, and choose one of the available angular quantities for that muscle. Currently only a generalized coordinate subset is available for this purpose, so θ will always be one of the q's in the model. When the angle of interest is actually the sum of several coupled generalized coordinates, one of them (called the *independent coordinate*) is scaled so that it reads as the total angle rather than just the angle it controls directly. Coupler constraints are added separately to enforce the desired cooperative motion of the other *dependent coordinates*. The algorithm below does not require this approach, but there must be a way to calculate θ from the q's and $\dot{\theta}$ from the $\dot{q}$'s.

Note that a muscle path may cross several independent coordinates, such as a hip and knee angle. When moment arm is calculated for one of those coordinates, the other is held rigid (meaning again that it is seen as coupled, but with a coupling factor of 0). Muscles crossing wrist, ankle, neck, and spine may be modeled with a single independent coordinate measuring the total angle, while several dependent coordinates are coupled to it.

---

* A spatial body force F^B is a pair of vectors: a moment applied to body B and a force applied at the body B origin; F here is a stacked vector of such spatial body forces.

Ways to calculate moment arm

Starting with the definition, there are a variety of ways to calculate moment arm differing in precision, implementation difficulty, and conceptual difficulty.

Finite differencing

We can calculate $r=dl/d\theta$ directly by finite differencing. That is, we can make a small perturbation $\Delta\theta$, satisfy all position constraints, update geometric calculations, and measure the resulting change Δl. The advantage of this method is that it directly implements the definition, and it is conceptually very simple. However, it has several drawbacks: it produces an approximate answer, and involves linearization difficulties due to the complex path geometry and the need to ensure satisfaction of the nonlinear position constraints. Also, because this is done at the position level it includes only holonomic constraints, and cannot account for nonholonomic constraints such as rolling.

Velocity-level kinematics

An easier and exact computation is available using velocities, since we have

$$r = \frac{dl}{d\theta} = \frac{dl/dt}{d\theta/dt} \equiv \frac{\dot{l}}{\dot{\theta}} \tag{4}$$

That is, if we can calculate $\dot{l}(\dot{\theta})$ then we need only enforce $\dot{\theta}=1$ (for example), satisfy all velocity constraints, then calculate $r=\dot{l}(1)$. This is probably the best way to calculate moment arm provided the operator $\dot{l}(\dot{\theta})$ is available. Unfortunately, it can be difficult to calculate $\dot{l}$ so we would like to find an alternative.

Partial velocity method

By assumption (1) above, we have $l=l(q)$ so

$$\dot{l} \triangleq \frac{dl}{dt} = \sum_i \frac{\partial l}{\partial q_i}\frac{\partial q_i}{\partial t} = P\dot{q} \tag{5}$$

for a row matrix $P(q)$ whose i^{th} entry is the scalar $p_i=\partial l/\partial q_i$. If we can write $\dot{q}=C\dot{\theta}$ for some coupling matrix $C(q)$ (a column with entries c_i), then from equation (5) we have

$$\dot{l} = PC\dot{\theta} \tag{6}$$

Then comparing (6) with (4) and noting that transposing a scalar doesn't change it, we have

$$r = PC = C^T P^T \tag{7}$$

If we had an explicit representation of P, this would be a very nice way to calculate r. However, this would again depend on being able to calculate $\dot{l}(\dot{q})$ which we're assuming is difficult. Since

generalized forces are dual to generalized speeds, we'll look at how to use forces instead of velocities to calculate *r*.

Generalized force method

Simbody can map body spatial forces *F* to generalized forces *f* via an operator that calculates

$$f = J^T F \tag{8}$$

where *J* is the system Jacobian (partial velocity matrix) that maps generalized speeds to the body spatial velocities they produce. (This is due to the dual nature of generalized velocity and force bases.)

