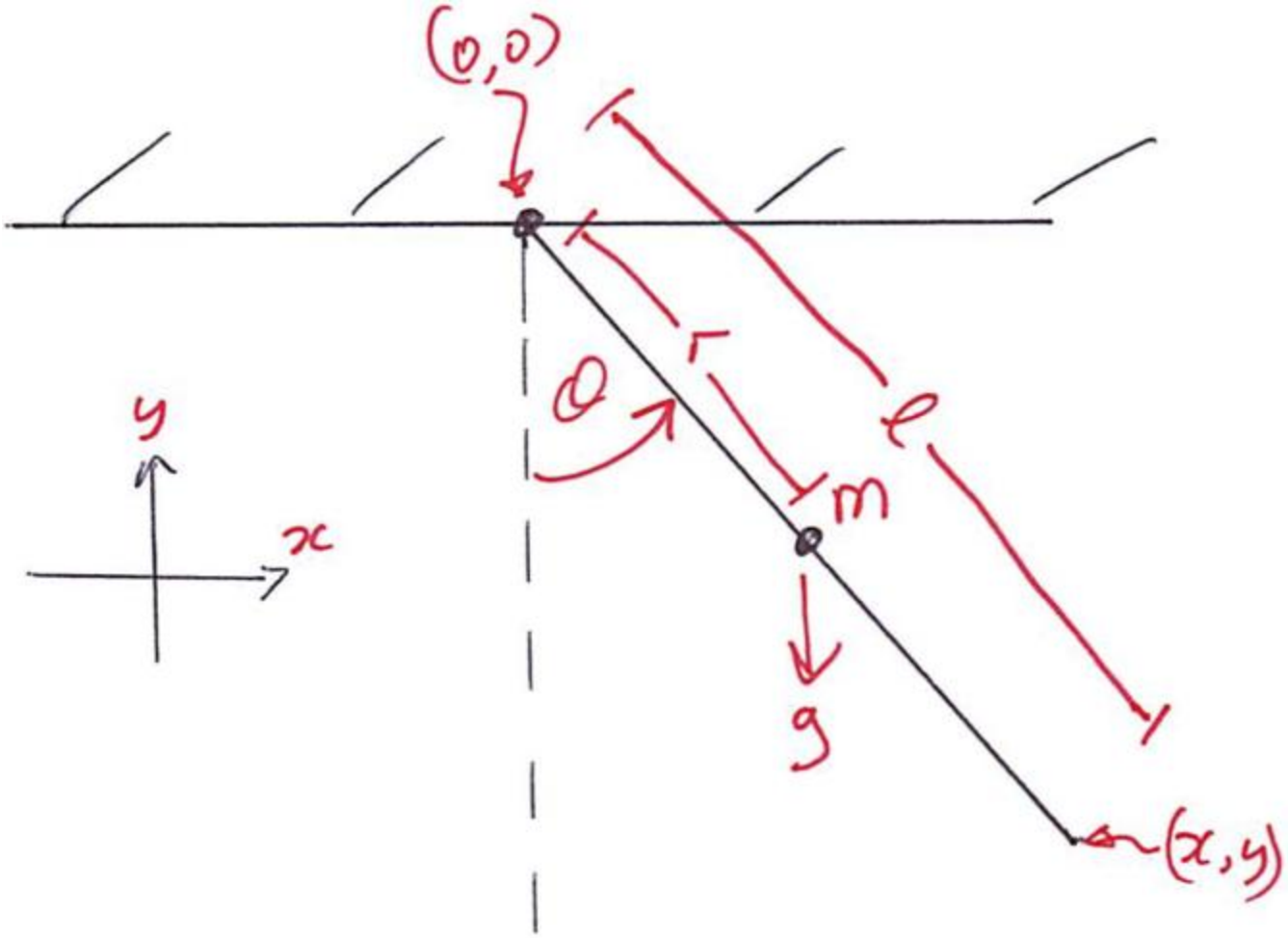
Computational Motor Control: Dynamics

Introduction

Here we will talk about deriving the **equations of motion** for a one-joint and then two-joint (planar) arm. We saw in earlier topics how to translate differential equations into Python code for simulation. Here we will be talking about how to derive the equations themselves. Then we will implement them in Python, and simulate the dynamical behaviour of the arm. For now we will still leave out muscles, and only talk about the dynamics of the arm linkage itself. Surprisingly, even a "simple" two-joint arm ends up having complex dynamical behaviour.

Equations of motion for a one-joint arm

Let's start with a simple one-joint planar arm (like an elbow joint), and let's put it in a vertical plane, with gravity acting down:



Schematic of a simple one-joint arm in a vertical plane

We have a single link of length l that is represented as a uniform rod of length l metres, with centre of mass m kg located r metres from the origin (about which the link rotates), and with rotational moment of inertia i (kgm2). The angle θ is the angle between the horizontal and the link, and we have the force of gravity g acting in the downward direction.

There are many ways of deriving the equations of motion of a dynamical system, including the Lagrangian, Hamiltonian, and Newton-Euler approaches. They are all different ways to get to the same place (for our purposes), the equations of motion relating the inputs (e.g. forces and torques) to the dynamical behaviour (accelerations, velocities, positions) of a system. We will look at the Lagrangian approach here. A great practical book for looking at all sorts of examples of the Lagrangian approach, for all sorts of mechanical systems, is Schaum's Outline: Lagrangian Dynamics.

Note: if you are squeamish about the calculus, or you're just not that interested in the nuts and bolts of *how* to derive equations of motion, that's OK … just skip ahead to hereand see what the final equation of motion looks like.

Optional: Euler-Lagrange Equation

The really cool thing about the Lagrangian approach is that you can derive equations of motion using only:

1. a characterization of the energy (kinetic & potential) of the system
2. a description of the kinematics of the system
3. running these through calculus

The Euler-Lagrange equation gives the "generalized" equations of motion as:

$$Qj = ddt(\partial L \partial qj^{\cdot}) - (\partial L \partial qj)$$

where j indexes into each "generalized coordinate", and where

$$L = T - U$$

and where T is the **kinetic energy** of the system, and U is the **potential energy** of the system.

The basic idea is once you derive an expression for L, then you just run it through the derivatives in the Euler-Lagrange equation, to get the equations of motion in terms of the "generalized forces" Qj. I know that sounds a bit opaque, so let's run through a concrete example with our single-joint pendulum.

Let's derive the two parts of L (T and U) and then we will see how to get the equations of motion by running L through the Euler-Lagrange equation to get generalized forces Qj.

Kinetic Energy

So first let's write down the kinetic energy T. There are two parts: the linear kinetic energy of the centre of mass m, and the rotational kinetic energy of the rod.

- Linear kinetic energy

We know that for a point mass (and we are representing our link's centre of mass as a point mass m kg located $l/2$ metres away from a rotary joint), the kinetic energy is $\frac{1}{2}mv^2$ where v is velocity:

$$T_{lin} = T_{lin} = T_{lin} = = = \frac{1}{2}mv^2 \frac{1}{2}m(\dot{x}^2 + \dot{y}^2) \frac{1}{2}m\dot{x}^2 + \frac{1}{2}m\dot{y}^2$$

- Rotational kinetic energy

  For rotation, kinetic energy is:

  $$T_{rot} = \frac{1}{2}i\dot{\theta}^2$$

  where i is the moment of inertia and $\dot{\theta}$ is the angular velocity.

  Putting them together, the total kinetic energy of the system is:

  $$T = \frac{1}{2}m\dot{x}^2 + \frac{1}{2}m\dot{y}^2 + \frac{1}{2}i\dot{\theta}^2$$

  We can transform the coordinates from extrinsic (x,y) (location of centre of mass) to intrinsic θ using the following relations, which we know from geometry:

  $$x y = = r\sin\theta - r\cos\theta$$

  so if we substitute and simplify terms (not shown) we get:

  $$T = \frac{1}{2}mr^2\dot{\theta}^2 + \frac{1}{2}i\dot{\theta}^2$$

Potential Energy

Now let's write down the potential energy U. We know from basic physics that in general, the potential energy for a mass in earth's gravitational field is:

$$U = mgh$$

where m is mass (kg), g is the gravitational constant (9.81 m/s) and h is the height "above the ground". So for our rotating arm, let's define the "zero potential energy point" (like the "ground") when the pendulum is pointing straight down, i.e. when $\theta = 0$:

$$U = mgr(1 - \cos\theta)$$

Lagrangian

So now we have the Lagrangian $L = T - U$ is:

$$L = T - U = \frac{1}{2} mr^2 \dot{\theta}^2 + \frac{1}{2} i \dot{\theta}^2 - mgr(1 - \cos\theta)$$

So to summarize, we have chosen our **generalized coordinates** to be in terms of the joint angle θ. Since we only have one degree of freedom in our system, in fact we only have one generalized coordinate, which is θ. The generalized forces Qj are just one, so we will write Q, and since we chose our generalized coordinate to be angular, θ, our generalized force Q is actually a torque (the rotational equivalent of a force).

Now it's a matter of computing the derivative terms in the Euler-Lagrange equation to get an expression giving the torque of the system in terms of the system states. You can do this by hand if you're a calculus ninja, or use a symbolic computing package like SymPy to do it for you.

$$\frac{\partial L}{\partial \theta} = -mgr\sin\theta$$

and

$$\frac{\partial L}{\partial \dot{\theta}} = \dot{\theta}(i + mr^2)$$

and

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) = \ddot{\theta}(i + mr^2)$$

and remember the Euler-Lagrange equation:

$$Qj = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{qj}}\right) - \left(\frac{\partial L}{\partial qj}\right)$$

so plugging in the values from the calculus/algebra above,

$$Q = \ddot{\theta}(i + mr^2) + mgr\sin\theta$$

So this is our **equation of motion**, it gives us a relationship between generalized force Q (which is a **torque**), and the **states of the system**, $(\theta, \dot{\theta}, \ddot{\theta})$ (note actually $\dot{\theta}$ doesn't appear in our equations of motion in this case).