Assumption (2) tells us how to calculate *F* from a given muscle tension scalar *s*, using the muscle's force transmission matrix *T*. Substituting (2) into (8):

$$f(s) = J^T T s \tag{9}$$

The column matrix $J^T T$ maps tension to generalized force; the dual problem mapping generalized speeds to $\dot{l}$ is then $\dot{l} = (J^T T)^T \dot{q}$ (proof?). From equation (5) it follows that $J^T T = P^T$. Substituting into equation (7) and using equation (9) gives

$$r = C^T P^T = C^T (J^T T) = C^T f(s) / s \tag{10}$$

This gives us the algorithm we need for calculating moment arm without knowing how to calculate $\dot{l}$ directly:

1. Determine the coupling matrix *C* for the angular quantity of interest θ (see below)
2. Apply unit tension *s*=1 to the muscle of interest and map to muscle forces *F*(*s*) using operator (2)
3. Use the Simbody operator (8) to map muscle forces *F*(*s*) to generalized forces *f*(*s*) (see below)
4. Use equation (10) to compute $r=C^T f / s$ (see below)

1. Calculating the coupling matrix *C*

Set $\dot{\theta} = \dot{\theta}_0 = 1$, use Simbody's `project()` operator to find a least squares solution that satisfies the velocity constraints (this will change $\dot{\theta}$ as well). Determine the new value for $\dot{\theta}$, and call it $\dot{\theta}_1$. Now each $\dot{q}_i = c_i \dot{\theta}_1$, so $c_i = \dot{q}_i / \dot{\theta}_1$ and we have determined $C = \dot{q} / \dot{\theta}_1$.

Code example:

```
// Calculate coupling matrix C:

// Assume "mobod" is a mobilized body whose 0th mobility has been scaled
// so that its generalized coordinate is the angle theta.
```

```
state.updU() = 0;
mobod.setOneU(state, MobilizerUIndex(0), 1); // thetadot_0 = 1
system.project(state, 1e-10,
    yWeights, cWeights, yErrEst, // dummies; see below
    System::ProjectOptions::VelocityOnly);
// Now calculate C.
const Vector C = state.getU() / mobod.getOneU(state, MobilizerUIndex(0)); // / thetadot_1
state.updU() = 0;

// Note: you can declare these dummies for the project() call above.
const Vector yWeights(state.getNY(), 1);
const Vector cWeights(state.getNMultipliers(), 1);
Vector yErrEst;
```

3. Mapping spatial forces to generalized forces

If you can collect the muscle's generated spatial body forces *F* into an array with an entry for each body (zero where the muscle does nothing), you can generate the equivalent generalized forces using Simbody's oddly-named method `calcInternalGradientFromSpatial()`.

Code example:

```
// Calculate the joint torques f equivalent to the muscle forces F, method 1:

Vector_<SpatialVec> F; // Indexed by MobilizedBodyIndex
muscle.setTension(s);
// ... obtain F from the muscle somehow
Vector f;
matter.calcInternalGradientFromSpatial(state, F, f);
```

An alternative is to let Simbody gather up all the body and generalized forces produced by all force elements, and work with the difference between inactive and activated muscle.

Code example:

```
// Calculate the joint torques f equivalent to the muscle forces F, method 2:

Vector f0, f1, f;
// Start with the muscle off
muscle.setTension(0);
system.realize(Stage::Dynamics);
matter.calcInternalGradientFromSpatial(state,
    system.getRigidBodyForces(state,Stage::Dynamics),
    f0);
f0 += system.getMobilityForces(state,Stage::Dynamics);

// Now turn on the muscle
muscle.setTension(s);
system.realize(Stage::Dynamics);
matter.calcInternalGradientFromSpatial(state,
    system.getRigidBodyForces(state,Stage::Dynamics),
    f1);
f1 += system.getMobilityForces(state,Stage::Dynamics);

// f is the change in generalized forces due to the muscle
f = f1 - f0;
```

4. Calculating the moment arm

With `C` and `f` calculated as above, the moment arm calculation is easy:

```
Real momentArm = ~C * f / s;
```