If you want to see the SymPy code for doing all this calculus, it is here: onejoint_lagrange.py

Equation of motion

Our equation of motion:

$$Q = \ddot{\theta}(i + mr^2) + mgr\sin\theta$$

where Q is joint torque (Nm), m is link mass (kg), l is link length (m), g is gravitational constant (m/s/s) and θ is joint angle (radians), gives **joint torque as a function of state**.

This is actually the *inverse dynamics* equation. To say it differently, this is the equation we can use to answer the question, "what torque do I need at the joint (for example from muscles) in order to generate a given dynamic state?"

For forward simulation of a dynamical system, we need the *forward dynamics* equation of motion. In other words, we need an equation that gives the derivatives of the system state(s) as a function of the system states themselves (and any other quantities, e.g. joint torque).

We can easily solve our equation of motion for $\ddot{\theta}$:

$$\ddot{\theta} = Q - mgr\sin\theta i + mr^2$$

Note that if the torque Q is zero, in other words if there is no **input** torque to the system, then:

$$\ddot{\theta} = -mgr\sin\theta i + mr^2$$

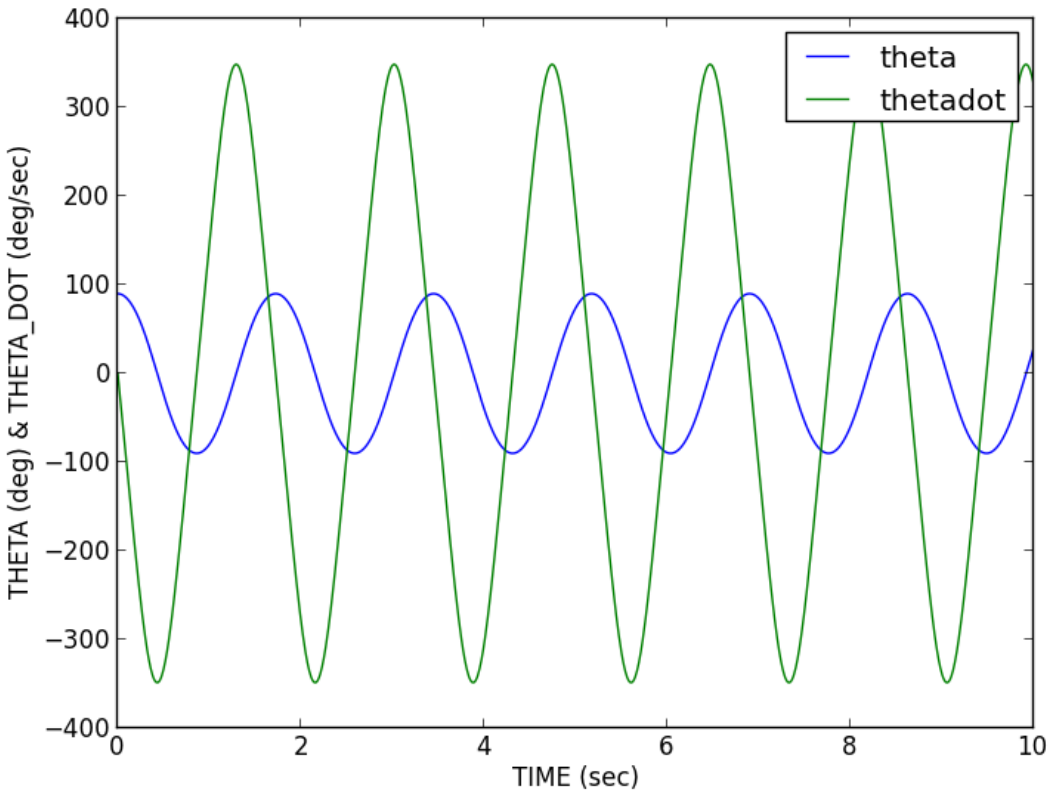
This characterizes the passive dynamics of the system (the dynamical behaviour of the system in the absence of any external driving torque).

Simulating the dynamics of the one-joint arm

Let's write a function, as we did earlier in the course for other systems, for the forward dynamics of our one-joint arm:

```
from scipy.integrate import odeint
def onejointarm(state,t):
```

```
  theta = state[0]      # joint angle (rad)
  theta_dot = state[1]  # joint velocity (rad/s)
  m = 1.65              # kg
  r = 0.50              # link length (m)
  g = 9.81              # gravitational constant (m/s/s)
  i = 0.025             # moment of inertia (kg m m)
  theta_ddot = -(m*g*r*sin(theta)) / (i + (m*r*r))
  return [theta_dot, theta_ddot]

t = linspace(0.0,10.0,1001)   # 10 seconds sampled at 1000 Hz
state0 = [90.0*pi/180.0, 0.0] # 90 deg initial angle, 0 deg/sec initial velocity
state = odeint(onejointarm, state0, t)

figure()
plot(t,state*180/pi)
legend(('theta','thetadot'))
xlabel('TIME (sec)')
ylabel('THETA (deg) & THETA_DOT (deg/sec)')
```

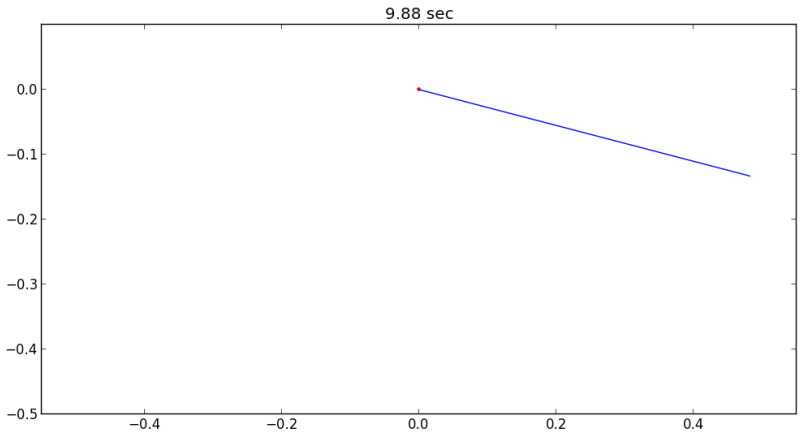


Dynamics of passive one-joint arm

Here's a little function that will animate the arm:

```
def animate_arm(state,t):
  l = 0.5
  figure(figsize=(12,6))
  plot(0,0,'r.')
  p, = plot((0,l*sin(state[0,0])),(0,-l*cos(state[0,0])),'b-')
  tt = title("%4.2f sec" % 0.00)
  xlim([-l-.05,l+.05])
  ylim([-l,.10])
  step = 3
  for i in xrange(1,shape(state)[0]-10,step):
    p.set_xdata((0,l*sin(state[i,0])))
    p.set_ydata((0,-l*cos(state[i,0])))
    tt.set_text("%4.2f sec" % (i*0.01))
```

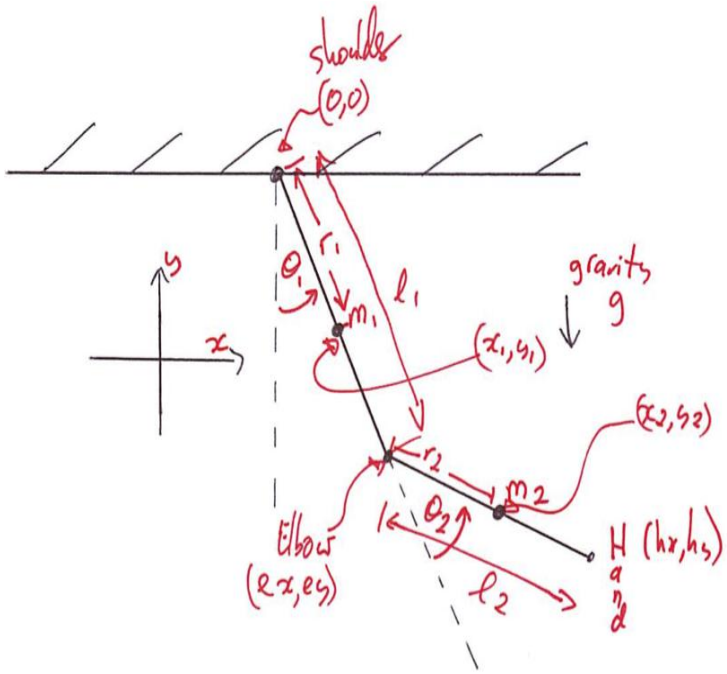
```
    draw()

animate_arm(state,t)
```



Animation of passive one-joint arm

Equations of motion for a two-joint arm

To derive the equations of motion for a two-jointarm arm we will follow the same basic steps as above for the one-joint arm. Note again that we are now putting the arm in a vertical plane, with gravity pointing down.



Schematic of a two-joint arm in a vertical plane

Now we have two links of length $l1$ and $l2$ metres, each represented as a uniform rod of mass $m1$ and $m2$ kg, with the centres of mass located $r1$ and $r2$ metres from the points of rotation. Moments of inertia (not shown on figure) are $i1$ and $i2$. The shoulder joint is located at the origin, (0,0) metres, the elbow joint E at (ex,ey) and the hand H at (hx,hy). Gravity g is pointing "down" $(-y)$ and joint angles $(\theta1,\theta2)$ are as indicated.

Note If you want to skip over the Lagrangian formulation you are welcome to, just skip right here to the equations of motion.

Optional: The Lagrangian

As before we will be using the Euler-Lagrange equation to derive the equations of motion:

$$Qj = ddt(\partial L \partial qj^{\cdot}) - (\partial L \partial qj)$$

where $j=1,2$ (see below) and where

$$L = T - U$$

Here we will have two generalized coordinates $\theta 1$ and $\theta 2$, and so our generalized forces $Q1$ and $Q2$ will correspond to shoulder torque and elbow torque, respectively.

Again we must write expressions for linear and rotational kinetic energy.

Linear kinetic energy

In general,

$$Tlinj = 12 mjv2j$$

for $j = 1,2$. Expanding vj:

$$Tlin1\ Tlin2 == 12 m1(x1\dot{}2 + y1\dot{}2)12 m2(x2\dot{}2 + y2\dot{}2)$$

Rotational kinetic energy

For rotation, kinetic energy is:

$$Trotj = 12 ij\theta j\dot{}2$$

so

$$Trot1\ Trot2 == 12 i1\,\theta 1\dot{}212 i2(\theta 1\dot{} + \theta 2\dot{})2$$

We can transform the coordinates from external (x,y) cartesian coordinates into our chosen (intrinsic, joint-based) generalized coordinate frame $(\theta 1, \theta 2)$ based on the following relations from geometry (our forward kinematic equations):

$$x1\ y1\ x2\ y2 = = = = r1\sin\theta 1 - r1\cos\theta 1\ /1\sin\theta 1 + r2\sin(\theta 1 + \theta 2) - l1\cos\theta 1 - r2\cos(\theta 1 + \theta 2)$$

Total kinetic energy is then:

$$T = Tlin1 + Tlin2 + Trot1 + Trot2$$

Potential energy

Just as for the onejoint arm we have potential energy of each link.

$$U1\ U2 == m1\,gr1(1 - \cos\theta 1)\,m2\,g(l1(1 - \cos\theta 1) + r2(1 - \cos(\theta 1 + \theta 2)))$$

Lagrangian

We then define the lagrangian L as

$$LL == T - UTlin1 + Tlin2 + Trot1 + Trot2 - U1 - U2$$

To get the equations of motion we then simply evaluate the Euler-Lagrange equation, once for each generalized force (torque):

$$Q1\ Q2 == ddt(\partial L \partial \theta 1\ \dot{}) - (\partial L \partial \theta 1)\ ddt(\partial L \partial \theta 2\ \dot{}) - (\partial L \partial \theta 2)$$

For those that are interested, here is a SymPy program that implements the Lagrangian approach to get the equations of motion for a two-joint arm:

twojointarm_lagrange.py

Equations of Motion

What we end up with (I won't take you step by through all the calculus) is two equations of motion, one for shoulder torque $Q1$ and one for elbow torque $Q2$.

$$Q1\ Q2 == M11\ \theta 1\ \ddot{} + M12\ \theta 2\ \ddot{} + C1 + G1\ M21\ \theta 1\ \ddot{} + M22\ \theta 2\ \ddot{} + C2 + G2$$

where

$$M11\ M12\ M22 == M21 == i1 + i2 + m1\ r21 + m2(l21 + r22 + 2\ l1\ r2\cos\theta 2)\ i2 + m2(r22 + l1\ r2\cos\theta 2)\ i2 + m2\ r22$$

and

$$C1\ C2 == -m2\ l1\ \theta 2\ \dot{}\ 2\ r2\sin\theta 2 - 2\ m2\ l1\ \theta 1\ \dot{}\ \theta 2\ \dot{}\ r2\sin\theta 2\ m2\ l1\ \theta 1\ \dot{}\ 2\ r2\sin\theta 2$$

and

$$G1\ G2 == g\sin\theta 1\ (m2\ l1 + m1\ r1) + gm2\ r2\sin(\theta 1 + \theta 2)\ gm2\ r2\sin(\theta 1 + \theta 2)$$

The M terms you can think of as inertial terms (they depend on joint accelerations). The C terms are usually called *Coriolis-centrifugal* terms, and the G terms are the terms due to gravity.

Joint Interaction Torques

Notice something interesting about these two-joint arm equations of motion: The torque at the shoulder $Q1$ depends not just on shoulder acceleration $\theta 1^{\cdot\cdot}$ but also on elbow joint acceleration $\theta 2^{\cdot\cdot}$. Similarly, elbow joint torque $Q2$ depends not just on elbow joint acceleration $\theta 2^{\cdot\cdot}$ but also on shoulder joint acceleration $\theta 1^{\cdot\cdot}$. These are inertial interaction torques.

If you look at the Coriolis/centrifugal terms C you also see a similar pattern but for velocities. Shoulder torque $Q1$ depends (via $C1$) on *elbow velocity* $\theta 2^{\cdot}$ and on the product of shoulder and elbow velocities $\theta 1^{\cdot}\theta 2^{\cdot}$. Elbow torque $Q2$ depends (via $C2$) on shoulder velocity squared $\theta 1^{\cdot}2$. These are Coriolis-centrifugal interaction torques.

So when torque at one joint depends on velocities and/or accelerations at another joint, we call these effects **joint interaction torques**. These interaction torques may be large, and significantly affect limb movement, especially when velocities and/or accelerations are large.

Simulating the dynamics of the two-joint arm

The equation of motion above for the two joint arm are inverse dynamics equations: they give the shoulder and elbow joint torques required (for example by muscles) to generate a particular arm kinematic trajectory.

To get the forward dynamics equations of motion, we just need to do a little bit of algebra. Let's first write our inverse dynamics equations from above, in matrix form:

$$Q=M\theta^{\cdot\cdot}+C+G$$

where

$$M\theta^{\cdot\cdot}\,CG====[M11\,M21\,M12\,M22][\theta 1^{\cdot\cdot}\,\theta 2^{\cdot\cdot}][C1\,C2][G1\,G2]$$

Now to change our equation into a forward dynamics equation, we simply need to solve for $\theta^{\cdot\cdot}$:

$$\theta^{\cdot\cdot}=(M-1)(Q-C-G)$$

and for a passive arm with no external (driving) torques (e.g. from muscles) this simplifies to:

$$\theta^{\cdot\cdot}=(M-1)(-C-G)$$

Here is a Python function that implements the forward dynamics equations of a passive two-joint arm:

```
def twojointarm(state,t,aparams):
    """
    passive two-joint arm in a vertical plane
    X is fwd(+) and back(-)
    Y is up(+) and down(-)
    gravity acts down
    shoulder angle a1 relative to Y vert, +ve counter-clockwise
    elbow angle a2 relative to upper arm, +ve counter-clockwise
    """
    a1,a2,a1d,a2d = state
    l1,l2 = aparams['l1'], aparams['l2']
    m1,m2 = aparams['m1'], aparams['m2']
    i1,i2 = aparams['i1'], aparams['i2']
    r1,r2 = aparams['r1'], aparams['r2']
    g = 9.81
    M11 = i1 + i2 + (m1*r1*r1) + (m2*((l1*l1) + (r2*r2) + (2*l1*r2*cos(a2))))
    M12 = i2 + (m2*((r2*r2) + (l1*r2*cos(a2))))
    M21 = M12
    M22 = i2 + (m2*r2*r2)
    M = matrix([[M11,M12],[M21,M22]])
    C1 = -(m2*l1*a2d*a2d*r2*sin(a2)) - (2*m2*l1*a1d*a2d*r2*sin(a2))
    C2 = m2*l1*a1d*a1d*r2*sin(a2)
    C = matrix([[C1],[C2]])
    G1 = (g*sin(a1)*((m2*l1)+(m1*r1))) + (g*m2*r2*sin(a1+a2))
    G2 = g*m2*r2*sin(a1+a2)
    G = matrix([[G1],[G2]])
    ACC = inv(M) * (-C-G)
    a1dd,a2dd = ACC[0,0],ACC[1,0]
    return [a1d, a2d, a1dd, a2dd]
```

Here is a full python program that will do a forward simulation of our passive two-joint arm, starting from specified initial shoulder and elbow joint angles and velocities. It will also show a rudimentary animation of the resulting motion of the arm.

twojointarm_passive.py

Does the brain know about interaction torques? (yes)

Joint interaction torques can be large and can significantly affect limb motion.

- Hollerbach, J. M., & Flash, T. (1982). Dynamic interactions between limb segments during planar arm movement. Biological cybernetics, 44(1), 67-77.

There is empirical evidence from studies of human arm movement that the CNS neurally represents these effects, and compensates for them in a predictive manner, when planning and controlling arm movements:

- Hasan, Z., & Karst, G. M. (1989). Muscle activity for initiation of planar, two-joint arm movements in different directions. Experimental Brain Research, 76(3), 651-655.
- Virji-Babul, N., & Cooke, J. D. (1995). Influence of joint interactional effects on the coordination of planar two-joint arm movements. Experimental brain research, 103(3), 451-459.
- Sainburg, R. L., Ghilardi, M. F., Poizner, H. & Ghez, C. (1995). Control of limb dynamics in normal subjects and patients without proprioception. Journal of Neurophysiology, 73(2), 820-835.
- Gribble, P. L., & Ostry, D. J. (1999). Compensation for interaction torques during single-and multijoint limb movement. Journal of Neurophysiology, 82(5), 2310-2326.
- Sainburg, R. L., Ghez, C., & Kalakanis, D. (1999). Intersegmental dynamics are controlled by sequential anticipatory, error correction, and postural mechanisms. Journal of Neurophysiology, 81(3), 1045-1056.
- Koshland, G. F., Galloway, J. C., & Nevoret-Bell, C. J. (2000). Control of the wrist in three-joint arm movements to multiple directions in the horizontal plane. Journal of neurophysiology, 83(5), 3188-3195.

Kinematic vs Dynamic models of motor control

We saw in the previous topic about kinematic models of movement (e.g. minimum-jerk) that computational models of movement at the kinematic level are capable of predicting a range of characteristics of naturalistic voluntary arm movements.

Behavioural studies

Here are two examples of behavioural studies looking at the question of whether the brain plans arm movements in terms of kinematics or dynamics.

- Flanagan, J. R., & Rao, A. K. (1995). Trajectory adaptation to a nonlinear visuomotor transformation: evidence of motion planning in visually perceived space. Journal of neurophysiology, 74(5), 2174-2178.
- Wolpert, D. M., Ghahramani, Z., & Jordan, M. I. (1995). Are arm trajectories planned in kinematic or dynamic coordinates? An adaptation study. Experimental brain research, 103(3), 460-470.

Computational models of Dynamics planning

Here are examples of computational models that propose dynamic variables (e.g. torques) are used to plan arm movements.

- Uno, Y., Kawato, M., & Suzuki, R. (1989). Formation and control of optimal trajectory in human multijoint arm movement. Biological cybernetics, 61(2), 89-101.
- Nakano, E., Imamizu, H., Osu, R., Uno, Y., Gomi, H., Yoshioka, T., & Kawato, M. (1999). Quantitative examinations of internal representations for arm trajectory planning: minimum commanded torque change model. Journal of Neurophysiology, 81(5), 2140-2155.
- Kawato, M. (1999). Internal models for motor control and trajectory planning. Current opinion in neurobiology, 9(6), 718-727.
- Wolpert, D. M., & Kawato, M. (1998). Multiple paired forward and inverse models for motor control. Neural Networks, 11(7), 1317-1329.
- Wolpert, D. M., Miall, R. C., & Kawato, M. (1998). Internal models in the cerebellum. Trends in cognitive sciences, 2(9), 338-347.
- Haruno, M., Wolpert, D. M., & Kawato, M. (2001). Mosaic model for sensorimotor learning and control. Neural computation, 13(10), 2201-2220.

A cool paper on insect flight:

- Berman, G. J., & Wang, Z. J. (2007). Energy-minimizing kinematics in hovering insect flight. Journal of Fluid Mechanics, 582(1), 153-168.

Electrophysiological studies

Empirical studies after the Georgopoulos series of papers investigated more systematically the question of kinematic- versus dynamic planning of arm movements, and the question of what motor brain areas represent. What investigators found is that in fact the activity of motor cortex cells is modulated by all sorts of "intrinsic" variables like background level of load force (e.g. mechanical loads) and arm orientation.

- Kalaska, J. F., Cohen, D. A., Hyde, M. L., & Prud'Homme, M. (1989). A comparison of movement direction-related versus load direction-related activity in primate motor cortex, using a two-dimensional reaching task. The Journal of neuroscience, 9(6), 2080-2102.
- Scott, S. H., & Kalaska, J. F. (1997). Reaching movements with similar hand paths but different arm orientations. I. Activity of individual cells in motor cortex. Journal of Neurophysiology, 77(2), 826-852.
- Scott, S. H., Sergio, L. E., & Kalaska, J. F. (1997). Reaching movements with similar hand paths but different arm orientations. II. Activity of individual cells in dorsal premotor cortex and parietal area 5. Journal of neurophysiology, 78(5), 2413-2426.
- Gandolfo, F., Li, C. S., Benda, B. J., Schioppa, C. P., & Bizzi, E. (2000). Cortical correlates of learning in monkeys adapting to a new dynamical environment. Proceedings of the National Academy of Sciences, 97(5), 2259-2263.
- Cabel, D. W., Cisek, P., & Scott, S. H. (2001). Neural activity in primary motor cortex related to mechanical loads applied to the shoulder and elbow during a postural task. Journal of neurophysiology, 86(4), 2102-2108.
- Scott, S. H., Gribble, P. L., Graham, K. M., & Cabel, D. W. (2001). Dissociation between hand motion and population vectors from neural activity in motor cortex. Nature, 413(6852), 161-164.
- Li, C. S. R., Padoa-Schioppa, C., & Bizzi, E. (2001). Neuronal correlates of motor performance and motor learning in the primary motor cortex of monkeys adapting to an external force field. Neuron, 30(2), 593-607.
- Gribble, P. L., & Scott, S. H. (2002). Overlap of internal models in motor cortex for mechanical loads during reaching. Nature, 417(6892), 938-941.
- Gandolfo, F., Li, C. S., Benda, B. J., Schioppa, C. P., & Bizzi, E. (2000). Cortical correlates of learning in monkeys adapting to a new dynamical environment. Proceedings of the National Academy of Sciences, 97(5), 2259-2263.

Some argue for a mixed representation in primary motor cortex.

- Kakei, S., Hoffman, D. S., & Strick, P. L. (1999). Muscle and movement representations in the primary motor cortex. Science, 285(5436), 2136-2139.

Other motor brain areas appear to have different balance of kinematics vs dynamics representation

- Thach, W. T. (1978). Correlation of neural discharge with pattern and force of muscular activity, joint position, and direction of intended next movement in motor cortex and cerebellum. Journal of neurophysiology, 41(3), 654-676.
- Fortier, P. A., Kalaska, J. F., & Smith, A. M. (1989). Cerebellar neuronal activity related to whole-arm reaching movements in the monkey. Journal of neurophysiology, 62(1), 198-211.
- Kalaska, J. F., Cohen, D. A. D., Prud'Homme, M., & Hyde, M. L. (1990). Parietal area 5 neuronal activity encodes movement kinematics, not movement dynamics. Experimental Brain Research, 80(2), 351-364.

Some review articles on the topic:

- Georgopoulos, A. P. (1995). Current issues in directional motor control. Trends in neurosciences, 18(11), 506-510.
- Kalaska, J. F., Scott, S. H., Cisek, P., & Sergio, L. E. (1997). Cortical control of reaching movements. Current opinion in neurobiology, 7(6), 849-859.
- Scott, S. H. (2003). The role of primary motor cortex in goal-directed movements: insights from neurophysiological studies on non-human primates. Current opinion in neurobiology, 13(6), 671-677.

Computational Models: Cortical control

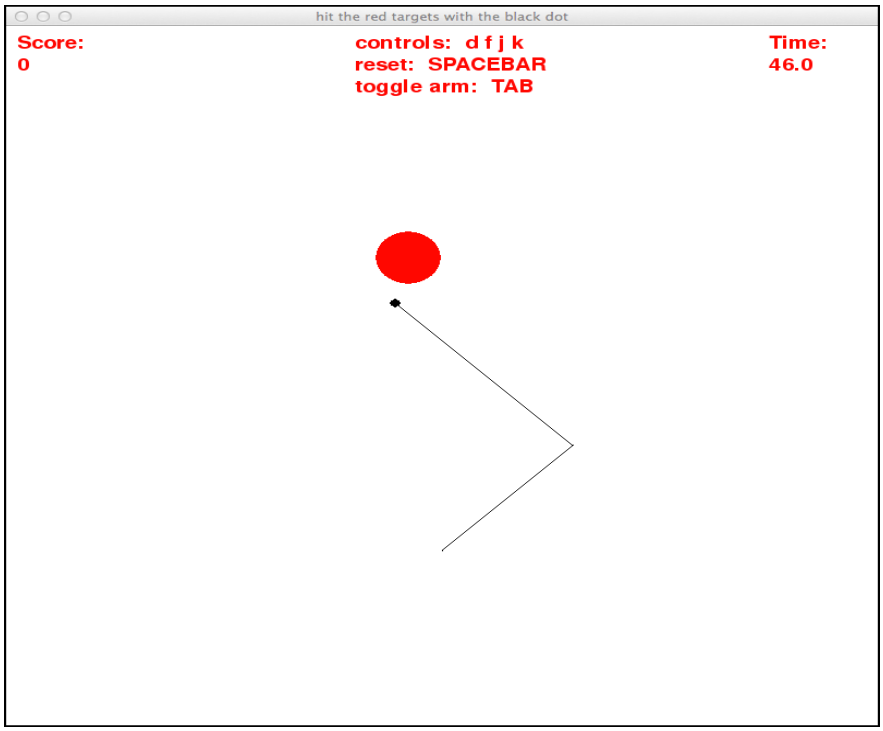
A number of computational models have been described to explore how motor cortex controls arm movement. One of the common themes has been that models using motor cortex neurons to directly code muscle activation also predict population-vector coding of hand direction (and other dynamics parameters too).

- Todorov, E. (2000). Direct cortical control of muscle activation in voluntary arm movements: a model. nature neuroscience, 3, 391-398.
- Guigon, E., Baraduc, P., & Desmurget, M. (2007). Coding of movement-and force-related information in primate primary motor cortex: a computational approach. European Journal of Neuroscience, 26(1), 250-260.

- Ajemian, R., Green, A., Bullock, D., Sergio, L., Kalaska, J., & Grossberg, S. (2008). Assessing the function of motor cortex: single-neuron models of how neural response is modulated by limb biomechanics. Neuron, 58(3), 414-428.

Two-joint arm video game

I coded up a little "game" that let's you control a two-joint arm with realistic limb dynamics, to hit targets that appear in the arm's workspace (not unlike many actual motor control experiments).



Two-joint arm video game

You use the [d,f,j,k] keys to control 4 "muscles" that deliver torques to the shoulder and elbow:

- d = pectoralis (shoulder flexor)
- f = posterior deltoid (shoulder extensor)
- j = biceps (elbow flexor)
- k = triceps (elbow extensor)

there is no real muscle model right now, they just deliver torques to the joints

The goal is to move the arm around the workspace so that the endpoint (hand) hits the red targets as they pop up. Hit as many red targets as you can before the clock runs down.

Hit the spacebar to "reset" the arm to it's home position. This is handy in case your arm starts spinning like a propellor ;) although each spacebar-reset costs you 1 point.

Apart from being a challenging "game", this is a nice little toy demo of the problem of what the brain is faced with when it has to figure out what time-varying neural control signals to send down to muscles, so that your hand moves to a desired location in cartesian space.

Here is the game: twojointarm_game.py

You will need to install the pygame add-on package for python. On a Debian-based GNU/Linux system (like Ubuntu) just type at a command-line prompt:

sudo apt-get install python-pygame

On Mac/Windows etc, see the pygame homepage for installation instructions.

To start the game, at a command-line prompt type:

python twojointarm_game.py

At the end of the game it will print out your score to the command line.

[Computational Motor Control: Muscle Models]

Source:
http://www.gribblelab.org/compneuro2012/5_Computational_Motor_Control_Dynamics.html